

CS 255 Computer Security

LAB 3: Packet Sniffing and Spoofing

Devasheesh Vaid
SID: 862395097
dvaid003@ucr.edu

Environment Setup using Containers:

\$ dcbuild

All images are already being used in the containers; hence this step is skipped, as shown below.

After that we run *dcup* command, which creates two hosts and one seed attacker and starts those docker containers as shown in the screenshot below.

\$ dcup

```
[11/23/23]seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/23/23]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadf8e8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
Creating hostB-10.9.0.6 ... done
Creating hostA-10.9.0.5 ... done
Creating seed-attacker ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 | * Starting internet supserver inetd          [ OK ]
hostB-10.9.0.6 | * Starting internet supserver inetd          [ OK ]
█
```

After starting the containers and running the below command we see the list of active running containers.

\$ dockps

```
[11/23/23]seed@VM:~$ dockps
b0607c98989d  seed-attacker
95c5fa082414  hostB-10.9.0.6
f58d70b67ac2  hostA-10.9.0.5
[11/23/23]seed@VM:~$ █
```

The IP address assigned to our VM is 10.9.0.1. We need to find the name of the corresponding network interface on our VM, because we need to use it in our programs. The interface name is the concatenation of *br-* and the ID of the network created by Docker. We use *ifconfig* to list network interfaces as below.

\$ ifconfig

```
[11/23/23]seed@VM:~$ ifconfig
br-8a184f62243c: 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:f5ff:fe52:373 prefixlen 64 scopeid 0x20<link>
    ether 02:42:f5:52:03:73 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 41 bytes 4987 (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:ca:97:32:7e 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
```

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

Task 1.1: Sniffing Packets

First, I create a *sniffer.py* script with the below code. Then I update the interface with the interface id which we got with *ifconfig* command previously.

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-8a184f62243c', filter='icmp', prn=print_pkt)
```

Task 1.1 A

On running the above *sniffer.py* with root privileges, we are sniffing for ICMP packets. Since *ping* command sends ICMP *echo-request* and receives ICMP *echo-reply* packets, I ping one of the hosts, Host A in our case, at 10.9.0.5 to look for ICMP packets.

On pinging the host A, I see ICMP packets in the terminal window where we are sniffing for packets. Below is the *echo-request* packet from source (*src*) 10.9.0.1 to destination (*dst*) 10.9.0.5.

The image shows two terminal windows side-by-side. The left window, titled 'root@VM: /home/seed/Downloads/Labsetup', shows the execution of a script named 'sniffer.py'. The script uses Scapy to sniff for ICMP packets on the interface 'br-8a184f62243c'. The output shows an ICMP echo-request packet from source 10.9.0.1 to destination 10.9.0.5. The packet details include version 4, ihl 5, tos 0x0, len 84, id 22994, flags DF, frag 0, ttl 64, proto icmp, checksum 0xc2bf, and sequence number 0x1. The right window, titled 'seed@VM: ~', shows the execution of the 'ping 10.9.0.5' command. The output shows four successful ping requests from 10.9.0.1 to 10.9.0.5, each receiving 64 bytes of data with varying times.

```
root@VM: /home/seed/Downloads/Labsetup
[11/25/23]seed@VM:~/Downloads/Labsetup$ sudo su
root@VM: /home/seed/Downloads/Labsetup# ./sniffer.py
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:44:d9:2a:c0
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 22994
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xc2bf
  src      = 10.9.0.1
  dst      = 10.9.0.5
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x2bca
  id       = 0x1
  seq      = 0x1
###[ Raw ]###
  load     = '\xc8\xa8be\x00\x00\x00\x00\xe1R\x01\x00\x00\x00\x00\x00\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
```

```
seed@VM: ~
[11/25/23]seed@VM:~$ 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=3.33 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.146 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.721 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.135 ms
^C
[1]+  Stopped                  ping 10.9.0.5
[11/25/23]seed@VM:~$
```

The *echo-reply* packet which is the response packet from 10.9.0.5 to 10.9.0.1 as shown below.

```
root@VM: /home/seed/Downloads/Labsetup
00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'

###[ Ethernet ]###
dst      = 02:42:44:d9:2a:c0
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 55245
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x8ec4
src      = 10.9.0.5
dst      = 10.9.0.1
options  =
###[ ICMP ]###
type     = echo-reply
code     = 0
chksum   = 0x33ca
id       = 0x1
seq      = 0x1
###[ Raw ]###
load     = '\xc8\xa8be\x00\x00\x00\xe1R\x01\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
```

```
seed@VM: ~
[11/25/23]seed@VM:~$ 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=3.33 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.146 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.721 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.135 ms
^Z
[1]+  Stopped                  ping 10.9.0.5
[11/25/23]seed@VM:~$
```

When I run the same *sniffer.py* script without the root privileges, we get the following *PermissionError*. Thus, root privileges are required for packet sniffing.

```
[11/25/23]seed@VM:~/../Labsetup$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt = sniff(iface='br-8a184f62243c', 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
[11/25/23]seed@VM:~/../Labsetup$
```

Task 1.1B.

- **Capture only the ICMP packet:**

This is done in the previous task- Task 1.1A

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

For this I update the *sniffer.py* file and modify the *filter* attribute of the *sniff()* as below. Therefore, it will sniff for any packet using TCP and coming from host 10.9.0.5 (one of the host container IP) with a destination port 23.

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-8a184f62243c', filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt)
```

Since the destination port mentioned is 23, it is used by *TELNET*, for remote access. So, to do that, I log in to the host A docker container, with IP 10.9.0.5, and do a *telnet 10.9.0.1* to our VM as shown below.

```
root@VM: /home/seed/Downloads/Labsetup# ./sniffer.py
###[ Ethernet ]###
dst      = 02:42:f5:52:03:73
src      = 02:42:8a:09:00:05
type     = IPv4

###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x10
len      = 60
id       = 9548
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
checksum = 0x149
src      = 10.9.0.5
dst      = 10.9.0.1

\options
###[ TCP ]###
sport    = 54404
dport    = telnet
seq      = 1357713544
ack      = 0
dataoffs = 10
reserved = 0
flags    = S
window   = 64240
chksum   = 0x1446
urgptr   = 0
options  = [('MSS', 1460), ('SAckOK', 'b'), ('Timestamp', (2517899384, 0)), ('NOP', None), ('WScale', 7)]

###[ Ethernet ]###
dst      = 02:42:f5:52:03:73
```

- **Capture packets come from or to go to a particular subnet.**

Again I start with modifying the *filter* attribute of the *sniff()* as below. I chose the subnet as *128.230.0.0/16* to monitor. We can pick any subnet, apart from the subnet that our VM is attached to. Below is the modified code.

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff( filter='net 128.230.0.0/16', prn=print_pkt)
```

So, when we *ping* 128.230.0.1, we capture the ICMP packets destined for 128.230.0.1 as shown below, thus sniffing the packets going to the subnet 128.230.0.0/16.

```
root@VM: /home/seed/Downloads/Labsetup# ./sniffer.py
###[ Ethernet ]###
dst      = 52:54:00:12:35:02
src      = 08:00:27:b7:29:8a
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 17002
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x6b49
src      = 10.0.2.15
dst      = 128.230.0.1
\options \
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x92aa
id       = 0xd
seq      = 0x1
###[ Raw ]###
load     = '\x8df'e\x00\x00\x00\x00\xb0\xa8\x00\x00\x00\x00\x00\x00
\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&')(*+,-./01234567'

###[ Ethernet ]###
dst      = 08:00:27:b7:29:8a
src      = 52:54:00:12:35:02
type     = IPv4

seed@VM: ~/../Labsetup$ ping 128.230.0.1
[11/24/23] seed@VM:~/../Labsetup$ ping 128.230.0.1
PING 128.230.0.1 (128.230.0.1) 56(84) bytes of data.
64 bytes from 128.230.0.1: icmp_seq=1 ttl=48 time=83.9 ms
64 bytes from 128.230.0.1: icmp_seq=2 ttl=48 time=82.1 ms
64 bytes from 128.230.0.1: icmp_seq=3 ttl=48 time=90.4 ms
^C
--- 128.230.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 82.111/85.455/90.378/3.554 ms
[11/24/23] seed@VM:~/../Labsetup$
```

Task 1.2: Spoofing ICMP Packets

First, I create `spoof.py` file with the below contents. Here `a.src` is the spoofed IP and the packet is sent to `a.dst`. So, the destination IP would think that the packet is sent from the mentioned source 50.1.2.3, where it was actually sent by attacker at 10.9.0.1.

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

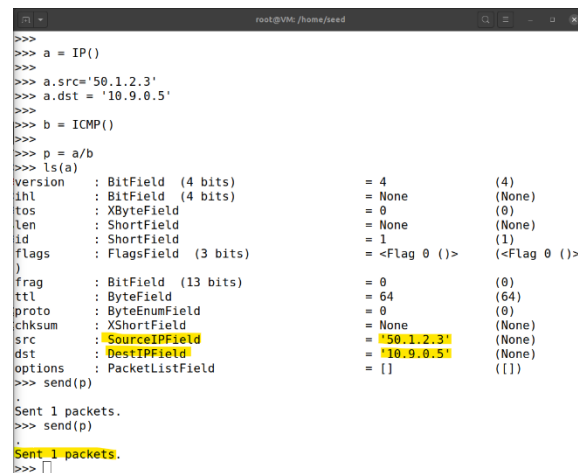
a = IP()

a.src='50.1.2.3'
a.dst = '10.9.0.5'

b = ICMP()

p = a/b
send(p)
```

If I run the above code in python IDLE and see the `ls(a)` output, we see the source IP is modified, as below.



```
root@VM: /home/seed
>>> a = IP()
>>> a.src='50.1.2.3'
>>> a.dst = '10.9.0.5'
>>> b = ICMP()
>>> p = a/b
>>> ls(a)
version      : BitField (4 bits)          = 4          (4)
ihl          : BitField (4 bits)          = None       (None)
tos          : XByteField (1 byte)        = 0          (0)
len          : ShortField (2 bytes)       = None       (None)
id           : ShortField (2 bytes)       = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 (> (<Flag 0 (>)
)
frag         : BitField (13 bits)         = 0          (0)
ttl          : ByteField (1 byte)         = 64         (64)
proto        : ByteEnumField (1 byte)     = 0          (0)
chksum       : XShortField (2 bytes)      = None       (None)
src          : SourceIPField (4 bytes)    = '50.1.2.3' (None)
dst          : DestIPField (4 bytes)      = '10.9.0.5' (None)
options      : PacketListField (0 bytes)  = []         ([])
>>> send(p)
Sent 1 packets.
>>> send(p)
Sent 1 packets.
>>> []
```

When I run this `spoof.py` script it sends out one packet, as shown below.

```
root@VM:/home/seed/Downloads/Labsetup# ls
docker-compose.yml sniffer.py spoof.py volumes
root@VM:/home/seed/Downloads/Labsetup# cat spoof.py
#!/usr/bin/env python3
from scapy.all import *

a = IP()

a.src='50.1.2.3'
a.dst = '10.9.0.5'

b = ICMP()

p = a/b
send(p)
root@VM:/home/seed/Downloads/Labsetup# ./spoof.py
Sent 1 packets.
root@VM:/home/seed/Downloads/Labsetup#
```

We observe the activity on Wireshark for the interface starting with `br-...` and see that there was indeed an ICMP *echo-request* packet sent from 50.1.2.3 to 10.9.0.5, which was later accepted with an ICMP *echo-reply* packet (row 3 & 4 below), where it was actually sent by attacker at 10.9.0.1.

[SEED Labs] Capturing from br-8a184f62243c						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-F>						
No.	Time	Source	Destination	Protocol	Length	Info
1	2023-11-24 18:00:33.906401201	02:42:77:7e:ea:e9	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
2	2023-11-24 18:00:33.906962939	02:42:8a:09:00:05	02:42:77:7e:ea:e9	ARP	42	10.9.0.5 is at 02:42:8a:09:00:05
3	2023-11-24 18:00:33.927597128	50.1.1.1	10.9.0.5	ICMP	42	Echo (ping) Request id=0x0000, seq=0/0, ttl=64 (reply in 4)
4	2023-11-24 18:00:33.928293044	10.9.0.5	50.1.1.1	ICMP	42	Echo (ping) Reply id=0x0000, seq=0/0, ttl=64 (request in 3)
5	2023-11-24 18:00:39.153872329	02:42:8a:09:00:05	02:42:77:7e:ea:e9	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
6	2023-11-24 18:00:39.153914054	02:42:77:7e:ea:e9	02:42:8a:09:00:05	ARP	42	10.9.0.1 is at 02:42:77:7e:ea:e9

▶ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-8a184f62243c, id 0 ▶ Ethernet II, Src: 02:42:77:7e:ea:e9 (02:42:77:7e:ea:e9), Dst: Broadcast (ff:ff:ff:ff:ff:ff) ▶ Address Resolution Protocol (request) Hardware type: Ethernet (1) Protocol type: IPv4 (0x0800) Hardware size: 6 Protocol size: 4 Opcode: request (1) Sender MAC address: 02:42:77:7e:ea:e9 (02:42:77:7e:ea:e9) Sender IP address: 10.9.0.1 Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00) Target IP address: 10.9.0.5						
---	--	--	--	--	--	--

0000	ff ff ff ff ff ff 02 42 77 7e ea e9 08 00 01B W.....
0010	08 00 06 04 00 01 02 42 77 7e ea e9 0a 09 00 01B W.....
0020	00 00 00 00 00 00 0a 09 00 05

Task 1.3: Traceroute

In this task we use Scapy to estimate the distance, in terms of number of routers, between the VM and a selected destination. In other words, we have to replicate the traceroute tool. So, I created the below traceroute.py script. In which the TTL starts from 1 and goes up till 64, which is the commonly used upper limit. I have chosen the destination as 50.1.1.1. We send the packet and receive the reply using scapy's `srl()`. If the reply is not null, the router IP is printed and in case if the IP matches with the destination IP, we print the total hops and break out of the loop and exit the program.

```
#!/usr/bin/env python3
from scapy.all import *
a=IP()
a.dst = '50.1.1.1'
b=ICMP()

for i in range(1,65):
    a.ttl = i
    reply=srl(a/b,verbose=False)
    if reply!=None:
        print(str(a.ttl)+'-> '+reply.src)
        if(a.dst==reply.src):
            print("Total TTL or HOPs= ",i)
            break
```

Below is the output of the program. It prints all the routers' IP addresses the packet goes through, on its way to the destination.

```
root@VM:/home/seed/Downloads/Labsetup# ./traceroute.py
1-> 10.0.2.2
2-> 10.25.0.1
3-> 66.215.13.81
4-> 159.111.156.35
5-> 72.129.5.116
6-> 209.18.43.72
7-> 66.109.5.247
8-> 62.115.156.224
9-> 62.115.119.90
10-> 62.115.181.227
11-> 135.180.179.146
12-> 135.180.179.166
13-> 50.1.36.6
14-> 50.1.1.1
Total TTL or HOPs= 14
root@VM:/home/seed/Downloads/Labsetup#
```

The IP addresses can also be collected from the wireshark tool, as at each hop when the TTL expires, the router sends back an ICMP *TTL expire* packet back to the sender, along with its own IP. The wireshark output is as below.

[SEED Labs] Capturing from enp0s3						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	2023-11-24 19:00:48.71071723	PcsCompu_b7:29:8a	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
2	2023-11-24 19:00:48.711011474	RealtekU12:35:02	PcsCompu_b7:29:8a	ARP	66	10.0.2.2 is at 52:54:00:12:35:02
3	2023-11-24 19:00:48.728505769	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f...
4	2023-11-24 19:00:48.728825650	10.0.2.2	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
5	2023-11-24 19:00:48.771369988	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f...
6	2023-11-24 19:00:48.773971750	10.25.0.1	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	2023-11-24 19:00:48.823559974	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f...
8	2023-11-24 19:00:48.825999580	66.215.13.81	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	2023-11-24 19:00:48.866886713	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f...
10	2023-11-24 19:00:48.971010980	152.15.116.35	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	2023-11-24 19:00:48.908227758	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f...
12	2023-11-24 19:00:48.913725345	72.129.5.116	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
13	2023-11-24 19:00:48.943483812	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f...
14	2023-11-24 19:00:48.951407330	200.18.43.72	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
15	2023-11-24 19:00:48.979495103	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f...
16	2023-11-24 19:00:48.985093880	66.109.5.247	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	2023-11-24 19:00:49.019355514	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f...
18	2023-11-24 19:00:49.028150342	34.15.156.224	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	2023-11-24 19:00:49.071995191	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response f...
20	2023-11-24 19:00:49.087653085	62.115.119.90	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	2023-11-24 19:00:49.119058799	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response ...
22	2023-11-24 19:00:49.143208568	62.115.181.227	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	2023-11-24 19:00:49.175873742	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response ...
24	2023-11-24 19:00:49.197055917	135.180.179.140	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
25	2023-11-24 19:00:49.232639051	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response ...
26	2023-11-24 19:00:49.251042780	135.180.179.160	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
27	2023-11-24 19:00:49.295320995	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response ...
28	2023-11-24 19:00:49.315047394	50.1.36.0	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
29	2023-11-24 19:00:49.359828175	10.0.2.15	50.1.1.1	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=14 (reply in 30)
30	2023-11-24 19:00:49.496144728	50.1.1.1	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=240 (request in ...

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0

Ethernet II, Src: PcsCompu_b7:29:8a (08:00:27:b7:29:8a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

Address Resolution Protocol (request)

```

0000  ff  ff  ff  ff  ff  08  00  27  b7  29  8a  08  06  00  01  .....f.....
0010  08  00  06  04  00  01  08  00  27  b7  29  8a  0a  00  02  0f  .....f.....

```

Task 1.4: Sniffing and-then Spoofing

For this task I created the below *sniffAndSpoof.py* script which sniffs ICMP packets and then constructs echo reply packets using fields information from the request packets. The code is shown below.

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

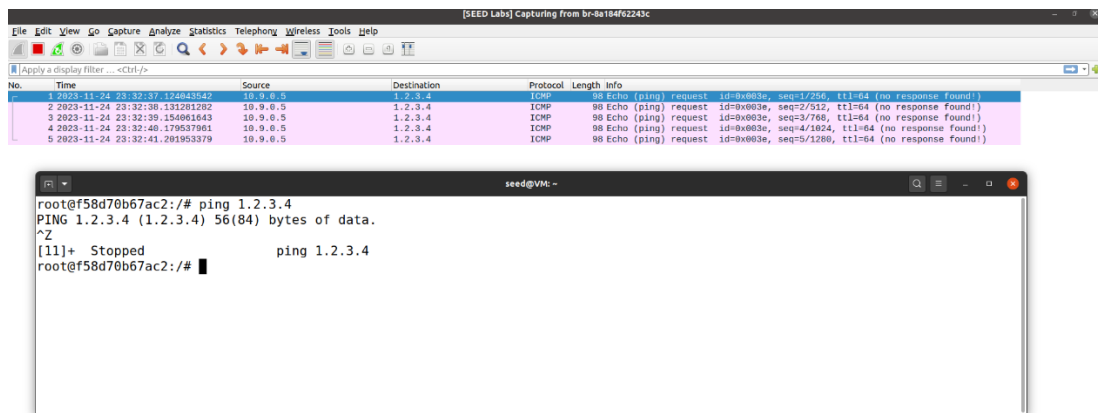
def print_pkt(pkt):
    a=IP(src=pkt[IP].dst, dst=pkt[IP].src)
    b=ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
    c=pkt[Raw].load
    reply=a/b/c
    send(reply)

pkt = sniff(iface='br-8a184f62243c', filter='icmp', prn=print_pkt)

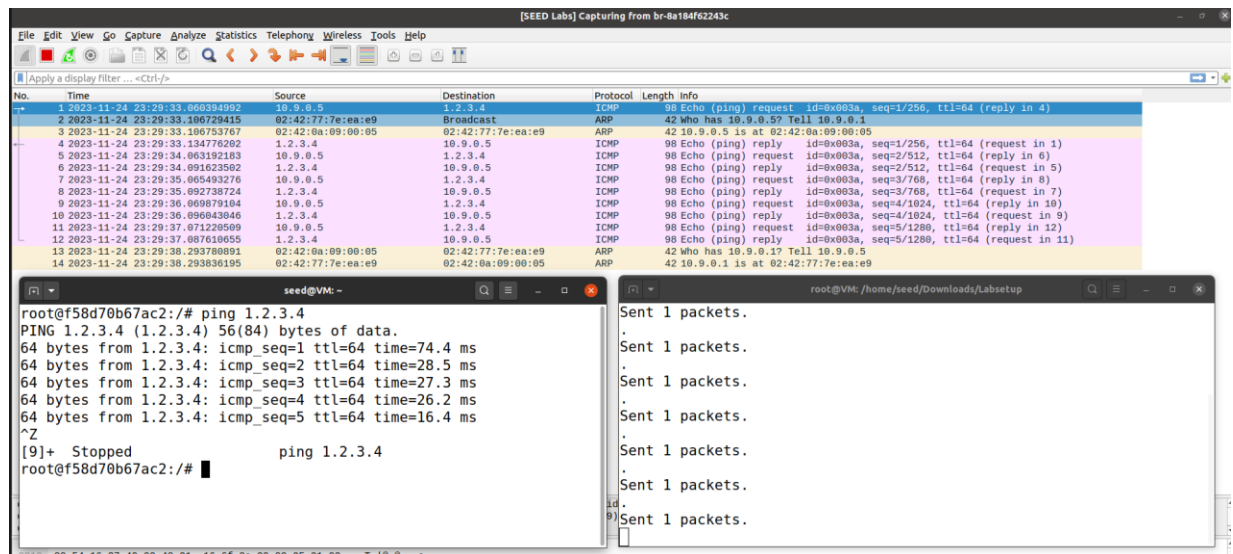
~
```

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

Since this is a non- existing host on the Internet, when we first try to ping this IP without running our *sniffAndSpoof.py* script we get no reply from the destination as shown below.



But when we again ping the same IP, but this time running our *sniffAndSpoof.py* script we see that to our *echo-request* packet, we get back *echo-response*. This response is not from the actual destination, since it is non existing, but spoofed from our attacker VM i.e. from 10.9.0.1. The terminal and wireshark results are shown below.



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

Since this is a nonexistent host on the same LAN, the ARP request is not resolved for this IP since no MAC address is attached to this IP. Hence, we get no ICMP echo request/response. Below is the screenshot.

[SEED Labs] Capturing from br-8a1b4f62243c									
No.	Time	Source	Destination	Protocol	Length	Info			
1	2023-11-24 23:27:22.115408921	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
2	2023-11-24 23:27:23.125198063	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
3	2023-11-24 23:27:24.147389113	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
4	2023-11-24 23:27:25.172740387	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
5	2023-11-24 23:27:26.195196411	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
6	2023-11-24 23:27:27.218466678	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
7	2023-11-24 23:27:28.247638379	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
8	2023-11-24 23:27:30.261936929	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	
9	2023-11-24 23:27:31.282324645	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99?	Tell	10.9.0.5	

```

root@f58d70b67ac2:~# 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
^Z
[8]+  Stopped                  ping 10.9.0.99
root@f58d70b67ac2:~#

```

```

root@VM: /home/seed/Downloads/Labsetup# ./sniffAndSpoof.py

```

- ping 8.8.8.8 # an existing host on the Internet

Since this is an existing host on the Internet, we get a response back from 8.8.8.8 to our ping request, as shown below. But since the attacker at 10.9.0.1 is also sending ICMP *echo-reply* packets, for each *echo-request* we get two *echo-reply* packet as shown in wireshark screenshot below. Also, for the duplicate responses we see (*DUP!*) messages, indicating duplicate response, in the terminal window, as below.

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-11-24 23:18:50.642848049	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0036, seq=1/256, ttl=64 (reply in 2)
2	2023-11-24 23:18:50.651280063	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=1/256, ttl=114 (request in 1)
3	2023-11-24 23:18:50.679268085	02:42:77:7e:ea:e9	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
4	2023-11-24 23:18:50.679301012	02:42:0a:09:00:05	02:42:77:7e:ea:e9	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
5	2023-11-24 23:18:50.695090270	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=1/256, ttl=64
6	2023-11-24 23:18:51.645125380	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0036, seq=2/512, ttl=64 (reply in 7)
7	2023-11-24 23:18:51.670667705	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=2/512, ttl=64 (request in 6)
8	2023-11-24 23:18:51.691725098	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=2/512, ttl=114
9	2023-11-24 23:18:52.648379166	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0036, seq=3/768, ttl=64 (reply in 10)
10	2023-11-24 23:18:52.659130602	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=3/768, ttl=114 (request in 9)
11	2023-11-24 23:18:52.669931738	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0036, seq=3/768, ttl=64

```

root@f58d70b67ac2:~# 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=114 time=8.66 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=52.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=25.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=46.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=10.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=21.6 ms (DUP!)
^Z
[7]+  Stopped                  ping 8.8.8.8
root@f58d70b67ac2:~#

```

```

root@VM: /home/seed/Downloads/Labsetup# ./sniffAndSpoof.py
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.

```