Name: Dhanashree Reddy Srinivasa Reddy

SUID: 393473169

NetId: dsriniva

Course: Internet Security

Task 1.1A

In the above program, for each captured packet, the callback function print pkt() will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

Solution:

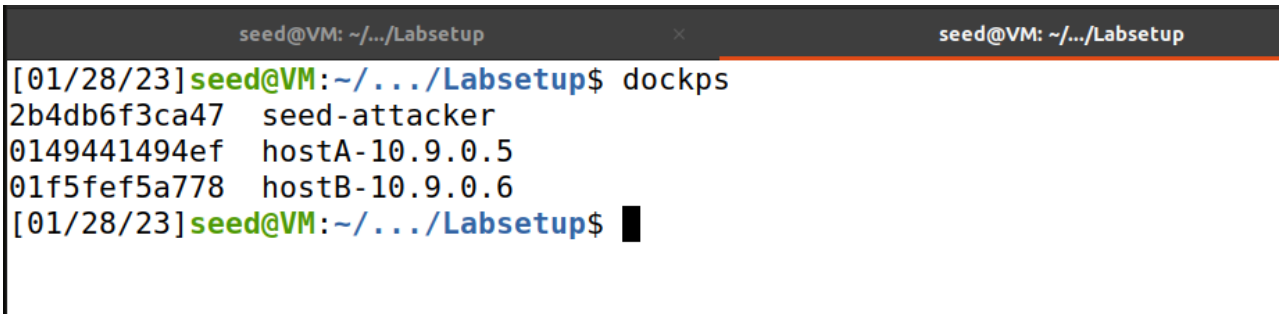
Initially we run docker compose commands:

$ dcbuild

$ dcup

$ dcdown

The containers run in the background. Then we use 'dockps' command to get the ID of the container and then the 'docksh <ID>' command.



```
seed@VM: ~/.../Labsetup          seed@VM: ~/.../Labsetup
[01/28/23]seed@VM:~/.../Labsetup$ dockps
2b4db6f3ca47  seed-attacker
0149441494ef  hostA-10.9.0.5
01f5fef5a778  hostB-10.9.0.6
[01/28/23]seed@VM:~/.../Labsetup$ docksh 01f
root@01f5fef5a778:/#
```

```
  GNU nano 4.8                          Q1a.py
#!/usr/bin/env python3
from scapy.all import*

def print_pkt(pkt):
  pkt.show()

pkt = sniff(iface='br-323eba80cf94',filter='icmp',prn=print_pkt)
```

Code: python3 Q1a.py

In the program we are using scapy tool which is a building tool for all sniffing and spoofing tool.The interface value ‘iface’ is retrieved by using ifconfig in terminal, the iface value is the value where the packets are sniffed from. The next parameter ‘filter’ with value ‘icmp’ will show us only ICMP packets.

To execute the program, we should run the python using the root privilege because the privilege is required for spoofing packets.

chmod a+x Q1a.py

a+x means execute + all

Using the ‘ping ’ command with ‘8.8.8.8’ on the host machine for ICMP echo request we receive packets .

```
root@01f5fef5a778:/# 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=61 time=55.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=9.21 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=61 time=9.73 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=61 time=9.85 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=61 time=9.16 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=61 time=9.46 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=61 time=9.57 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=61 time=12.6 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=61 time=11.0 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=61 time=9.01 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=61 time=9.09 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=61 time=11.3 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=61 time=9.25 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=61 time=8.96 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=61 time=8.89 ms
```

Output: On the attacker machine

```
root@VM:/volumes# python3 Q1a.py
###[ Ethernet ]###
  dst       = 02:42:42:02:c4:55
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 9998
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf97d
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
```

Next when we try to execute with root permissions below is the output.

'su seed'

```
root@VM:/volumes# su seed
seed@VM:/volumes$ python3 Q1a.py
Traceback (most recent call last):
  File "Q1a.py", line 7, in <module>
    pkt = sniff(iface='br-323eba80cf94',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, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # 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
```

The above error "permissionError: Operation not permitted' is because of the permission. When we try to run the above code in 'root' we can see the whole network traffic in the given interfaces. If we want to sniff packets, we need root privileges to see the traffic and to capture the relevant packets.

Task 1.1B. Usually, when we sniff packets, we are only interested in certain types of packets. We can do that by setting filters in sniffing. Scapy's filter uses the BPF (Berkeley Packet Filter) syntax; you can find the BPF manual from the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

• Capture only the ICMP packet

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

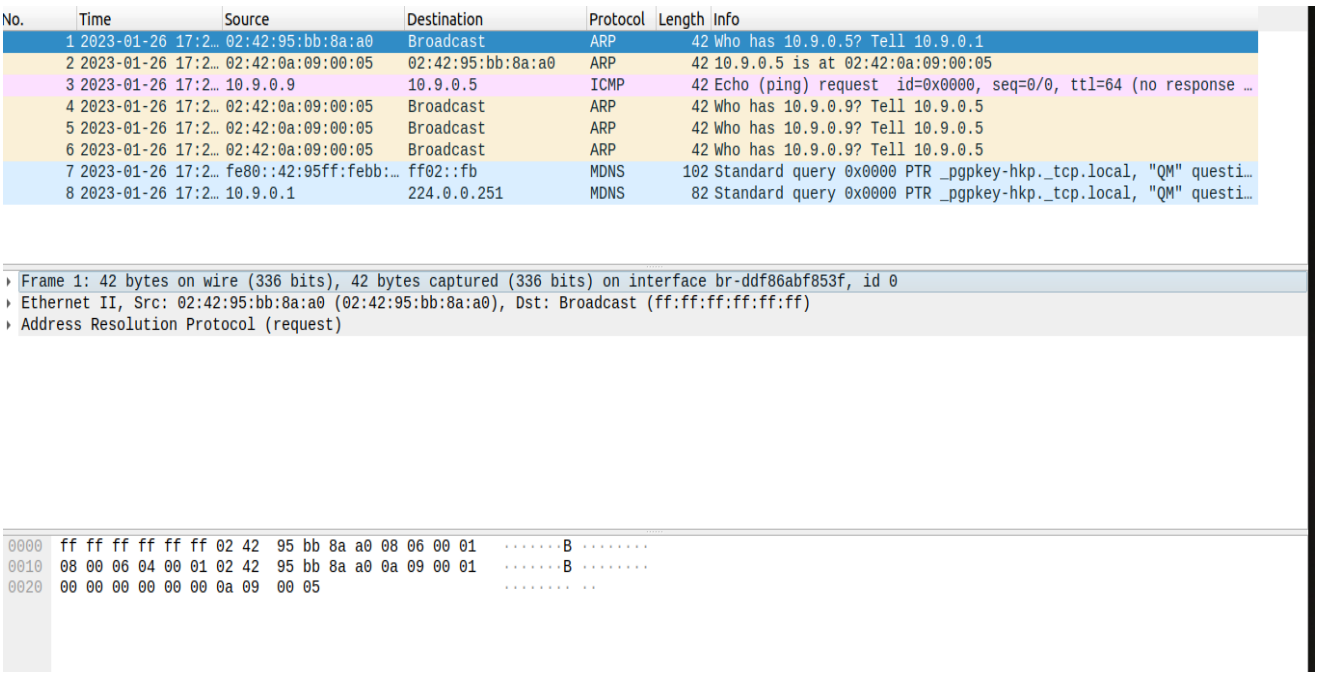
Solution:

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

def print_pkt(pkt):
  pkt.show()

pkt = sniff(iface='br-323eba80cf94',filter='tcp and dst port 23 and src host 10.9.0.5' ,prn=print_pkt)
```

In the filter with ‘tcp port 23 and src host 10.9.0.5 .

The default VM’s IP is ‘10.9.0.5’ and I sent it to ‘10.9.0.9’ which is another IP.

To receive the packets, execute command telnet 10.9.0.9 from the host machine. The program Q1b.py sniffed the TCP packets. We use telnet since TCP port number for telnet is 23.



• Capture packets that come 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.

Solution:

Code: python3 Q1b.py

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

def print_pkt(pkt):
  pkt.show()

pkt = sniff(iface='br-323eba80cf94',filter='dst net 234.143.141.0/24' ,prn=print_pkt)
```

```
root@VM:/volumes# python3 Q1b.py
###[ Ethernet ]###
  dst       = 01:00:5e:0f:8d:09
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 0
     flags     = DF
     frag      = 0
     ttl       = 1
     proto     = icmp
     chksum    = 0xf802
     src       = 10.9.0.5
     dst       = 234.143.141.9
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7660
        id        = 0x27
```

Filter: ‘dst net 234.143.141.0/24’. ‘dst’ means a possible direction. ‘net’ returns true if there is a possible type of net, in the above program it is the subnet. A packet was sent to a particular subnet and the program sniffed only packets that were sent from source ‘10.9.0.5’ to destination IP from the other subnet.

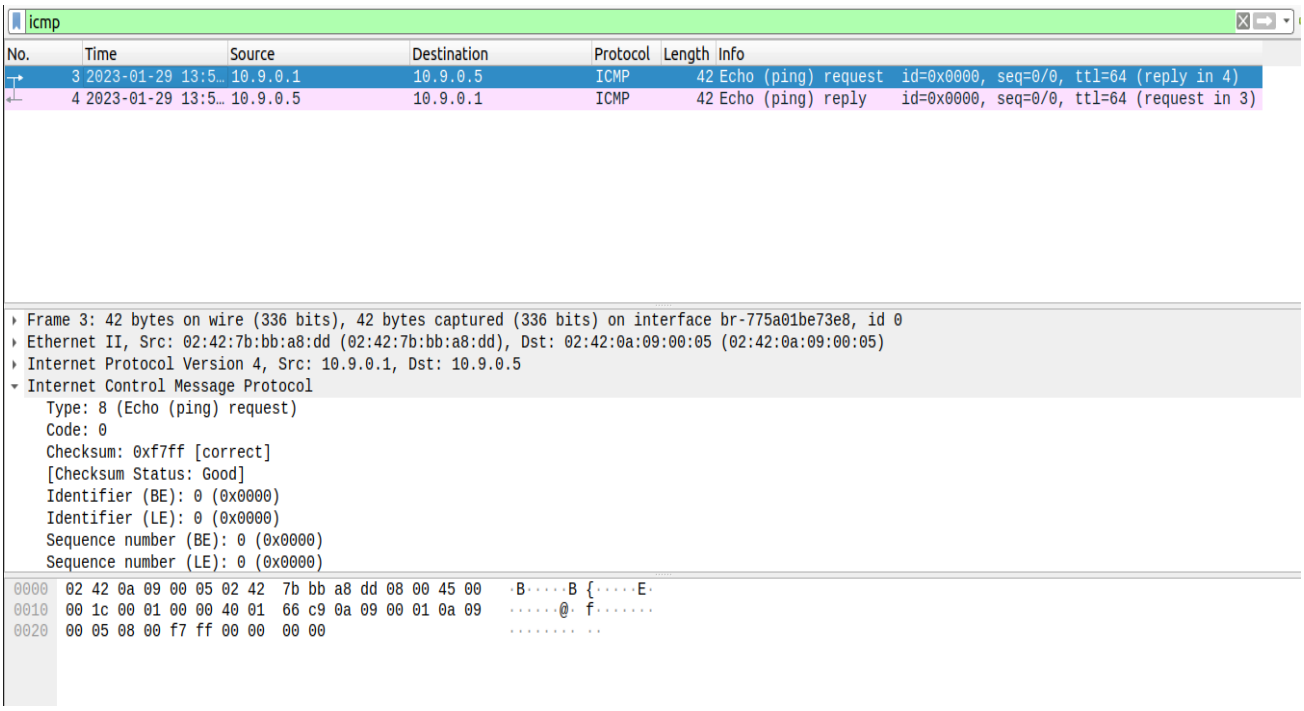
Task 1.2: Spoofing ICMP Packets Please make any necessary changes to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.

Solution:

Code: python3 Q2a.py

```
  GNU nano 4.8                                  Q2a.py
from scapy.all import *
a=IP()
a.dst = '10.9.0.5'
b = ICMP()
p = a/b
send(p)
ls(a)
```

```
root@VM:/volumes# python3 Q2a.py
.
Sent 1 packets.
version    : BitField  (4 bits)                = 4               (4)
ihl        : BitField  (4 bits)                = None            (None)
tos        : XByteField                        = 0               (0)
len        : ShortField                        = None            (None)
id         : ShortField                        = 1               (1)
flags      : FlagsField  (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField  (13 bits)               = 0               (0)
ttl        : ByteField                         = 64              (64)
proto      : ByteEnumField                     = 0               (0)
chksum     : XShortField                       = None            (None)
src        : SourceIPField                     = '10.9.0.1'      (None)
dst        : DestIPField                       = '10.9.0.5'      (None)
options    : PacketListField                   = []              ([])
```



Another VM (IP destination ‘10.9.0.5’) and sent an ICMP packet using a random IP source ‘10.9.0.1’.

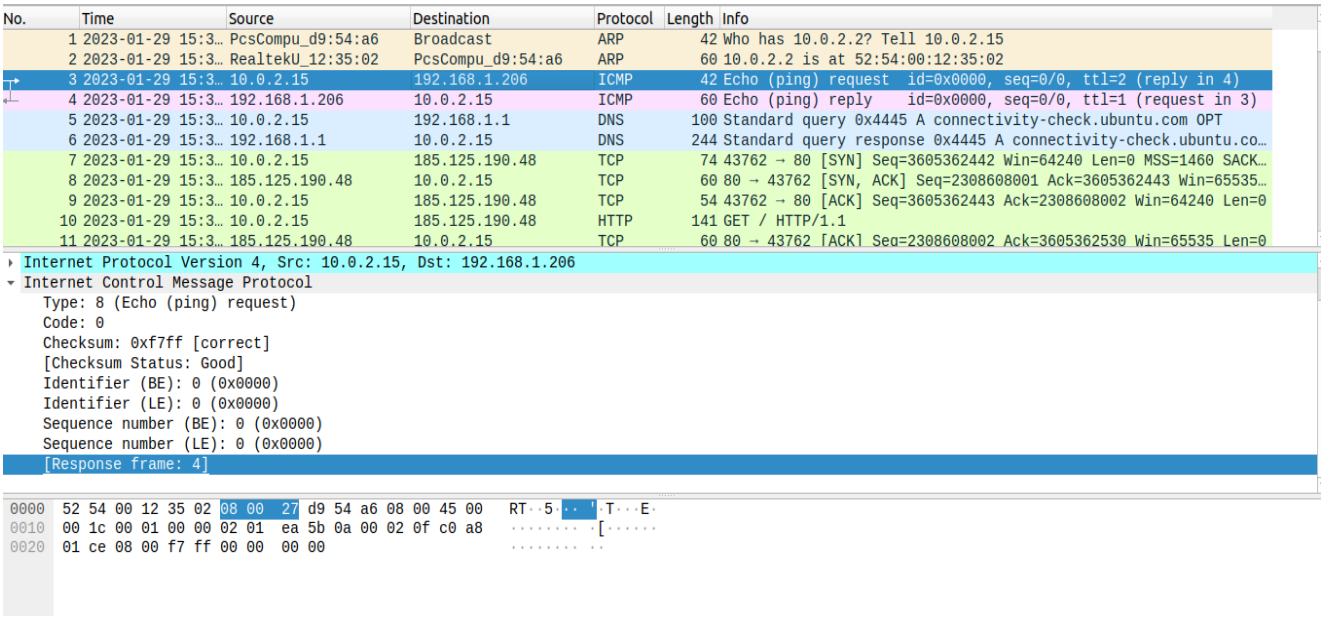
Spoofed ICMP echo request packet and sent it to another VM on the same subnet. We can use Wireshark to observe whether our request will be accepted by the receiver. The source ip was overwritten with our own ip: 10.9.0.1 and sent the packet to destination 10.9.0.5; the packet was received by 10.9.0.5 and sent an echo reply to 10.9.0.1

Task 1.3: Traceroute

Code: python3 Q3a.py

```
 GNU nano 4.8                                Q3a.py
from scapy.all import *
a = IP()
b= ICMP()
a.dst = '192.168.1.206'
a.ttl= 2
h = sr1(a/b, timeout=2, verbose=0)
if h is None:
   print("Router: *** (hops = {})".format(a.ttl))
else:
   print("Router: {} (hops = {})".format(h.src, a.ttl))
```

```
seed@VM: ~/.../Labsetup        seed@VM: ~/.../volumes
root@VM:/volumes# python3 Q3a.py
Router: 192.168.1.206 (hops = 2)
root@VM:/volumes#
```



I asked for the destination IP ‘192.168.1.206’. The flag ‘ttl’ of the packet increases by one in each given packet. The sr1() method of Scapy is a method which will listen

and wait for a packet response. (timeout = time limit for response, verbose = ignore printing unnecessary details).

This program figures out how many routers (hops) it takes to send out that packet all the way to the IP address destination. Each line at the display is a different router. The time- to-live is used to return an error of each hop till the destination, this way we can print each IP router till it stops. In this case the ttl value is 2, in which the response returned with a response frame of 4.

Task 1.4 Sniffing and Spoofing

```
 GNU nano 4.8                          Q4.py
#!usr/bin/python3
from scapy.all import *

def spoof_pkt(pkt):
        if ICMP in pkt and pkt[ICMP].type == 8:
                print("Original Packet...")
                print("Source IP ", pkt[IP].src)
                print("Destination IP:", pkt[IP].dst)

                ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
                icmp = ICMP(type=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
                data = pkt[Raw].load
                newpkt= ip/icmp/data

                print("poofed packet...")
                print("Source IP: ", newpkt[IP].src)
                print("Destination IP; ",newpkt[IP].dst)
                send(newpkt,verbose=0)
filter= 'icmp'
pkt = sniff(iface='br-775a01be73e8',prn=spoof_pkt)
```

In the above code the ‘if’ block checks if an ICMP is a request. And if it’s true, the reply packet will be based on details derived from the original packet, but the dst and src is exchanged so whenever it sees an ICMP echo request, regardless of what the target IP address is, the program should immediately send out an echo reply using this packet spoofing technique.

The pkt[Raw].load is used to store the original packet data payload this way it Will return properly to the sender.

In this task as mentioned in the question used 3 different IPs to ping. The program ‘Q4.py’ will sniff for any ICMP packets in the subnet, and when it catches one, the

program will return to the sender an ICMP reply packet back. So even if the IP echo request isn't available at all, the program will always return to the sender a reply packet. On the attacker VM the python is executed and on the host machine the 3 different IP is pinged.

Ip Address: 1.2.3.4

```
root@0149441494ef:/# 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=64 time=21.6 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=5.11 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=3.34 ms

--- 1.2.3.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 3.335/10.016/21.606/8.226 ms
```

```
root@VM:/volumes# python3 Q4.py
filter: icmp and host 1.2.3.4

Original Packet...
Source IP  10.9.0.5
Destination IP: 1.2.3.4
poofed 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
poofed 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
poofed packet...
Source IP:  1.2.3.4
Destination IP;  10.9.0.5
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2023-01-29 22:5…	10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request id=0x004e, seq=1/256, ttl=63 (no respons…
2	2023-01-29 22:5…	10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request id=0x004e, seq=2/512, ttl=63 (no respons…
3	2023-01-29 22:5…	142.250.190.74	10.0.2.15	TLSv1.2	139	Application Data
4	2023-01-29 22:5…	10.0.2.15	142.250.190.74	TCP	54	41232 → 443 [ACK] Seq=1926198211 Ack=3518216540 Win=63714 Len…
5	2023-01-29 22:5…	10.0.2.15	1.2.3.4	ICMP	98	Echo (ping) request id=0x004e, seq=3/768, ttl=63 (no respons…
6	2023-01-29 22:5…	10.0.2.15	185.125.190.56	NTP	90	NTP Version 4, client
7	2023-01-29 22:5…	185.125.190.56	10.0.2.15	NTP	90	NTP Version 4, server
8	2023-01-29 22:5…	PcsCompu_d9:54:a6	RealtekU_12:35:02	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
9	2023-01-29 22:5…	RealtekU_12:35:02	PcsCompu_d9:54:a6	ARP	60	10.0.2.2 is at 52:54:00:12:35:02
10	2023-01-29 22:5…	10.0.2.15	192.168.1.1	DNS	86	Standard query 0x6fe6 AAAA mail.google.com OPT
11	2023-01-29 22:5…	192.168.1.1	10.0.2.15	DNS	114	Standard query response 0x6fe6 AAAA mail.google.com AAAA 2607…

```
▸ Frame 8: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
▸ Ethernet II, Src: PcsCompu_d9:54:a6 (08:00:27:d9:54:a6), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Address Resolution Protocol (request)
```

```
0000  52 54 00 12 35 02 08 00  27 d9 54 a6 08 06 00 01   RT··5··· '·T·····
0010  08 00 06 04 00 01 08 00  27 d9 54 a6 0a 00 02 0f   ········ '·T·····
0020  00 00 00 00 00 00 0a 00  02 02                     ········ ··
```

The host machine sends a ping to '1.2.3.4' which is a non-existing host on the Internet. With the program we will get a 0% packet loss. We can see in the screenshot of the wireshark that the ARP protocol is asking for 1.2.3.4 and asking on the network who has that IP destination? So, the attacker returns with an answer to it and the ICMP packet reply is coming back to the host machine.

IP Address: 10.9.0.99

```
root@0149441494ef:/# 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 2040ms
pipe 3
```

The attacker VM sends a ping to '10.9.0.99' which is a non-existing host on the LAN. In this scenario we're getting the same concept with the same idea of the ARP protocol. Even though the host doesn't even exist. The program from the host machine will send back an ICMP response packet.

IP address: 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=64 time=22.0 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=61 time=30.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=7.85 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=30.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=13.6 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +2 duplicates, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 7.850/20.850/30.674/9.010 ms
```

```
root@VM:/volumes# python3 Q4.py
Original Packet...
Source IP  10.9.0.5
Destination IP: 8.8.8.8
poofed 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
poofed 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
poofed packet...
Source IP:  8.8.8.8
Destination IP;  10.9.0.5
```

The attacker machine sends a ping to ‘8.8.8.8’ which is an existing host on the Internet. This scenario is different from the other IP addresses because it exists on the net. So, in this case we’re getting duplicate responses, that’s because there are two responses, the real destination is responding to the source, and even the host machine is also responding to the source.