Table of Contents:

1 LAB MANUAL FOR SNIFFING AND SPOOFING

1.1 Introduction

Packet sniffing and spoofing are the two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. There are many packets sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software. The objective of this lab is for students to master the technologies underlying most of the sniffing and spoofing tools. Students will play with some simple sniffer and spoofing programs, read their source code, modify them, and eventually gain an in-depth understanding on the technical aspects of these programs. At the end of this lab, students should be able to write their own sniffing and spoofing programs.

1.1.1 Sniffing (Snooping)

Listening to a conversation. For example, if you login to a website that uses no encryption, your username and password can be sniffed off the network by someone who can capture the network traffic between you and the web site.

1.1.2 Spoofing

Actively introducing network traffic pretending to be someone else. For example, spoofing is sending a command to computer A pretending to be computer B. It is typically used in a scenario where you generate network packets that say they originated by computer B while they really originated by computer C. Spoofing in an email context means sending an email pretending to be someone else.

2 Using Tools to Sniff and Spoof Packets using Scapy

Attacker Machine: 10.0.2.9

Victim Machine: 10.0.2.10

In this task set, a Python-based Scapy tool is used to sniff and spoof packets.

If you don’t have Scapy installed in your system, you can use the below mentioned command to install it.

Command:
```
sudo apt-get install Scapy
```

2.1 Task 1: Sniffing Packets

2.1.1 Task 1.1 Sniff IP packets using Scapy

The below code shows a sample code to sniff ip packets. The callback function in the sniff function displays the source, destination IP address and the protocol. The program acts as a sniffer and captures the ping requests sent by another machine on the same network.

In the callback function, we can write any code like get packet information or spoof packet or process packets etc.

sample.py

```
#!/usr/bin/python
from scapy.all import *
print("SNIFFING PACKETS...");
def print_pkt(pkt):
     pkt.show()
pkt = sniff(filter='icmp',prn=print_pkt)
```

Command:

```
sudo python sample.py
```

Explain on which VM you ran this command and why? Provide a screenshot of your observations.

Command:

```
python sample.py
```

Now, we run the same program without root privileges. Do you find any issues? If so, why?

Provide a screenshot of your observations.

2.1.2 Task 1.2 Capturing ICMP, TCP packet and Subnet

2.1.2.1 Capture only the ICMP packet

The ICMP packets captured by the Scapy sniffer program. The filter is only for the ICMP packets. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

sample.py

```
#! / usr/bin/python
from scapy.all import *
print ("SNIFFING PACKETS...");
def print_pkt(pkt):
     pkt. Show ()
pkt = sniff (filter='icmp', prn=print_pkt)
```

Command:

```
sudo python sample.py
```

Provide a screenshot of your observations

Open another terminal on the same VM ping 8.8.8.8

Command:

```
ping 8.8.8.8
```

The ICMP packets are captured by the sniffer program. Provide a screenshot of your observations.

2.1.2.2 Capture any TCP packet that comes from a particular IP and with a destination port number 23

The below code sniffs the TCP traffic from a specific host (10.0.2.10) to port 23.

sniff.py

```
#!/usr/bin/python
from scapy.all import *
print ("SNIFFING PACKETS...");
def print_pkt(pkt):
     pkt. show ()
pkt = sniff (filter='tcp and (src host 10.0.2.10 and dst
port 23)’, prn=print_pkt)
```

Command:

```
telnet 10.0.2.9
```

Explain where you will run Telnet. Provide screenshots of your observations.

2.1.2.2.1 iii) Capture packets comes from or to go to a particular subnet

You can pick any subnet, such as 192.168.254.0/24; you should not pick the subnet that your VM is attached to. The below code sniffs the TCP traffic from a subnet as the source to any destination.

sniff1.py

```
#!/usr/bin/python
from scapy.all import *
print("SNIFFING PACKETS...");
def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='src net 192.168.254.0/24',
prn=print_pkt)
```

Show that on sending ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 (the src subnet in filter).

Commands:

```
ping 192.168.254.1
```

Provide a screenshot of your observations.

```
sudo python sniff1.py
```

Provide a screenshot of your observations.

2.1.3 Task 2: Spoofing

VM1 Attacker Machine: 10.0.2.9

VM2 Victim Machine: 10.0.2.10

In this task set, Python-based Scapy tool is used to spoof packets.

Command: `sudo apt-get install Scapy`

The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof **ICMP echo request packets** and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. Below shows the code to create the ICMP packet. The spoofed request is formed by creating our own packet with the header specifications. Here we create an ICMP header. The default type of ICMP is 8. Hence, we don’t have to mention it while creating the packet. We can mention it as: ICMP (type=8).

Similarly, we fill the IP header with source IP address of any machine within the local network and destination IP address of any remote machine on the internet which is alive.

spoof.py

```
#! /usr/bin/python
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...");
IPLayer = IP()
IPLayer.src="10.0.2.10"
IPLayer.dst="192.168.254.1"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt. Show ()
send(pkt,verbose=0)
```

Command:

```
sudo python spoof.py
```

Provide a screenshot of your observations.

Show from Wireshark capture that the live machine sends back an ICMP response.

Command:

```
ping 10.0.2.9
```

Open Wireshark and observe the ICMP packets as they are being captured.

Provide screenshots of your observations.

2.1.4 Task 3: Traceroute

The objective of this task is to implement a simple traceroute tool using Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. We will send a packet (any type) to the destination, with its Time-To-Live (TTL) field set to 1 first. This packet will be dropped by the first router, which will send us an ICMP error message, telling us that the TTL has exceeded. Hence, we get the IP address of the first router. We then increase our TTL field to 2, send out another packet, and get the IP address of the second router. We will repeat this procedure until our packet finally reach the destination.

The below code is a simple traceroute implementation using Scapy. It takes hostname or IP address as the input. We create IP packet with destination address and TTL value and ICMP packet. We send the packet using function sr1(). This function waits for the reply from the destination. If the ICMP reply type is 0, we receive echo response from the destination, else we increase the TTL value and resend the packet.

traceroute.py

```
from scapy.all import *
'''Usage:  ./traceroute.py " hostname or ip address"'''
host=sys.argv[1]
Print ("Traceroute "+ host)
ttl=1
while 1:
    IPLayer=IP ()
    IPLayer.dst=host
    IPLayer.ttl=ttl
    ICMPpkt=ICMP()
    pkt=IPLayer/ICMPpkt
    replypkt = sr1(pkt,verbose=0)
    if replypkt is None:
        break
    elif replypkt [ICMP].type==0:
        print "%d hops away: "%ttl, replypkt [IP].src
        print "Done”, replypkt [IP].src
        break
    else:
        print "%d hops away: "%ttl, replypkt [IP].src
        ttl+=1
```

Command:

```
sudo python traceroute.py 192.168.254.1
```

On running the above python code, provide a screenshot of the response.

Provide a screenshot of the Wireshark capture that shows the ICMP requests sent with increasing TTL and the error response from the routers with a message as “Time to live exceeded”.

2.1.5 Task 4: Sniffing and-then Spoofing

In this task, victim machine pings a non-existing IP address “1.2.3.4”. As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

VM 1 (Victim): 10.0.2.10

Ping X: 1.2.3.4

VM 2 (attacker with sniffer-spoofer running): 10.0.2.9

The below code shows the sniffing and spoofing using Scapy. The below code sniffs ICMP packets sent out by the victim machine. Using the callback function, we can use the packets to send the spoofed packets. We retrieve source IP and destination IP from the sniffed packet and create a new IP packet. The new source IP of the spoofed packet is the sniffed packet’s destination IP address and vice versa. We also generate ICMP packet with id and sequence number. In the new packet, ICMP type should 0 (ICMP reply). To avoid truncated packet, we also add the data to the new packet.

sniffspoof.py

```
#!/usr/bin/python
from scapy.all import *
def spoof_pkt(pkt):
     newseq=0
     if ICMP in pkt:
          print("original packet.........")
          Print ("source IP :", pkt [IP].src)
          Print ("Destination IP :", pkt [IP]. dst)
          srcip = pkt [IP]. dst
          dstip = pkt[IP].src
          newihl = pkt [IP]. ihl
          newtype = 0
          newid = pkt [ICMP].id
          newseq = pkt [ICMP]. seq
          data = pkt [Raw]. load
          IPLayer = IP (src=srcip, dst=dstip, ihl=newihl)
          ICMPpkt = ICMP (type=newtype, id=newid, seq=newseq)
          newpkt = IPLayer/ICMPpkt/data
          print ("spoofed packet........")
          print ("Source IP:", newpkt [IP].src)
          print ("Destination IP:", newpkt [IP]. dst)
          send (newpkt, verbose=0)
pkt = sniff (filter='icmp and src host 10.0.2.10', prn=spoof_pkt)
```

On running above code, we can see that our spoofer program sends spoofed ICMP responses to the ICMP requests set by the victim machine. The victim machine pings a non-existing IP address, but gets back ICMP response.

Command:

```
ping 1.2.3.4
```

Provide a screenshot of your observations.

open another terminal in the same VM and execute the following command

Command:

```
sudo python sniffspoof.py
```

Provide a screenshot of your observations.