**ACS 545 Cryptography and Network Security**

**Lab 2: Packet Sniffing and Spoofing Lab**

**Name:** Vijay Anand Pandian

**PUID:** 900383786

**Repo Link:**

**Task1.1 – Using Scapy to Sniff and Spoof Packets**

**Graphical user interface, text, application

Description automatically generated**

**Task1.1A –**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

print("Ready to sniff packets...")

def print\_pkt(pkt):

pkt.show()

pkt = sniff(iface="br-c83e1ac99060", prn=print\_pkt)

**Output with root privilege:**

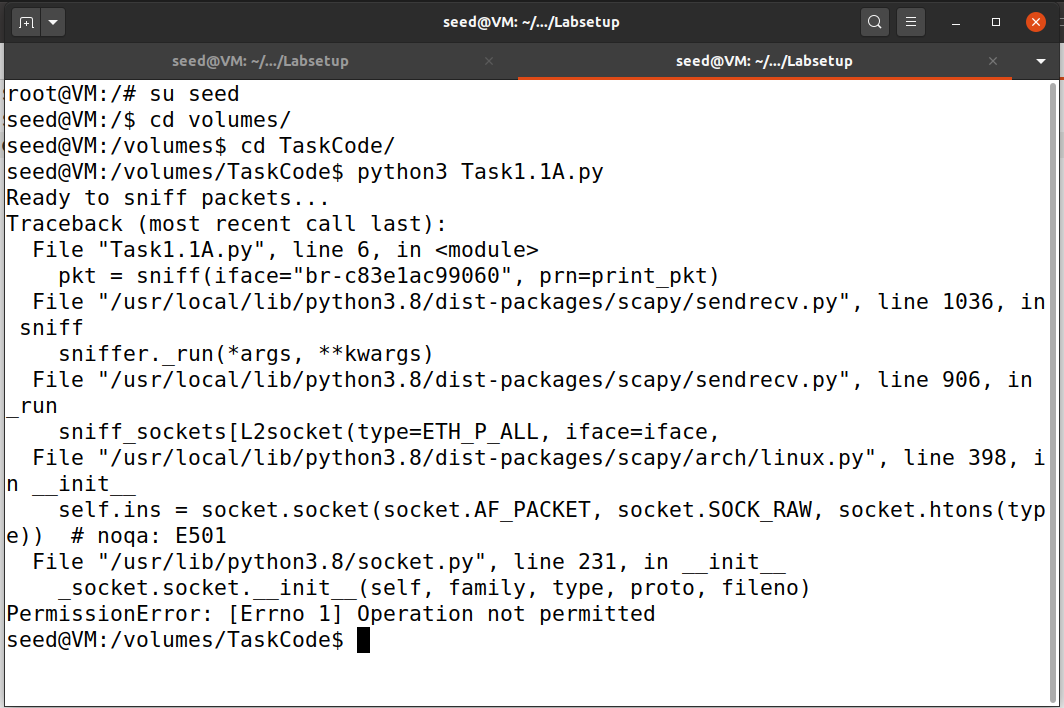
**Graphical user interface, text, application, email

Description automatically generated**

**Explanation:**

When the above code is executed and when we try to ping an IP from host A(10.9.0.5) the attacker will be able to sniff the packets.

**Output without root privilege:**



**Explanation:**

Scapy requires root privilege to sniff packets because it will not be able to access the network interface of the machine to capture and analyze traffic. Because accessing the network interface is allowed only to root users as a security measure to prevent unauthorized access. Here without root privilege scapy is unable to access the raw packet and sniff it.

**Task1.1B – Sniffing Packets By Setting Filters**

**Capture only ICMP Packet**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

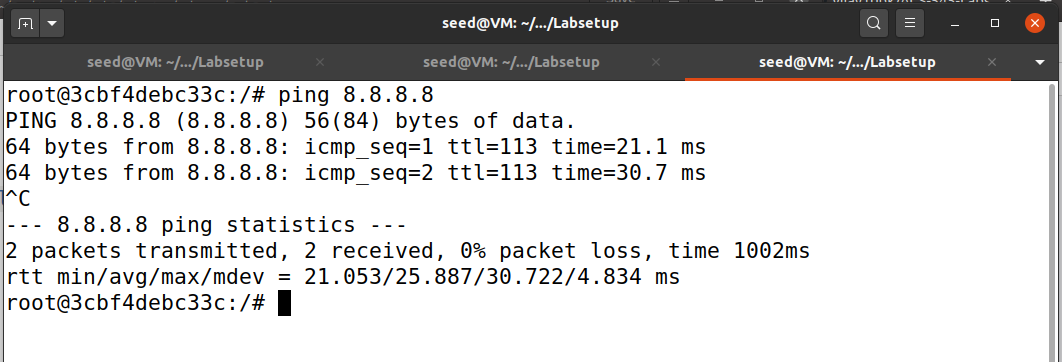
print("Ready to sniff packets...")

def print\_pkt(pkt):

pkt.show()

pkt = sniff(iface="br-c83e1ac99060", filter="icmp", prn=print\_pkt) #This will sniff only the icmp packets

**Output:**

****

**Text

Description automatically generated**

**Explanation:**

In the above code, I have set the filter as ‘ICMP’ so that it will filter only ICMP packets when HostA(10.9.0.5) tries to ping 8.8.8.8 which uses the BPF(Berkeley Packet Filter) syntax.

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

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

print("Ready to sniff packets...")

def print\_pkt(pkt):

pkt.show()

pkt = sniff(iface="br-c83e1ac99060", filter="tcp and src host 10.9.0.5 and dst port 23", prn=print\_pkt) #This will sniff only the tcp packets from the mentioned IP and with destination port number of 23

**Output:**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Explanation:**

In the code, I have set the filter value to filter only tcp packets from the IP 10.9.0.5 with the destination port number 23. telnet command will establish a telnet session in hostA(10.9.0.5) to capture and analyze telnet traffic.

**Capture packets comes from or to go to a particular subnet.**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

print("Ready to sniff packets...")

def print\_pkt(pkt):

pkt.show()

pkt = sniff(iface="br-c83e1ac99060", filter="src net 172.17.0.0/24", prn=print\_pkt) #This will sniff packets only the from above mention subnet

**Output:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Explanation:**

In the above code, the filter value is set in a way to filter packets only from or to a particular subnet(172.17.0.1/24).

**Task1.2 - Spoofing ICMP Packets**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

print("Sending spoofed packets...")

a = IP()

a.src = '10.9.0.9' #arbitary attacket IP

a.dst = '10.9.0.5' #HostA Ip

b = ICMP()

p = a/b # the division operator signifies that ICMP as payload of IP

p.show() #displays the data

send(p, verbose=0)

**Output: Text

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Explanation:**

In the above code, I have written code to send spoofed ICMP packets from one VM container 10.9.0.9(arbitrary attacker IP) to another container 10.9.0.5(host A IP).

**Task1.3: Traceroute**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

a = IP()

a.dst=sys.argv[1] #input hostname or ip address

a.ttl = 1 #distance

while True:

b = ICMP()

p = a/b

rp = sr1(p, timeout=2, verbose=0) #wait for response from dst

if rp is None:

print("No Response")

break

elif rp [ICMP].type==0: #if it gets the response it will print the number of hops and IP addresses of router it hops through.

print(f'{a.ttl} hops away:', rp [IP].src)

print("Done", rp [IP].src)

break

else:

print(f'{a.ttl} hops away: ', rp [IP].src)

a.ttl+=1 #ttl value is increased and packet is resent

**Output:Text

Description automatically generated**

**Explanation:**

In the above code, I have written a program to trace the route and number of hops required to reach the destination based on ttl. If attacker didn’t get any response from the destination it will print “No Response”. If the attacker gets a response from the destination it will print the route and IP addresses router it hops through. Else it will increase the ttl and resend the packets.

**Task1.4: Sniffing and then Spoofing**

**Code:**

#!/usr/bin/env python3

from scapy.all import \*

def spoof\_pkt(pkt):

if ICMP in pkt:

print("Original Packet...")

print("Source IP: ", pkt[IP].src)

print("Destination IP: ", pkt[IP].dst)

ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl, ttl=50)

icmp = ICMP(type=0, id= pkt[ICMP].id, seq= pkt[ICMP].seq)

data = pkt[Raw].load

newpkt = ip/icmp/data

print("Spoofed Packet....")

print("Source IP: ", newpkt[IP].src)

print("Destination IP: ", newpkt[IP].dst)

send(newpkt, verbose=0)

pkt = sniff(iface=['br-c83e1ac99060'], filter='icmp', prn=spoof\_pkt)

**Output:**

**Graphical user interface, text, application

Description automatically generated**

**Text

Description automatically generated**

**Explanation:**

In the above code, the 'if' block determines whether it is an ICMP request.

If it is true, the reply packet will be based on the original packet's details, but it will flip dst and src so that whenever it sees an ICMP echo request, regardless of the target IP address, the program should immediately respond. Using this packet spoofing technique, send out an echo response. The pk[Raw].load is used to store the original packet data payload so that it can be returned to the sender properly.

**Scenario 1:**

**Ping 1.2.3.4**

In this scenario, When I tried to ping 1.2.3.4 from a host VM which is a non existing IP but still the attacker will spoof packets and sniff packets from the host VM.

**Scenario 2 – ping 10.9.0.99 :**

In this scenario, When I tried to ping 10.9.0.99 from a host VM which is also a non existing IP. So there is not response to the ARP request so it will show error message as “Destination Host Unreachable”

**Scenario 3 – ping 8.8.4.4:**

In this scenario, When I tried to ping 8.8.4.4 from a host VM I an existing IP on the internet. So in this case we’re getting Duplicate responses which are mentioned as “DUP!”, that is because the real destination and attacker are also sending a response to the source VM making the request.

Graphical user interface, text, application

Description automatically generated

In the above screenshot, we can see that there is no route for 10.9.0.99 except for the other two IPs. ARP try to figure out the route and it fails.