

Lab 1: Packet Sniffing and Spoofing

Name: Guo Yuchen

Student ID: 1004885

Setup and preparation

Starting the containers using `dcup`

```
seed@VM: ~/.../Labsetup
[09/24/23]seed@VM:~/.../Labsetup$ dcup
hostB-10.9.0.6 is up-to-date
seed-attacker is up-to-date
hostA-10.9.0.5 is up-to-date
Attaching to hostB-10.9.0.6, seed-attacker, hostA-10.9.0.5
hostA-10.9.0.5 | * Starting internet superserver inetd      [ OK ]
hostB-10.9.0.6 | * Starting internet superserver inetd      [ OK ]

```

Using `ifconfig` to get network interface ID `br-e6512860ed72`

```
seed@VM: ~/.../Labsetup
[09/24/23]seed@VM:~/.../Labsetup$ ifconfig
br-e6512860ed72: 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::42:39ff:fe84:f524 prefixlen 64 scopeid 0x20<link>
    ether 02:42:39:84:f5:24 txqueuelen 0 (Ethernet)
    RX packets 124 bytes 9632 (9.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 114 bytes 13821 (13.8 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 02:42:92:b9:bf:40 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::8ae2:d31:c806:f29 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:17:62:32 txqueuelen 1000 (Ethernet)
    RX packets 672530 bytes 975379738 (975.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0

```

Task 1.1A

Create a Python file `sniffer.py` and make it executable using `chmod a+x sniffer.py`.

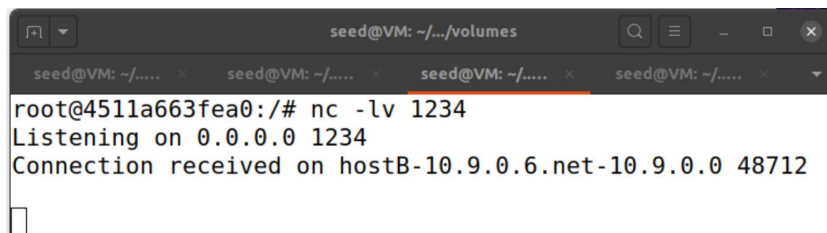
```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72", prn=print_pkt)
```

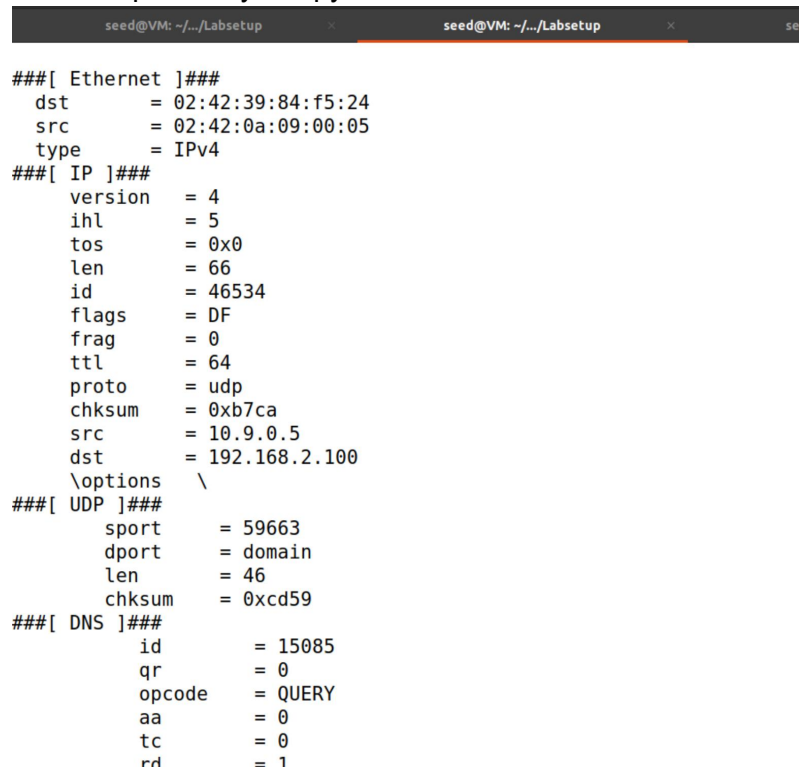
Running sniffer.py with root privilege

To create some traffic, use `nc` command on host 10.9.0.5 to listen to the connection from host 10.9.0.6

A terminal window titled 'seed@VM: ~/.../volumes' showing a netcat listener. The user is root@4511a663fea0. The command 'nc -lv 1234' is entered. The output shows 'Listening on 0.0.0.0 1234' and 'Connection received on hostB-10.9.0.6.net-10.9.0.0 48712'.

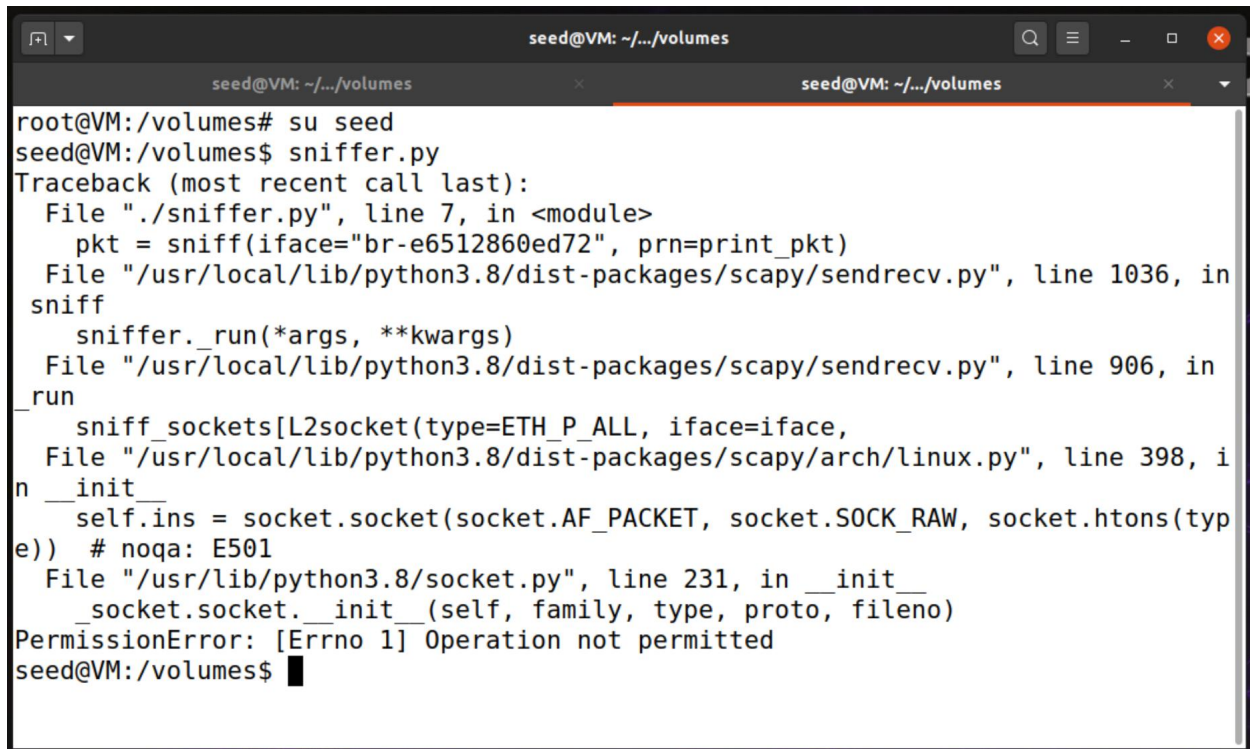
```
seed@VM: ~/.../volumes
root@4511a663fea0:/# nc -lv 1234
Listening on 0.0.0.0 1234
Connection received on hostB-10.9.0.6.net-10.9.0.0 48712
```

Traffic captured by Scapy:

A terminal window titled 'seed@VM: ~/.../Labsetup' showing the output of a Scapy sniff operation. The output is structured with '###' markers for Ethernet, IP, UDP, and DNS layers, listing various fields like dst, src, type, version, ihl, tos, len, id, flags, frag, ttl, proto, checksum, sport, dport, len, checksum, id, qr, opcode, aa, tc, and rd.

```
seed@VM: ~/.../Labsetup
###[ Ethernet ]###
  dst      = 02:42:39:84:f5:24
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 66
  id       = 46534
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = udp
  checksum = 0xb7ca
  src      = 10.9.0.5
  dst      = 192.168.2.100
  \options \
###[ UDP ]###
  sport    = 59663
  dport    = domain
  len      = 46
  checksum = 0xcd59
###[ DNS ]###
  id       = 15085
  qr       = 0
  opcode   = QUERY
  aa       = 0
  tc       = 0
  rd       = 1
```

Running sniffer.py without root privilege



```
seed@VM: ~/.../volumes
root@VM:/volumes# su seed
seed@VM:/volumes$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt = sniff(iface="br-e6512860ed72", 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_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    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
seed@VM:/volumes$
```

Observation

1. Only root can sniff packages.
2. Reasons: only root is allowed to open raw sockets, which is used by sniff to receive packets such that the packet won't be intercepted by the normal socket.

Task 1.1B

Capture icmp packet

Modify the filter parameter to be "icmp" to capture only the ICMP packet. Ping from 10.9.0.5 to 10.9.0.6 since ping sends icmp package.

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72", filter="icmp", prn=print_pkt)
```

```
seed@VM: ~/../volumes
[09/24/23]seed@VM:~/../volumes$ docksh 45
root@4511a663fea0:/# ping 10.9.0.6
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=0.287 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.191 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.280 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.362 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.098 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.216 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.092 ms
```

The packets are captured by the sniffer as shown:

```
seed@VM: ~/../volumes
root@VM: /volumes# sniffer.py
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 48906
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x6782
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x9897
  id       = 0x3f
  seq      = 0x1
###[ Raw ]###
  load     = '\x94\x91\x12e\x00\x00\x00\x00\xf6^\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
```

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

Change the filter parameter accordingly.

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72",
            filter="tcp and src net 10.9.0.5 and dst port 23",
            prn=print_pkt)
```

Then try several different command:

- ping 10.9.0.6: wrong type, wrong dst port
- ping 10.9.0.6/23: wrong type
- nc -vz amazon.com 80: wrong dst port
- nc -v google.com 23

```
seed@VM: ~/.../Labsetup  x      seed@VM: ~/.../Labsetup  x      seed@VM: ~/.../volumes
[09/24/23]seed@VM:~/.../volumes$ docksh 45
root@4511a663fea0:/# ping 10.9.0.6
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=0.080 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.061 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.140 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.057/0.084/0.140/0.033 ms
root@4511a663fea0:/# ping 10.9.0.6 23
PING 23 (0.0.0.23) 56(124) bytes of data.
^C
--- 23 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6135ms

root@4511a663fea0:/# nc -vz amazon.com 80
Connection to amazon.com 80 port [tcp/http] succeeded!
root@4511a663fea0:/# printf "GET / HTTP/1.0\r\n\r\n" | nc -v google.com 23
^C
root@4511a663fea0:/# █
```


Sniffer captures:

```
###[ Ethernet ]###
  dst      = 02:42:39:84:f5:24
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 60
  id       = 21530
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  checksum = 0x79b2
  src      = 10.9.0.5
  dst      = 74.125.24.101
  \options \
###[ TCP ]###
  sport     = 55966
  dport     = telnet
  seq       = 1706082090
  ack       = 0
  dataofs   = 10
  reserved  = 0
  flags     = S
  window    = 64240
  checksum  = 0x6d1e
```

It only responds for the last case, when the port number, src ip, and packet type fit the filter.

Capture packets comes from or to go to a particular subnet

Choose the subnet to be 128. 230.0.0/16 and change to filter parameter accordingly.

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface="br-e6512860ed72",
            filter="net 128.230.0.0/16",
            prn=print_pkt)
```

Let one of the host sends a tcp packet to this ip address:

```
seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x
root@b8f397ec8e24:/# nc -zv 128.230.0.0 16
^C
root@b8f397ec8e24:/# █
```

Then the sniffer program captures the packet which has dst and port as specified:

```
###[ Ethernet ]###
  dst      = 02:42:39:84:f5:24
  src      = 02:42:0a:09:00:06
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 60
  id       = 41093
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0xf42
  src      = 10.9.0.6
  dst      = 128.230.0.0
  \options \
###[ TCP ]###
  sport    = 56732
  dport    = 16
  seq      = 2464214675
  ack      = 0
  dataofs  = 10
  reserved = 0
  flags    = S
  window   = 64240
  chksum   = 0x8b23
  urgptr   = 0
```

Task 1.2

icmpSpoof.py:

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.src = "6.6.6.6"
a.dst = "10.9.0.6"
b = ICMP()
p = a/b
send(p)
```

The screenshot shows a terminal window on the right and a Wireshark packet capture window on the left. The terminal window displays the command `root@VM:/volumes# icmpSpoof.py` and the output `Sent 1 packets.` and `root@VM:/volumes#`. The Wireshark window shows a packet capture with four packets. The first two packets are ARP requests from the source IP to the destination IP. The third and fourth packets are ICMP echo (ping) requests and replies, respectively, from the source IP to the destination IP.

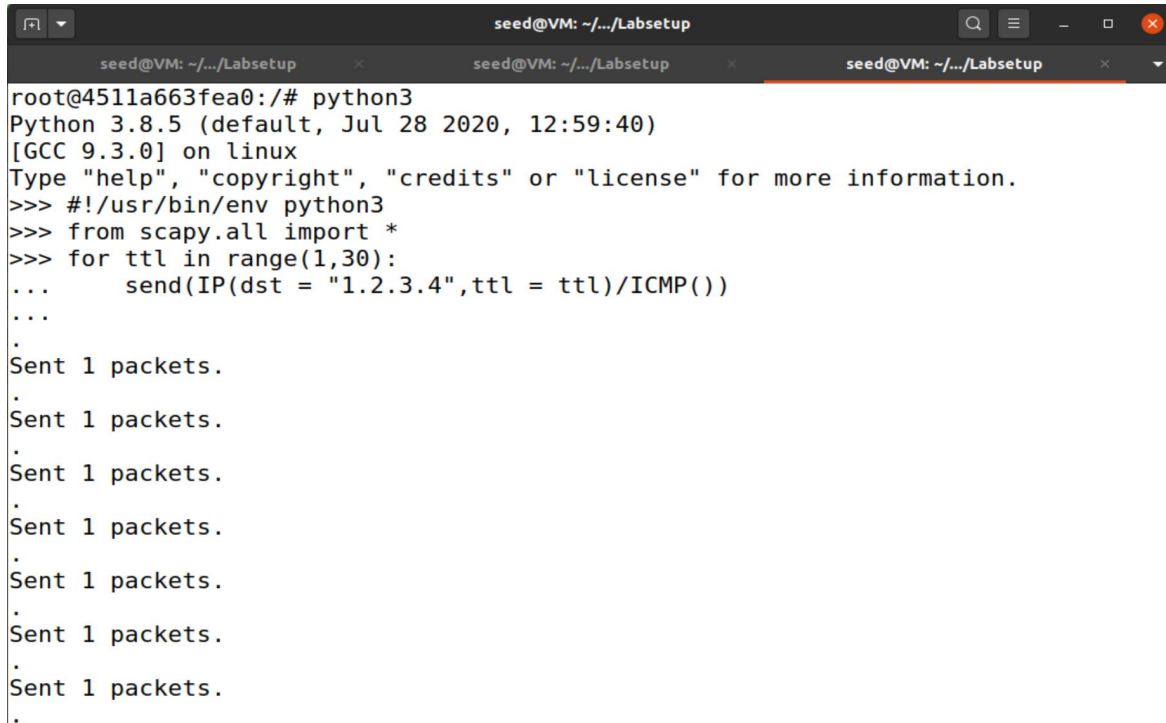
No.	Time	Source	Destination	Protocol	Length	Info
1	2023-09-24 04:4...	02:42:fa:06:4e:7a	Broadcast	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
2	2023-09-24 04:4...	02:42:0a:09:00:06	02:42:fa:06:4e:7a	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
3	2023-09-24 04:4...	6.6.6.6	10.9.0.6	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, t
4	2023-09-24 04:4...	10.9.0.6	6.6.6.6	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, t

In the `icmpSpoof.py`, the program sends a spoofed icmp echo request with source ip 6.6.6.6 to the host 10.9.0.6. And as observed in the Wireshark (the last line), the host 10.9.0.6 receives an icmp echo reply from the spoofed source 6.6.6.6, which is specified in our spoofer program.

Task 1.3

To estimate the distance between the VM and a selected destination in terms of number of routers, let one of the hosts to run the program as shown in the terminal.

Basically, we are creating a sequence of icmp packets with a destination ip address to be 1.2.3.4, and repeatedly increased ttl from 1 to 30.



```
seed@VM: ~/.../Labsetup
root@4511a663fea0:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> #!/usr/bin/env python3
>>> from scapy.all import *
>>> for ttl in range(1,30):
...     send(IP(dst = "1.2.3.4",ttl = ttl)/ICMP())
...
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
```

Then as observed and recorded in the Wireshark, the routes are shown in the black lines source column, which I summarized as follows:

1. 10.9.0.1
2. 10.0.2.2
3. 10.12.0.1
4. 172.16.1.13
5. 172.16.1.106
6. 172.16.1.210
7. 192.168.22.27
8. 103.24.77.1
9. 61.16.87.153
10. 203.118.6.233

And since 1.2.3.4 does not exist, the host could not get an echo reply from there.

[SEED Labs] *br-e6512860ed72

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmpl

No.	Time	Source	Destination	Protocol	Length	Info
3	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f...
4	2023-09-24 03:11	10.9.0.1	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
5	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f...
6	2023-09-24 03:11	10.0.2.2	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f...
8	2023-09-24 03:11	10.12.0.1	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f...
10	2023-09-24 03:11	172.16.1.13	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f...
12	2023-09-24 03:11	172.16.1.100	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
13	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f...
14	2023-09-24 03:11	172.16.1.210	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
15	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f...
16	2023-09-24 03:11	192.168.22.27	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f...
18	2023-09-24 03:11	109.21.77.1	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response f...
20	2023-09-24 03:11	61.10.87.153	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response ...
22	2023-09-24 03:11	203.118.6.233	10.9.0.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response ...
24	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response ...
25	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response ...
26	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response ...
27	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response ...
28	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response ...
29	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=17 (no response ...
30	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=18 (no response ...
31	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=19 (no response ...
32	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (no response ...
33	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=21 (no response ...
34	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=22 (no response ...
35	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=23 (no response ...
36	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=24 (no response ...
37	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=25 (no response ...
38	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=26 (no response ...
39	2023-09-24 03:11	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=27 (no response ...

Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-e6512860ed72, id 0

wireshark_br-e6512860ed72_20230924031313_OFregs.pcapng

Packets: 45 - Displayed: 39 (86.7%)

Profile: Default

Task 1.4

The sniff and spoof program ss.py is as follows:

```
#!/usr/bin/env python3
from scapy.all import *

# def print_pkt(pkt):
#     pkt.show()

def spoof_pkt(pkt):
    src = pkt[IP].dst
    dst = pkt[IP].src
    a = IP(src=src, dst=dst, ttl=40)
    b = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq, code=0)
    raw = pkt[Raw]
    send(a/b/raw)

pkt = sniff(iface="br-e6512860ed72",
            filter="icmp and host 10.9.0.5",
            prn=spoof_pkt)
```

After a packet is sniffed, the callback function `spoof_pkt` is called to spoof the packet. The `spoof_pkt` will reverse the destination and source ip address, take the icmp id and seq number, then construct an icmp echo reply message to send it back.

ping 1.2.3.4

a non-existing host on the Internet

```
seed@VM: ~/.../volumes
root@4511a663fea0:/# ping -c 3 1.2.3.4
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=40 time=46.8 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=40 time=19.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=40 time=25.1 ms

--- 1.2.3.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 19.146/30.369/46.820/11.886 ms
root@4511a663fea0:/#
```

Observation: the ping program receives 3 echo replies, which are spoofed by the spoofer program, although 1.2.3.4 does not exist.

The image shows a Wireshark packet capture window titled "[SEED Labs] *br-e6512860ed72". The packet list shows 10 packets. Packets 1, 2, and 3 are ARP requests from 10.9.0.5 to 1.2.3.4. Packets 4, 5, 6, 7, 8, and 9 are ICMP Echo (ping) requests from 10.9.0.5 to 1.2.3.4. Packet 10 is an ARP request from 10.9.0.1 to 1.2.3.4. The packet details pane shows the selected packet (packet 10) as an Internet Control Message Protocol (ICMP) Echo (ping) request. The data field shows the raw data: 54d69a00000000001112131415161718191a1b1c1d1e1f...

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-09-27 02:11	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x005a, seq=1/256, ttl=64 (reply in 4)
2	2023-09-27 02:11	02:42:fa:06:4e:7a	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
3	2023-09-27 02:11	02:42:0a:09:00:05	02:42:fa:06:4e:7a	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
4	2023-09-27 02:11	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005a, seq=1/256, ttl=40 (request in 4)
5	2023-09-27 02:11	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x005a, seq=2/512, ttl=64 (reply in 6)
6	2023-09-27 02:11	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005a, seq=2/512, ttl=40 (request in 6)
7	2023-09-27 02:11	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x005a, seq=3/768, ttl=64 (reply in 8)
8	2023-09-27 02:11	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005a, seq=3/768, ttl=40 (request in 8)
9	2023-09-27 02:11	02:42:0a:09:00:05	02:42:fa:06:4e:7a	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
10	2023-09-27 02:11	02:42:fa:06:4e:7a	02:42:0a:09:00:05	ARP	42	10.9.0.1 is at 02:42:fa:06:4e:7a

Frame 10: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-e6512860ed72, id 0
Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:fa:06:4e:7a (02:42:fa:06:4e:7a)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 1.2.3.4
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0xface [correct]
[Checksum Status: Good]
Identifier (BE): 90 (0x005a)
Identifier (LE): 23040 (0x5a00)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
[Response frame: 4]
Timestamp from icmp data: Sep 27, 2023 02:12:27.000000000 EDT
[Timestamp from icmp data (relative): 0.710267679 seconds]
Data (48 bytes)
Data: 54d69a00000000001112131415161718191a1b1c1d1e1f...

ping 10.9.0.99

a non-existing host on the LAN

```
seed@VM: ~/.../volumes
root@4511a663fea0:/# ping -c 3 10.9.0.99
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
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable

--- 10.9.0.99 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2055ms
pipe 3
root@4511a663fea0:/#
```

The image shows a Wireshark packet capture window titled "[SEED Labs] *br-e6512860ed72". The packet list shows three packets, all of which are ARP requests (Protocol: ARP, Length: 42) sent from source IP 10.9.0.5 to destination IP 10.9.0.99. The first packet is a request, and the second and third are also requests, indicating no response was received. The packet details pane shows the first packet's structure: Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: Broadcast (ff:ff:ff:ff:ff:ff), and Address Resolution Protocol (request). The status bar at the bottom indicates "Packets: 3 · Displayed: 3 (100.0%) · Dropped: 0 (0.0%)".

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-09-27 02:1...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
2	2023-09-27 02:1...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
3	2023-09-27 02:1...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5

Observation: no packets is received, means that the spoofer could not spoof any message. Because the 10.9.0.99 is on the same LAN with the host 10.9.0.5, thus before they can exchange message, it needs the MAC address of 10.9.0.99. Since it's not stored in the ARP cache, it sends out a broadcast message to ask for reply, and the real communication only takes place after it acquires the destination device's MAC address. In our case, there is neither ARP reply or spoofed ARP reply, our spoofed icmp packet will not be received.

ping 8.8.8.8

an existing host on the Internet

```
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes

root@4511a663fea0:/# ping -c 3 8.8.8.8
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=113 time=4.83 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=40 time=50.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=8.13 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=40 time=25.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=6.99 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +2 duplicates, 0% packet loss,
time 2005ms
rtt min/avg/max/mdev = 4.826/19.196/50.136/17.207 ms
root@4511a663fea0:/#
```

The image shows a Wireshark packet capture of a ping operation. The packet list pane displays 13 packets. Packets 5, 7, 8, 9, 10, and 11 are ICMP Echo (ping) replies from 8.8.8.8 to 10.9.0.5. Packet 5 is the first real reply (ttl=113). Packet 7 is a spoofed reply (ttl=40) marked as a duplicate. Packets 8, 9, 10, and 11 are subsequent real replies (ttl=111, 113, 113, 113 respectively). The packet details pane for packet 5 shows the ICMP Echo (ping) reply with a sequence number of 1 and a TTL of 113. The packet bytes pane shows the raw data of the ICMP echo reply.

No.	Time	Source	Destination	Protocol	Length	Info
4	2023-09-27 02:2...	02:42:0a:09:00:05	02:42:fa:06:4e:7a	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
12	2023-09-27 02:2...	02:42:0a:09:00:05	02:42:fa:06:4e:7a	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
3	2023-09-27 02:2...	02:42:fa:06:4e:7a	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
13	2023-09-27 02:2...	02:42:fa:06:4e:7a	02:42:0a:09:00:05	ARP	42	10.9.0.1 is at 02:42:fa:06:4e:7a
1	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) request id=0x005d, seq=1/256, ttl=64 (reply in 2)
6	2023-09-27 02:2...	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x005d, seq=2/512, ttl=64 (reply in 7)
9	2023-09-27 02:2...	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x005d, seq=3/768, ttl=64 (reply in 1...)
2	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=1/256, ttl=113 (request i...)
5	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=1/256, ttl=40
7	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=2/512, ttl=111 (request i...)
8	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=2/512, ttl=40
10	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=3/768, ttl=113 (request i...)
11	2023-09-27 02:2...	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x005d, seq=3/768, ttl=40

Observation: both ping requests (the real reply and the spoofed reply) are received by the ping program. But the ones that are spoofed by me(ttl is set to 40 to distinguish) are marked as duplicated since they come later than the real one.