Packet Sniffing and Spoofing Lab

Wang,Xiao: xwang99@syr.edu
SUID:495425405

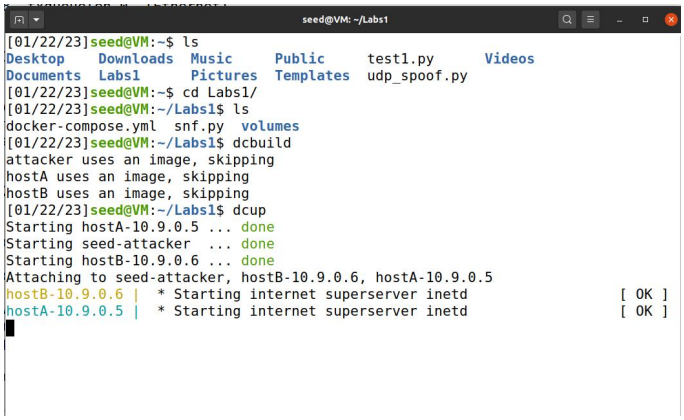
0. Lab environment setup and preparation

1. Build the lab environment:

Step 0 :Before I up our dockers, I create a folder named Labs1 and download our setting file into it.
Step 1: I change the path to Labs1 (**cd Labs1**).
Step2: build our environment (**dcbuild dcup**)

1.1 Output:

1.1.1

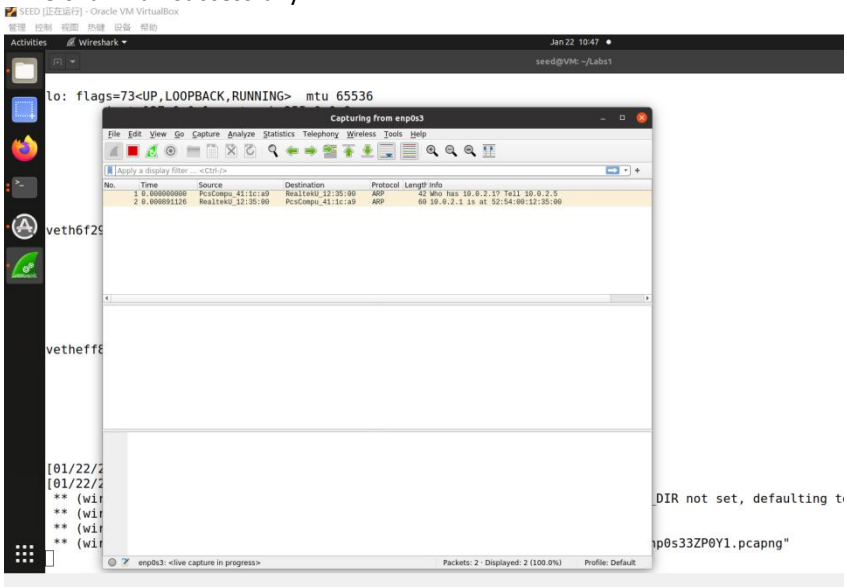


1.1.2 I open another terminal use the ifconfig to check the interface configuration.
The **br-ec3907382b6b** is our corresponding network interface.

2. Download & Setup Wireshark

Step1: Update the APT package repository cache (**$ sudo apt update**)
Step2: Install Wireshark (**$ sudo apt install wireshark**)
Step3: To run the Wireshark, it requires the root privilege (**$ sudo wireshark**)

2.1 Output:
Wire-shark run successfully



3. BPF explore:

The Berkeley Packet Filter (BPF) is a technology used in certain computer operating systems for programs that need to, among other things, analyze network traffic.
There is a great reference for BPF : https://www.gigamon.com/content/dam/resource-library/english/guide---cookbook/gu-bpf-reference-guide-gigamon-insight.pdf

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

Task 1.1: Sniffing Packets

Task 1.1A.

(Experiment with root privilege)

Set the sniffing side:

1. get the super user privilege (**$ sudo su**)
2. Go the volumes folder write our python script (**$ cd /home/seed/Labs1/volumes/**)
3. Create a file named task1.1a.py and write code

Code:

From above(**$ ifconfig)** I can get the **br-ec3907382b6b** is our corresponding network interface then input set it to our code.

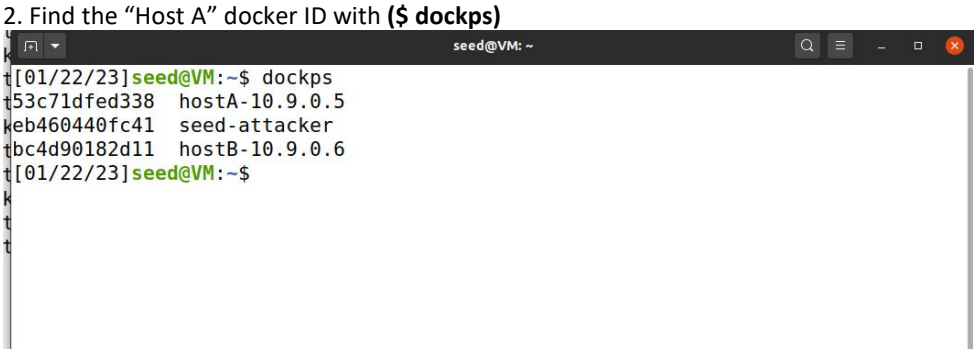
```
#!/user/bin/env/ python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt=sniff(iface='br-ec3907382b6b',filter='icmp',prn=print_pkt)

~
~
~
~
```

Run this code with Root privilege then it wait for packets.

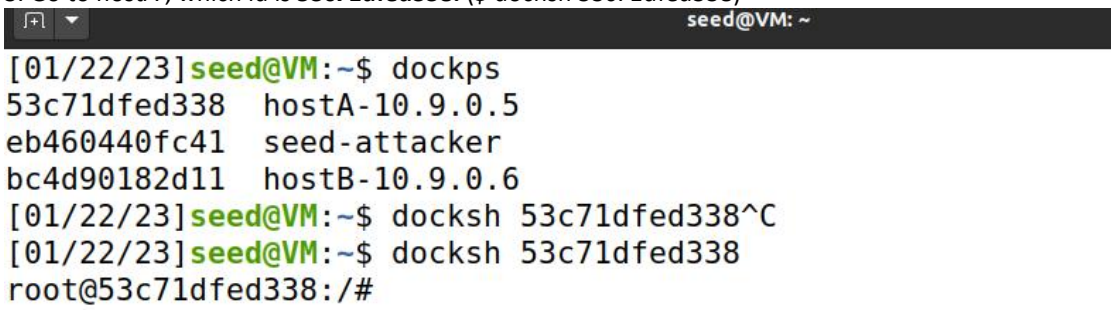
```
root@VM:/home/seed/Labs1/volumes# vim task1.1a.py
root@VM:/home/seed/Labs1/volumes# python3 task1.1a.py
```

Go to a host send a packet:

1. Open a new terminate
2. Find the “Host A” docker ID with **($ dockps)**

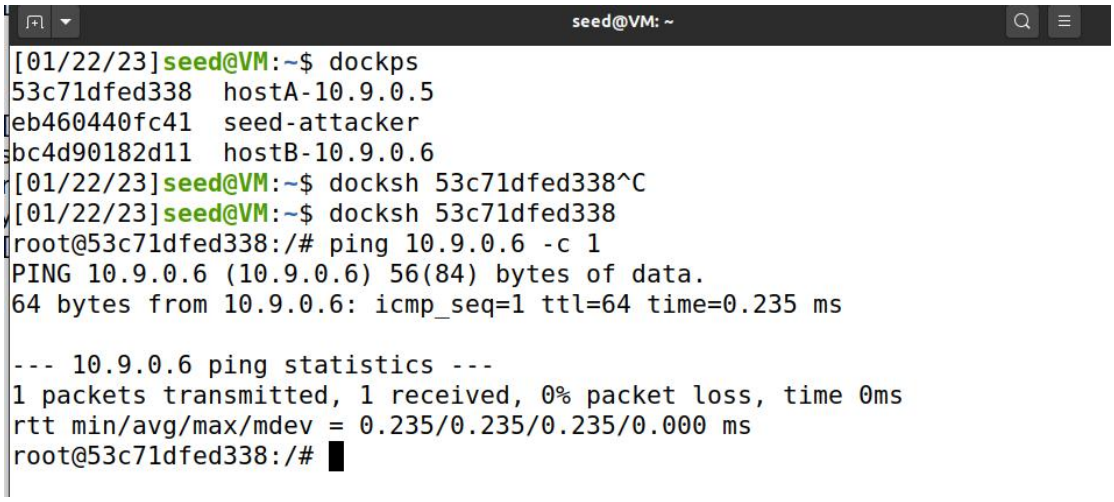


3. Go to hostA , which id is **53c71dfed338.** ($ dochsh 53c71dfed338)

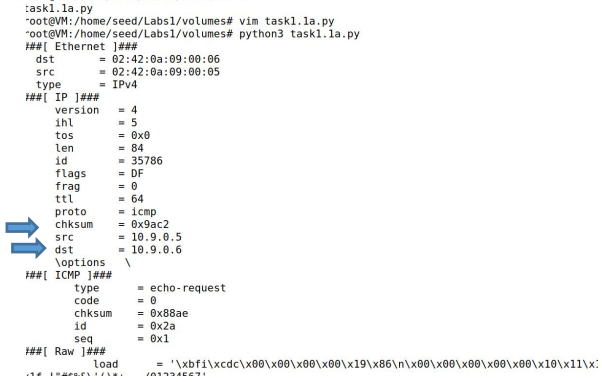


4. Now in hostA we ping to hostB(10.9.0.6) (**$ ping 10.9.0.6 -c 1**) "-c 1" means after one ping,ping stop.

What is ping:The "ping" command is a tool used to test the reachability of a network host. It works by sending an Internet Control Message Protocol (ICMP) Echo Request message to the target host, and waiting for an ICMP Echo Reply. It can be used to check if a host is online, and to measure the round-trip time for packets to travel from the source host to the target host and back



Captured package: we **captured the packet** from HostA (src=10.9.0.5) send to HostB (des=10.9.0.6).



(Experiment without root privilege)

1. Use ($ **su seed**) give up the root privilege
2. Execute the code again and get errors(permission error)

```
01/22/23]seed@VM:~/.../volumes$ python3 task1.1a.py
raceback (most recent call last):
  File "task1.1a.py", line 7, in <module>
    pkt=sniff(iface='br-ec3907382b6b',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)
ermissionError: [Errno 1] Operation not permitted
01/22/23]seed@VM:~/.../volumes$
```

Explain:

1. We need root privilege to turn the **NIC promiscuous mode** on.
When a NIC is operating in promiscuous mode, it will receive all packets on the network, regardless of the destination address. This allows the NIC to read and analyze all network traffic, including traffic that is not addressed to the host

2. Since the destination of the packets we want to capture is not our IP address, the operation system will drop the packets automatically. To solve the problem , we use raw socket to finish our task,which need the root privilege.

A raw socket is a type of socket that allows direct access to the underlying transport protocol. This means that the application can construct and send its own packets, rather than relying on the operating system to provide a higher-level interface.

Task 1.1B:

For this part, I still use Host A send a packet/packets to Host B.

1.1B.1
Capture only the ICMP packet:

Sniffing side:

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

def print_pkt(pkt):
    pkt.show()

pkt=sniff(iface='br-ec3907382b6b',filter='icmp',prn=print_pkt)

~
~
~
~
```

Host A:

```
seed@VM: ~
[01/22/23]seed@VM:~$ dockps
53c71dfed338  hostA-10.9.0.5
eb460440fc41  seed-attacker
bc4d90182d11  hostB-10.9.0.6
[01/22/23]seed@VM:~$ docksh 53c71dfed338^C
[01/22/23]seed@VM:~$ docksh 53c71dfed338
root@53c71dfed338:/# ping 10.9.0.6 -c 1
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=0.235 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.235/0.235/0.235/0.000 ms
root@53c71dfed338:/#
```

Captured package: we **captured the packet** from HostA (src=10.9.0.5) send to HostB (des=10.9.0.6), which is icmp

```
root@VM:/home/seed/Labs1/volumes# ls
task1.1a.py
root@VM:/home/seed/Labs1/volumes# vim task1.1a.py
root@VM:/home/seed/Labs1/volumes# python3 task1.1a.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        = 35786
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9ac2
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x88ae
        id        = 0x2a
        seq       = 0x1
###[ Raw ]###
           load      = '\xbfi\xcdc\x00\x00\x00\x00\x19\x86\n\x00\x00\x00\x00\x00\x10\x11\x1
```

1.1B.2

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

Sniffing side:
BPF : src host 10.9.0.5 and tcp port 23

Host A:
Since we want test TCP protocol, the Netcat (NC)is a good choice make things easy.
The grammar for NC is **nc [options] host port**, the default options is TCP/IP , you can use -u set as UDP.

```
--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.235/0.235/0.235/0.000 ms
root@53c71dfed338:/# nc 10.9.0.6 23
```

Captured package: we **captured the packet** from **HostA** (src=10.9.0.5) send to HostB (des=10.9.0.6), Proto is **tcp** and port is **telnet (23).**

```
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 46718
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x7021
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ TCP ]###
        sport     = 53736
        dport     = telnet
        seq       = 2219947464
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x144b
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (897388202, 0)), ('NOP', None), ('WScale', 7)]
```

1.1 B.3

Capture packets comes from or to go to a particular subnet.

I decide to capture packets send to the subnet 190.0.1.0/24, the IP range of the subnet is190.0.1.0-190.0.1.255. I use **host A** send a packet to **190.0.1.1**

Sniffing side:

BPF : net 190.0.1.0/24

```
root@VM: /home/seed/Labs1/volumes
#!/user/bin/env/ python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt=sniff(iface='br-ec3907382b6b',filter='net 190.0.1.0/24',prn=print_pkt,count=1)
```

HostA:

```
root@53c71dfed338:/# ping 190.0.1.1
PING 190.0.1.1 (190.0.1.1) 56(84) bytes of data.
```

Captured package: we **captured the packet** from **HostA** (src=10.9.0.5) send to HostB (des=10.9.0.1.1), (by the setting, we can capture all packets send to all subnets)

```
root@VM:/home/seed/Labs1/volumes# python3 task1.1b.3.py
###[ Ethernet ]###
  dst       = 02:42:45:3d:98:a3
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 45519
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xbfca
     src       = 10.9.0.5
     dst       = 190.0.1.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x87d
        id        = 0x2f
        seq       = 0x1
###[ Raw ]###
           load      = '\xf3}\xcdc\x00\x00\x00\x00\x00b\x9e\r\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
 !"#$%&\'()*+,-./01234567'
```

3.2 Task 1.2: Spoofing ICMP Packets

Attacker side:

I decide to use an arbitrary IP address to test, the **src =’1.2.3.4’** and I send this packet to the Host A

The packet sent successfully.

```
root@VM:/home/seed/Labs1/volumes# python3 task1.2.py
.
Sent 1 packets.
```

Wireshark:

1. opened a wireshark, and set capture to **br-ec3907382b6bA73VY1**
2. set filter only show icmp.
3. The source change to 1.2.3.4 (none-exist),which means we can spoof an ICMP echo request packet with an **arbitrary source IP Address**.

Task 1.3: Traceroute

Explore sr1():

The sr1() function in the Scapy library is used to send a packet and receive a response. The function takes several arguments, including the packet to be sent, the timeout value, and the verbosity level. The function returns the first response packet received or None if no response is received within the specified timeout.

If the sr1() function receives a response from a router, it returns a packet object that contains the ICMP "Time Exceeded" message returned by the router. The packet object has several attributes such as src (the source IP address of the router), dst (the destination IP address of the packet), ttl (the current TTL value of the packet), and others.

If the function does not receive a response within the specified timeout, it will return None.

Code:
The code just use simple while loop and for each line and annotation is write within the code.

```python
#!/user/bin/env/ python3
from scapy.all import *
icmp=ICMP();
ip=IP()
ip.dst="1.1.1.1" # dst is one of the atrribute of IP class
MaxTry=30 # The distination may not be reachable, set it to 30 to avoid infinity loop
TTL=0
StopFlag=True
while StopFlag and TTL< MaxTry:
    TTL+=1
    ip.ttl=TTL # set the current ttl value and for each loop increase 1
    hops=sr1(ip/icmp, timeout=2,verbose=0) # hops is the return value of sr1()
    if hops is None:# if return value is none, which means we cannot get the target by this TTL
        # print the TTL then go to the next loop.
        print("Router:*** (hops={})".format(TTL))
    else:# reach the dst and break the loop
        print("Router:{}(hops={})".format(hops.src,TTL))# the return value will send my a route
        # with it ip address, so we can use hops.src get the router ip address and print
        if hops.src==ip.dst:
            StopFlag=False
```

Output & verification correctness:

The output shows when we TLL=8 , we can reach the '1.1.1.1'

```
root@VM:/home/seed/Labs1/volumes# python3 task1.3.py
Router:10.0.2.1(hops=1)
Router:10.183.24.1(hops=2)
Router:173.230.5.13(hops=3)
Router:66.253.252.234(hops=4)
Router:173.230.125.67(hops=5)
Router:206.82.104.31(hops=6)
Router:199.27.132.36(hops=7)
Router:1.1.1.1(hops=8)
root@VM:/home/seed/Labs1/volumes#
```

Use traceroute to verification the correctness:

Use the traceroute -I 1.1.1.1
Rather than default tcp/ip or udp , we use (-I) for ICMP.
The out put is same,which means my design is correct.

```
root@VM:/home/seed/Labs1/volumes# traceroute -I 1.1.1.1
traceroute to 1.1.1.1 (1.1.1.1), 30 hops max, 60 byte packets
 1  _gateway (10.0.2.1)  0.437 ms  0.294 ms  0.213 ms
 2  10.183.24.1 (10.183.24.1)  4.327 ms  4.943 ms  4.855 ms
 3  173.230.5.13 (173.230.5.13)  4.783 ms  4.972 ms  4.849 ms
 4  host-252-234.nynycolo.pavlovmedia.net (66.253.252.234)  28.865 ms  28.826 ms  30.505 ms
 5  v3503.edge2.nylga1.pavlovmedia.net (173.230.125.67)  27.559 ms  27.885 ms  27.840 ms
 6  de-cix-new-york.as13335.net (206.82.104.31)  38.441 ms  45.069 ms  45.213 ms
 7  199.27.132.36 (199.27.132.36)  23.380 ms  20.513 ms  20.283 ms
 8  one.one.one.one (1.1.1.1)  19.216 ms  19.175 ms  19.131 ms
root@VM:/home/seed/Labs1/volumes#
```

Task 1.4: Sniffing and-then Spoofing

The code comes from our lecture. I comment on each line and set the Host A that we want Sniffing and-then Spoofing.

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

def spoof_pkt(pkt):
    #Inside the spoof_pkt() function, it checks if the packet is an ICMP packet with a type of 8 (ICMP Echo Request)
    #using the ICMP in pkt and pkt[ICMP].type==8 condition.
    if ICMP in pkt and pkt[ICMP].type==8:
        print("Original Pakcet.......")
        print("Source IP:",pkt[IP].src)# print the resource IP,which send the packet
        print("Destination IP:",pkt[IP].dst)# print the desination IP, which the captured packet send to.

        # creates a new IP packet using the IP() function
        # 1. set the source IP address to the destination IP address of the original packet
        # 2. set the destination IP address to the source IP address of the original packet
        # 3.the Internet Header Length (IHL) to the value of the IHL field of the original packet
        # 4. the Time to Live (TTL) to 90.
        ip=IP(src=pkt[IP].dst,dst=pkt[IP].src,ihl=pkt[IP].ihl,ttl=90)
        #creates a new ICMP packet using the ICMP() function
        # 1.sets the type to 0 (ICMP Echo Reply)
        # 2.the ID and Sequence Number to the values of the ID and Sequence Number fields of the original packet.

        icmp= ICMP(type=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
        #create new data packet using the Raw() function and sets the load to the data of the original packet.
        data=pkt[Raw].load
        # concatenates the IP, ICMP and data packets to create a new packet.
        newpkt=ip/icmp/data
        print("Spoofed Packet.....")
        print("Source IP:",newpkt[IP].src)
        print("Destination IP:",newpkt[IP].dst)
        # send the packet
        send(newpkt,verbose=0)
                                                                                                    1,24
```

```
pkt=sniff(iface='br-68686c4fbec5',filter='icmp and src host 10.9.0.5',prn=spoof_pkt)
```

1.4.1 ping 1.2.3.4

Attacker:

```
root@VM:/home/seed/Labs1/volumes# python3 task1.4.py
Original Pakcet.......
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 Pakcet.......
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
```

Host A:

```
root@6afb21ddf544:/# ping 1.2.3.4 -c 2
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=90 time=58.9 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=90 time=12.5 ms

--- 1.2.3.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 12.548/35.747/58.947/23.199 ms
root@6afb21ddf544:/#
```

Explain:

For a non-local IP address, the OS will check the routing table and gateway to send a packet. Our OS-kernel would choose **default gateway 10.9.0.1** send our packet. So, the packet can be sent then captured.

```
root@6afb21ddf544:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
```

Then the ARP cache will prepare send the packet , as a result, the Sniffing and-then Spoofing will capture the socket.

```
root@6afb21ddf544:/# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
root@6afb21ddf544:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.99                        (incomplete)                              eth0
10.9.0.1                 ether   02:42:77:a3:9e:32   C                     eth0
root@6afb21ddf544:/#
```

1.4.2 ping 10.9.0.99

Attacker:

```
^Croot@VM:/home/seed/Labs1/volumes# python3 task1.4.py
```

Host A:

```
root@6afb21ddf544:/# ping 10.9.0.99 -c 5
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

--- 10.9.0.99 ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4095ms
pipe 4
root@6afb21ddf544:/#
```

Explain:

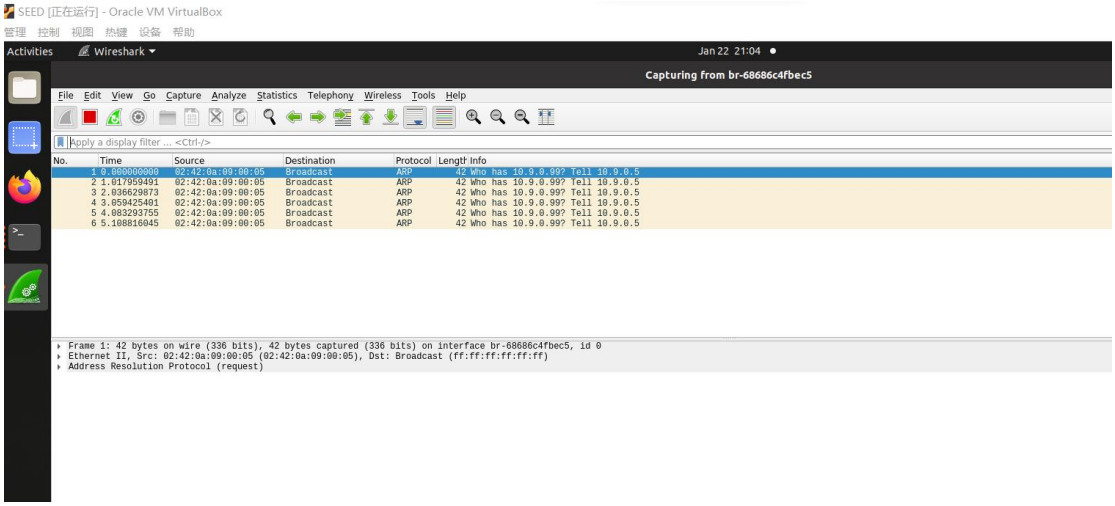
For device in LAN, (in our example IP address between 10.9.0.0-10.9.0.255), our host A is broadcast to find MAC address of 10.9.0.99 rather than send the ICMP packet.

Without the MIC address, in the ARP cache , we can find the state is incomplete,so not pocket will send .

It is clear observed by the wireshark, we can clear see it looking for 10.0.0.99 but no one response.

So, the **ICMP will not send when ping.**

```
root@a9e263899a6a:/# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
10.9.0.99                        (incomplete)                              eth0
10.9.0.1                 ether   02:42:60:e5:87:a9   C                     eth0
```



1.4.3 ping 8.8.8.8

In this case, program works as our design. Packets from host A are sniffing and then spoofing.

Attacker:

```
^Croot@VM:/home/seed/Labs1/volumes# python3 task1.4.py
Original Pakcet.......
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 Pakcet.......
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
```

Host A:

```
root@6afb21ddf544:/# ping 8.8.8.8 -c 2
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=57 time=18.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=90 time=60.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=90 time=13.4 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, +1 duplicates, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 13.374/30.625/59.995/20.872 ms
root@6afb21ddf544:/#
```