The Mitnick Attack Lab

1 Overview

Kevin Mitnick is probably one of the most well-known hackers in USA. He was on FBI's wanted list of criminals. While on the run, he became interested in hacking cellular phone networks, and was in need for specialized software that could help him do that. That led him to Tsutomu Shimomura, a researcher working at the San Diego Supercomputer Center, who was one of the leading researchers on the security of cellular phone networks. He had the code that Mitnick wanted.

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 triggered a dramatic showdown between the two people, and it eventually led to the arrest of Mitnick. The showdown was turned into books and Hollywood movies later. The attack is now known as the Mitnick attack, which is a special type of TCP session hijacking.

The objective of this lab is to recreate the classic Mitnick attack, so students can gain the first-hand experience on such an attack. We will emulate the settings that was 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. This lab covers the following topics:

- TCP session hijacking attack
- TCP three-way handshake protocol
- The Mitnick attack
- Remote shell `rsh`
- Packet sniffing and spoofing

Readings and videos. Detailed coverage of the TCP session hijacking can be found in the following:

- Chapter 16 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.
- Section 6 of the SEED Lecture, *Internet Security: A Hands-on Approach*, by Wenliang Du. See details at `https://www.handsonsecurity.net/video.html`.

Lab environment. This lab has been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website.

2 How the Mitnick Attack Works

The Mitnick attack is a special case of TCP session hijacking attacks. 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, host A was called X-Terminal, which was the target. Mitnick wanted to log into X-Terminal and run his commands on it. Host B was a trusted server, which 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. There are four primary steps in this attack.

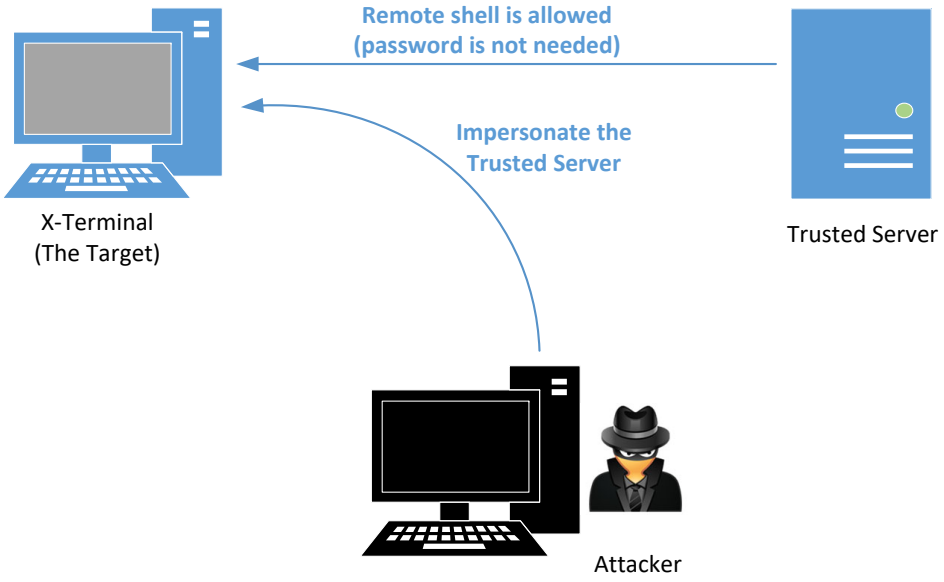

Figure 1: The illustration of the Mitnick Attack

Step 1: Sequence number prediction. Before the attack, Mitnick needed to learn the pattern of the initial sequence numbers (ISN) on X-terminal (in those days, ISNs were not random). Mitnick sent SYN requests to X-terminal and received SYN+ACK responses, then he sent RESET packet to X-Terminal, to clear the half-open connection from X-Terminal's queue (to prevent the queue from being filled up). After repeating this for twenty times. He found there was a pattern between two successive TCP ISNs. This allowed Mitnick to predict ISNs, which was essential for the attack.

Step 2: SYN flooding attack on the trusted server. To send a connection request from the trusted server to X-Terminal, Mitnick needed to send out a SYN packet from the trusted server to X-Terminal. X-Terminal would respond with a 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.

To solve this problem. Mitnick had to silence the trusted server. Therefore, before spoofing, Mitnick launched a SYN flooding attack on the server. Back then, operating systems were far more vulnerable to the SYN flooding attack. The attack could actually shut down the trusted computer, completely silencing it.

Step 3: Spoofing a TCP connection. Mitnick wanted to use `rsh` (remote shell) to run a backdoor command on X-Terminal; once the backdoor was setup, he could then log into X-Terminal. To run a remote shell on X-Terminal, Mitnick needed to pass the authentication, i.e, he needed to have a valid account on X-Terminal and know its password. Obviously, he did not have that.

Shimomura often needed to log into X-Terminal from the trusted server. To avoid typing passwords each time, he added some information in the `.rhosts` file on X-Terminal, so when he logged into X-Terminal from the trusted server, no password would be asked. This was quite a common practice back then. With this setup, without typing any password, Shimomura could run a command on X-Terminal from the trusted server using `rsh`, or run `rlogin` to log into X-Terminal. Mitnick wanted to exploit this trusted relationship.

He needed to create a TCP connection between the trusted server and X-Terminal, and then run `rsh` inside this connection. He first sent a SYN request to X-Terminal, using the trusted server's IP as the source IP address. X-Terminal then sent a SYN+ACK response to the server. Since the server had been shut down, it would not send RESET to close the connection.

To complete the three-way handshake protocol, Mitnick needed to spoof an ACK packet, which must acknowledge the sequence number in X-Terminal's SYN+ACK packet. Unfortunately, the SYN+ACK response only went to the trusted server, not to Mitnick, he could not see the sequence number. However, because of the prior investigation, Mitnick was able to predict what this number was, so he was able to successfully spoof the ACK response sent to X-Terminal to complete the TCP three-way handshake.

Step 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. Using this command, Mitnick wanted to create a backdoor on X-Terminal so that he could get a shell on X-Terminal anytime without repeating the attack.

All he needed to do was to add `"+ +"` to the `.rhosts` file on X-Terminal. He could achieve that by executing the following command using `rsh` on X-Terminal: `"echo + + > .rhosts"`. Since `rsh` and `rlogin` program used `.rhosts` file for authentication, with this addition, X-Terminal would trust every `rsh` and `rlogin` request from anyone.

3 Lab Environment Setup Using Container

3.1 Container Setup

In this lab, we need three machines, one for X-Terminal, one for Trusted Server, and the other for the attacker. In the real Mitnick attack, the attacker machine is a remote machine. In this lab, for the sake of simplicity, we put all these three machines on the same network. Students can use three virtual machines for this lab, but it will be much more convenient to use containers. The lab environment is depicted Figure 2.

3.2 Container Setup and Commands

Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

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

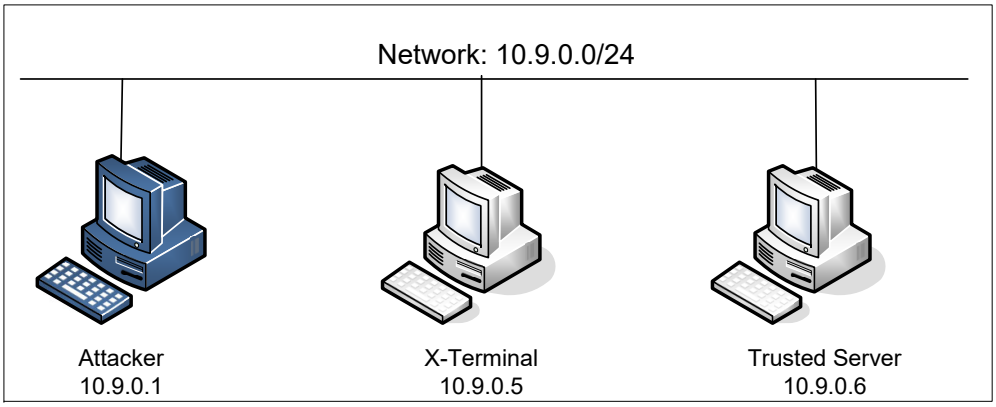

Figure 2: Lab environment setup

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

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

3.3 About the Attacker Container

In this lab, we can either use the VM or the attacker container as the attacker machine. If you look at the Docker Compose file, you will see that the attacker container is configured differently from the other containers. Here are the differences:

- *Shared folder.* When we use the attacker container to launch attacks, we need to put the attacking code inside the attacker container. Code editing is more convenient inside the VM than in containers, because we can use our favorite editors. In order for the VM and container to share files, we have created a shared folder between the VM and the container using the Docker `volumes`. If you look at the Docker Compose file, you will find out that we have added the following entry to some of the containers. It indicates mounting the `./volumes` folder on the host machine (i.e., the VM) to the `/volumes` folder inside the container. We will write our code in the `./volumes` folder (on the VM), so they can be used inside the containers.

```
volumes:
       - ./volumes:/volumes
```

- *Host mode.* In this lab, the attacker needs to be able to sniff packets, but running sniffer programs inside a container has problems, because a container is effectively attached to a virtual switch, so it can only see its own traffic, and it is never going to see the packets among other containers. To solve this problem, we use the `host` mode for the attacker container. This allows the attacker container to see all the traffics. The following entry used on the attacker container:

```
network_mode: host
```

  When a container is in the `host` mode, it sees all the host's network interfaces, and it even has the same IP addresses as the host. Basically, it is put in the same network namespace as the host VM. However, the container is still a separate machine, because its other namespaces are still different from the host.

- *Privileged mode.* To be able to modify kernel parameters at runtime (using `sysctl`), such as enabling IP forwarding, a container needs to be privileged. This is achieved by including the following entry in the Docker Compose file for the container.

```
privileged: true
```

3.4 Installing the `rsh` program (no action is needed)

The remote shell `rsh` is a command line program that can execute shell commands remotely. Although we will use `rsh` in this task, we should know that `rsh` and `rlogin` programs are not secure, and they are not used any more. They have been replaced by more secured programs, such as `ssh`. That is why in the modern Linux operating systems, the `rsh` command is actually a symbolic link to the `ssh` program.

```
$ ls -al /etc/alternatives | grep rsh
lrwxrwxrwx   1 root root    12 Jul 25  2017 rsh -> /usr/bin/ssh
```

To recreate the Mitnick attack, we need to install the unsecure version of the `rsh` program. 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 can use the following commands to install `rsh` server and client. **Note:** The `rsh` programs are already installed in the X-Terminal and Trusted Server containers (see the `Dockerfile` inside the container image folder).

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

3.5 Configuration

The `rsh` server program uses two files 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. Note that the `.rhosts` file must reside at the top level of a user's home directory and can be written **only by the owner/user**.

Please use the following commands on X-Terminal to set up the `.rhosts` file. It should be noted when we get into a container, we will be in the root account. In this lab, we need to switch to a normal user account called seed, which is already created inside the container:

```
# su seed          ← Switch to the seed account
$ cd               ← Go to seed's home directory
$ touch .rhosts    ← Create an empty file
$ echo [Server's IP address] > .rhosts
$ chmod 644 .rhosts
```

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

```
# su seed          ← Switch to the seed account
$ rsh [X-Terminal's IP] date
```

If the command prints the current date and time, your configuration is working now. If you see "Authentication Failure", something in your setup may not be correct. One of the common mistakes is the permission on the `.rhosts` file: you should make sure it is only writable to the owner.

Allow all. To allow users to execute commands on X-Terminal from all IP addresses, we just need to put two plus signs (`"+ +"`) in the `.rhosts` file. This is very dangerous, and nobody should do that. But if you are an attacker, this is a convenient way to set up a backdoor. As we have mentioned before, this is what has been used in the Mitnick attack.

4 Task 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. We will simulate this effect.

We can manually stop the trusted server container, but that is not enough. When X-Terminal receives a SYN packet from the trusted server, it will respond with a 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, before silencing the trusted server, we could simply spoof an ICMP echo request from the trusted

server to X-Terminal, that would trigger X-Terminal to reply to the trusted server, and hence would get the trusted server's MAC address, and save it to the cache.

To simplify the task, before stopping the trusted server, we will 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. It should be noted that cache entry may be deleted by the operating system if the OS fails to reach a destination using the cached MAC address. To simply your attack, you can run the following command on X-Terminal to permanently add an entry to the ARP cache (it needs to run in the root account):

```
# arp -s [Server's IP] [Server's MAC]
```

5 Task 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.

One of the difficulties in the Mitnick attack is to predict the TCP sequence numbers. It was possible back then when TCP sequence numbers were not randomized. However, modern operating systems now randomize their TCP sequence numbers (as a countermeasure against TCP session hijacking attacks), so predicting the numbers becomes infeasible. To simulate the situation of the original Mitnick attack, we allow students to sniff packets, so they can get the sequence numbers, instead of guessing them.

Restriction. To simulate the original Mitnick attack as closely as we can, even though students can sniff the TCP packets from X-Terminal, they cannot use all the fields in captured packets, because in the real attacks, Mitnick could not sniff packets. When students write their attack programs, they can only use the following fields from the captured packets. Penalty will be applied if other fields are used.

- **The TCP sequence number field** (this does not include the acknowledgment field).
- **The TCP flag field**. This allows us to know the types of the captured TCP packets. In the actual Mitnick attack, Mitnick knew exactly what type of packets were sent out by X-Terminal, because they are part of the TCP three-way handshake protocol. We allow students to use this field for task simplification.
- **All the length fields**, including IP header length, IP total length, and TCP header length. These pieces of information are not necessary for the attacks. In the actual Mitnick attack, Mitnick knew exactly what their values are. We allow students to use these fields for task simplification.

The behavior of `rsh`. To create a spoofed `rsh` session between the trusted server and X-Terminal, we need to understand the behavior of `rsh`. Let us start a `rsh` session from Trusted Server to X-Terminal, and then use Wireshark to capture the packets between them (note: we will run Wireshark on the attacker VM; make sure to select the correct network interface corresponding to the `10.9.0.0/24` network). We use the following command to run the `date` command on Host B from Host A via the `rsh` remote shell.

```
// On Trusted Server
$ rsh 10.9.0.5 date
```

The packet trace in this `rsh` session is shown in the following. Here `10.9.0.6` is the Trusted Server's IP address, and `10.9.0.5` is X-Terminal's IP address. If a packet does not carry any TCP data, the length information (i.e. `Len=0`) is omitted.

Listing 1: Packet trace of a `rsh` session

```
# The first connection
   SRC IP    DEST IP   TCP Header
1  10.9.0.6  10.9.0.5  1023 -> 514 [SYN] Seq=778933536
2  10.9.0.5  10.9.0.6  514 -> 1023 [SYN,ACK] Seq=10879102 Ack=778933537
3  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103
4  10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933537 Ack=10879103 Len=20
                       RSH Session Establishment
                       Data: 1022\x00seed\x00seed\x00date\x00
5  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557

# The second connection
6  10.9.0.5  10.9.0.6  1023 -> 1022 [SYN] Seq=3920611526
7  10.9.0.6  10.9.0.5  1022 -> 1023 [SYN,ACK] Seq=3958269143 Ack=3920611527
8  10.9.0.5  10.9.0.6  1023 -> 1022 [ACK] Seq=3920611527 Ack=3958269144


# Going back to the first connection
9  10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879103 Ack=778933557 Len=1
                       Data: \x00
10 10.9.0.6  10.9.0.5  1023 -> 514 [ACK] Seq=778933557 Ack=10879104
11 10.9.0.5  10.9.0.6  514 -> 1023 [ACK] Seq=10879104 Ack=778933557 Len=29
                       Data: Sun Feb 16 13:41:17 EST 2020
```

We can observe that a `rsh` session consists of two TCP connections. The first connection is initiated by Host A (the client). An `rshd` process on Host B is listening to connection requests at port 514. Packets 1 to 3 are for the three-way handshake protocol. After the connection has been established, the client send `rsh` data (including user IDs and commands) to the Host B (Packet 4). The `rshd` process will authenticate the user, and if the user is authenticated, `rshd` initiates a separate TCP connection with the client.

The second connection is used for sending error messages. In the trace above, since there was no error, the connection was never used, but the connection must be successfully established, or `rshd` will not continue. Packets 6 to 7 are for the three-way handshake protocol of the second connection.

After the second connection has been established, Host B will send a zero byte to the client (using the first connection), Host A will acknowledge the packet. After that, `rshd` on Host B will run the command sent by the client, and the output of the command will be sent back to the client, all via the first connection. Students can use Wireshark to capture a `rsh` session, and study its behaviors, before launching the Mitnick attack. We divide the attack task into two sub-tasks, each one focusing on one connection.

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 3, 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.

Step 1: Spoof a SYN packet. Students should 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.

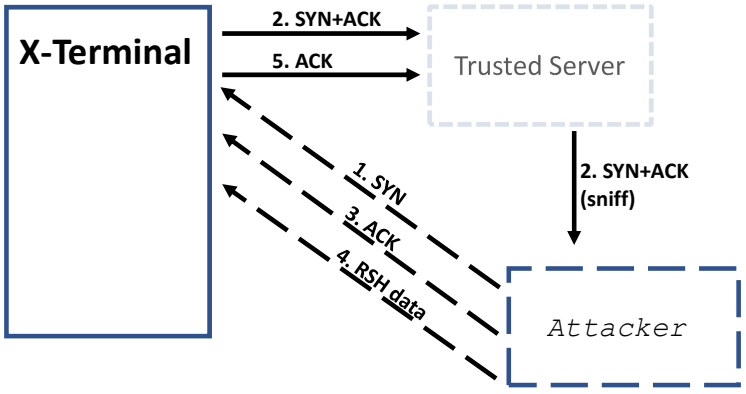

Figure 3: First Connection

```
# 'U': URG bit
# 'A': ACK bit
# 'P': PSH bit
# 'R': RST bit
# 'S': SYN bit
# 'F': FIN bit

tcp = TCP()

# Set the SYN and ACK bits
tcp.flags = "SA"

# Check whether the SYN and ACK are the only bits set
if tcp.flags == "SA":

# Check whether the SYN and ACK bits are set
if 'S' in tcp.flags and 'A' in tcp.flags:
```

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 (see Packet 2 in Listing 1).

Step 2: Respond to the SYN+ACK packet. After X-Terminal sends out a SYN+ACK, the trusted server needs to send out an ACK packet to complete the three-way handshake protocol. The acknowledge number in the packet should be `S+1`, where `S` is the sequence number contained in the SYN+ACK packet. See Packet 3 in Listing 1.

In the actual Mitnick attack, the attacker could not see the SYN+ACK packet, because it was sent to the trusted server, not to the attacker. That is why Mitnick had to guess the value of the sequence number. In this lab, we allow students to get the sequence number via packet sniffing.

Students need to write a sniff-and-spoof program using `Scapy` and run it on the attacker's machine. Here is a skeleton of a sniff-and-spoof program that might be useful. Please make sure to follow the restrictions described at the beginning of the section, or you will get a penalty.

```
#!/usr/bin/python3
```

```
from scapy.all import *

x_ip      = "10.9.0.5"  # X-Terminal
x_port    = 514         # Port number used by X-Terminal

srv_ip    = "10.9.0.6"  # The trusted server
srv_port  = 1023        # Port number used by the trusted server

# Add 1 to the sequence number used in the spoofed SYN
seq_num     = 0x1000 + 1


def spoof(pkt):
  global seq_num   # We will update this global variable in the function

  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={}".format(old_ip.src, old_tcp.sport,
                         old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))



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

  # Check whether it is a SYN+ACK packet or not;
  #   if it is, spoof an ACK packet

  # ... Add code here ...

myFilter = 'tcp'   # You need to make the filter more specific
sniff(iface='br-****', filter=myFilter, prn=spoof)
                  ↖ You need to set the correct value here.
```

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. The port number will be used for the second connection (see Task 2.2). Both client and server's user ID is `seed` in our container. The four fields are separated by a byte 0. Note that there is also a byte 0 at the end of the `rsh` data. An example is given in the following. 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)
```

Students should modify the sniff-and-spoof program written in Step 2, so an `rsh` data packet is sent to

X-Terminal (see Packet 4 in Listing 1). If this step is successful, from Wireshark, we can see that X-Terminal is going to initiate a TCP connection to the trusted server's port `9090`, which is the port number specified in our `rsh` data.

In your report, please describe whether the `touch` command has been executed on X-Terminal or not. Please also include snapshots of your Wireshark.

5.2 Task 2.2: Spoof the Second TCP Connection

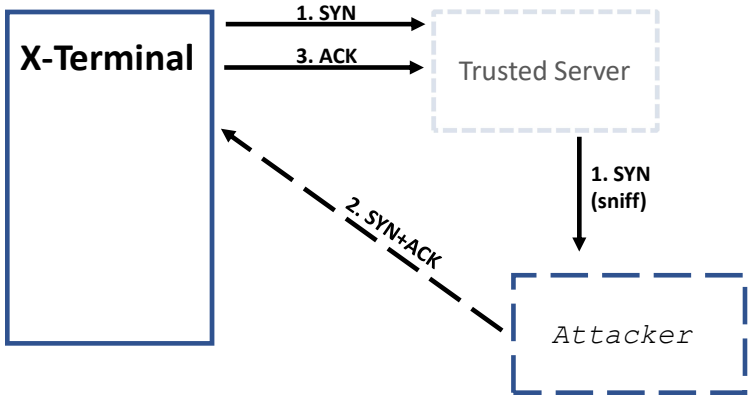

Figure 4: Second Connection

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. See Figure 4.

Students need to write another sniff-and-spoof program, which sniffs the TCP traffic going to the port 9090 of the trusted server (assuming `9090` is used in Task 2.1). When it sees a SYN packet, it should respond with a SYN+ACK packet. See Packet 7 in Listing 1 for an example.

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. Please include your evidence in your report.

6 Task 3: Set Up a Backdoor

In Task 2, we only run a `touch` command in the attack to prove that we can successfully run a command on X-Terminal. If we want to run more commands later, we can always launch the same attack. That is quite inconvenient.

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, as we have discussed in Section 3.5, all we need to do is to add the string `"+ +"` to the `.rhosts` file (in a single line). We can include the following command in our `rsh` data.

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

Students should replace the `touch` command in Task 2 with the `echo` command above, and then repeat the attack. 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]
```

The `rsh` program may have not been installed on the attacker container, but you can easily install it using the following commands:

```
# apt-get update && apt-get -y install rsh-redone-client
```

7 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.