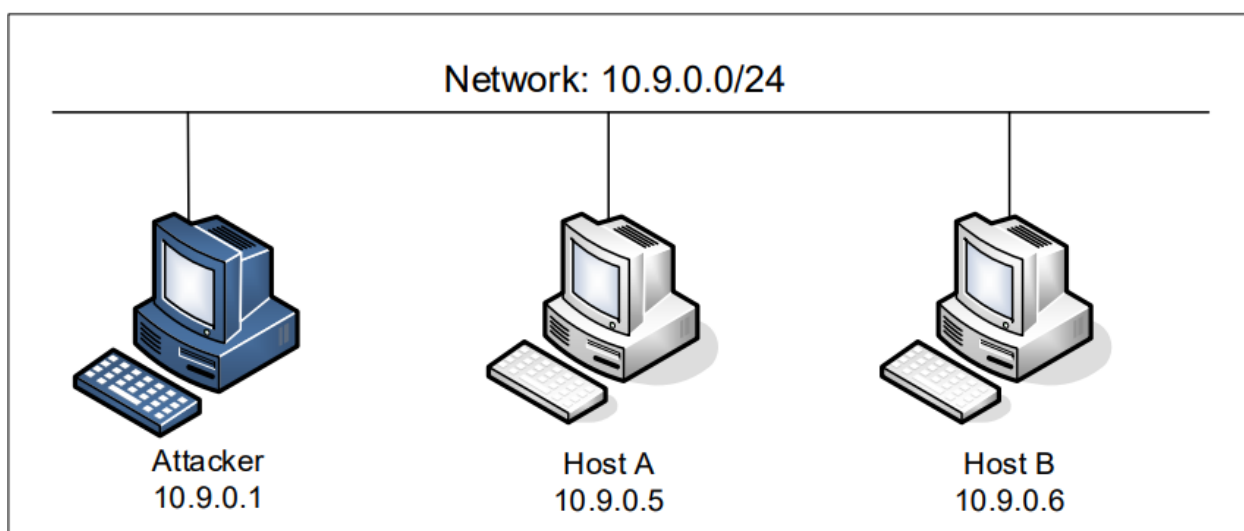




# Packet Sniffing and Spoofing

## Mô hình



Mô hình bài Lab

Thực hiện kiểm tra các host đang chạy trên container

```
[12/11/22] seed@VM:~/.../Labsetup$ dockps
81211927e4a2  hostA-10.9.0.5
2cbc3d2a1884  seed-attacker
235721c1d0f4  hostB-10.9.0.6
```

## Thực hành

### Task 1.1: Snifing Packet

Thực hiện kiểm tra network interface của *Attacker* (10.9.0.1) ta được kết quả

```
[12/11/22]seed@VM:~/.../Sniffing-Spoofing$ ifconfig
br-c1812fd1ba36: 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:cff:fe69:a461 prefixlen 64 scopeid 0x20<link>
    ether 02:42:0c:69:a4:61 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 70 bytes 9191 (9.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Network interface của Attacker là **br-c1812fd1ba36**

Lúc này ta sẽ có được đoạn code sau

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-c1812fd1ba36', filter='icmp', prn=print_pkt)
```

## Task 1.1A

Trong đoạn code trên, ta sẽ thấy rằng cứ mỗi packet bắt được, hàm `print_pkt()` sẽ được gọi và in ra các thông tin trong mỗi packet. Thực hiện chạy chương trình dưới quyền **root** để bắt các gói tin, ta thấy rằng chương trình đã bắt thành công và in ra các thông tin (*Ở đây thực hiện ping từ Host A đến Attacker*).

```
[12/12/22]seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1_1.py
#### Ethernet ####
    dst      = 02:42:0c:69:a4:61
    src      = 02:42:0a:09:00:05
    type     = IPv4
#### IP ####
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 50892
    flags    = DF
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x5fc5
    src      = 10.9.0.5
```

```

    dst      = 10.9.0.1
    \options  \
####[ ICMP ]###
    type     = echo-request
    code     = 0
    chksum   = 0x26fd
    id       = 0x25
    seq      = 0x1
####[ Raw ]###
    load     = '\x18\xb7\x96c\x00\x00\x00\x00U\xef\r\x00\x00\x00\x00\x00\x10\x11\x
12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$$%&\'()*+,-./01234567'

####[ Ethernet ]###
    dst      = 02:42:0a:09:00:05
    src      = 02:42:0c:69:a4:61
    type     = IPv4
####[ IP ]###
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 19876
    flags    =
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x18ee
    src      = 10.9.0.1
    dst      = 10.9.0.5
    \options  \
####[ ICMP ]###
    type     = echo-reply
    code     = 0
    chksum   = 0x2efd
    id       = 0x25
    seq      = 0x1
####[ Raw ]###
    load     = '\x18\xb7\x96c\x00\x00\x00\x00U\xef\r\x00\x00\x00\x00\x00\x10\x11\x
12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$$%&\'()*+,-./01234567'

```


Thực hiện chạy lại chương trình với quyền user thông thường, không phải quyền root, ta sẽ gặp lỗi khi chạy chương trình

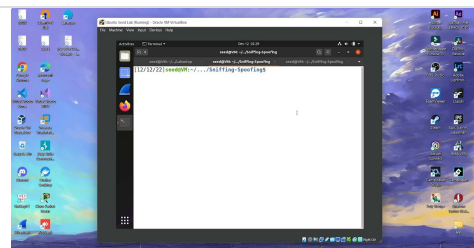
```
[12/12/22] seed@VM:~/.../Sniffing-Spoofing$ python3 Task1_1.py
Traceback (most recent call last):
  File "Task1_1.py", line 7, in <module>
    pkt = sniff(iface='br-c1812fd1ba36', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line
1036, in sniff
    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", lin
e 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.h
tons(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

Operation not permitted (yêu cầu cần thực hiện dưới quyền root)
```

## Video minh họa

Task1.1A.mp4

 <https://drive.google.com/file/d/1JTnvJbl4e-kbS8BSXnaP3m6exQjabqq-/view?usp=sharing>



## Task 1.1B

Thực hiện lọc để chọn ra các loại packet mong muốn sử dụng BPF (Berkeley Packet Filter) syntax

### 1. Chỉ bắt các gói tin ICMP

Thực hiện tương tự như Task 1.1A, ta thấy rằng gói tin khi ping tới mặc định chính là gói tin ICMP

```

root@81211927e4a2:/# ping -c 1 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.088 ms

--- 10.9.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.088/0.088/0.088/0.000 ms

```

## 2. Bắt các gói tin TCP từ một địa chỉ IP nhất định với port = 23

Ta có đoạn code sau:

```

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-c1812fd1ba36', filter='src host 10.9.0.5 and tcp dst port 23', prn=print_pkt)

```

Ở đây ta thực hiện bắt các gói tin có địa chỉ IP nguồn là 10.9.0.5 và có port đích là 23, ở đây ta thực hiện telnet từ *host A* đến *Attacker* (Do port mặc định của telnet là 23). Lúc này ta sẽ được gói tin với kết quả như sau

```

[12/12/22]seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1-1B.py
####[ Ethernet ]####
  dst      = 02:42:0c:69:a4:61
  src      = 02:42:0a:09:00:05
  type     = IPv4
####[ IP ]####
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 41331
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x8521
  src      = 10.9.0.5
  dst      = 10.9.0.1

```


```

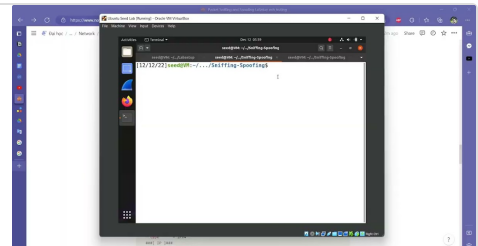
\options \
###[ TCP ]###
sport    = 60082
dport    = telnet
seq      = 576648103
ack      = 0
dataofs  = 10
reserved = 0
flags    = S
window   = 64240
chksum   = 0x1446
urgptr   = 0
options  = [('MSS', 1460), ('SACKOK', b''), ('Timestamp', (2977524954, 0)), ('NOP', None), ('WScale', 7)]

```

## Video minh họa

Task1.1B - Telnet.mp4

 <https://drive.google.com/file/d/1nJski4vjrTYW-dl2XWsQ0HPQytapFnMj/view?usp=sharing>



### 3. Bắt các packet có src hoặc dst là một subnet cụ thể

Ở đây thực hiện ping đến host chính, đầu tiên cần kiểm tra địa chỉ IP của máy chính, ta có kết quả

Wireless LAN adapter Wi-Fi:

```

Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::9088:e1fa:2e2c:bdc1%21
IPv4 Address. . . . . : 192.168.50.26
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.50.1

```

Địa chỉ IP: 192.168.50.26

Lúc này ta có đoạn code sau, chọn subnet là 192.168.50.0 :

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-c1812fd1ba36', filter='net 192.168.50.0/24', prn=print_pkt)
```

Thực hiện ping từ máy A đến host chính (do tất cả đều đi qua network interface của Attacker nên ta có thể thực hiện ping từ chính máy khác). Lúc này ta có kết quả

```
[12/12/22]seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1-1B.py
####[ Ethernet ]####
    dst      = 02:42:0c:69:a4:61
    src      = 02:42:0a:09:00:05
    type     = IPv4
####[ IP ]####
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 41316
    flags    = DF
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x9c74
    src      = 10.9.0.5
    dst      = 192.168.50.26
    \options \
####[ ICMP ]####
    type     = echo-request
    code     = 0
    chksum   = 0x1ac7
    id       = 0x2e
    seq      = 0x1
####[ Raw ]####
    load     = '\xa0\xe9\x96c\x00\x00\x00\x00\xe6\xe9\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      = 02:42:0c:69:a4:61
    type     = IPv4
####[ IP ]####
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
```


```

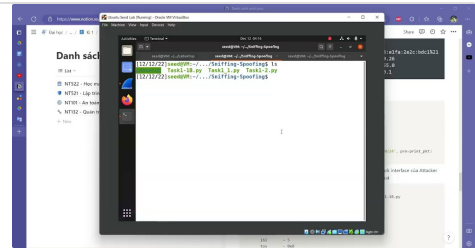
id      = 65387
flags   =
frag    = 0
ttl     = 126
proto   = icmp
chksum  = 0x406d
src     = 192.168.50.26
dst     = 10.9.0.5
\options \
####[ ICMP ]####
    type      = echo-reply
    code      = 0
    chksum    = 0x22c7
    id        = 0x2e
    seq       = 0x1
####[ Raw ]####
    load      = '\xa0\xe9\x96c\x00\x00\x00\x00\xe6\xe9\x00\x00\x00\x00\x00\x10
\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'

```

## Video minh họa

Task1.1B - Net.mp4

 [https://drive.google.com/file/d/1EVuIDhzHeRd0GuJ8W6a\\_gdsNWMbHtSrT/view?usp=sharing](https://drive.google.com/file/d/1EVuIDhzHeRd0GuJ8W6a_gdsNWMbHtSrT/view?usp=sharing)



## Task 1.2: Spoofing ICMP Packet

Ta có đoạn code sau:

```

#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.src = '10.9.0.1'
a.dst = '10.0.0.3'
b = ICMP()
p = a/b
send(p)

```

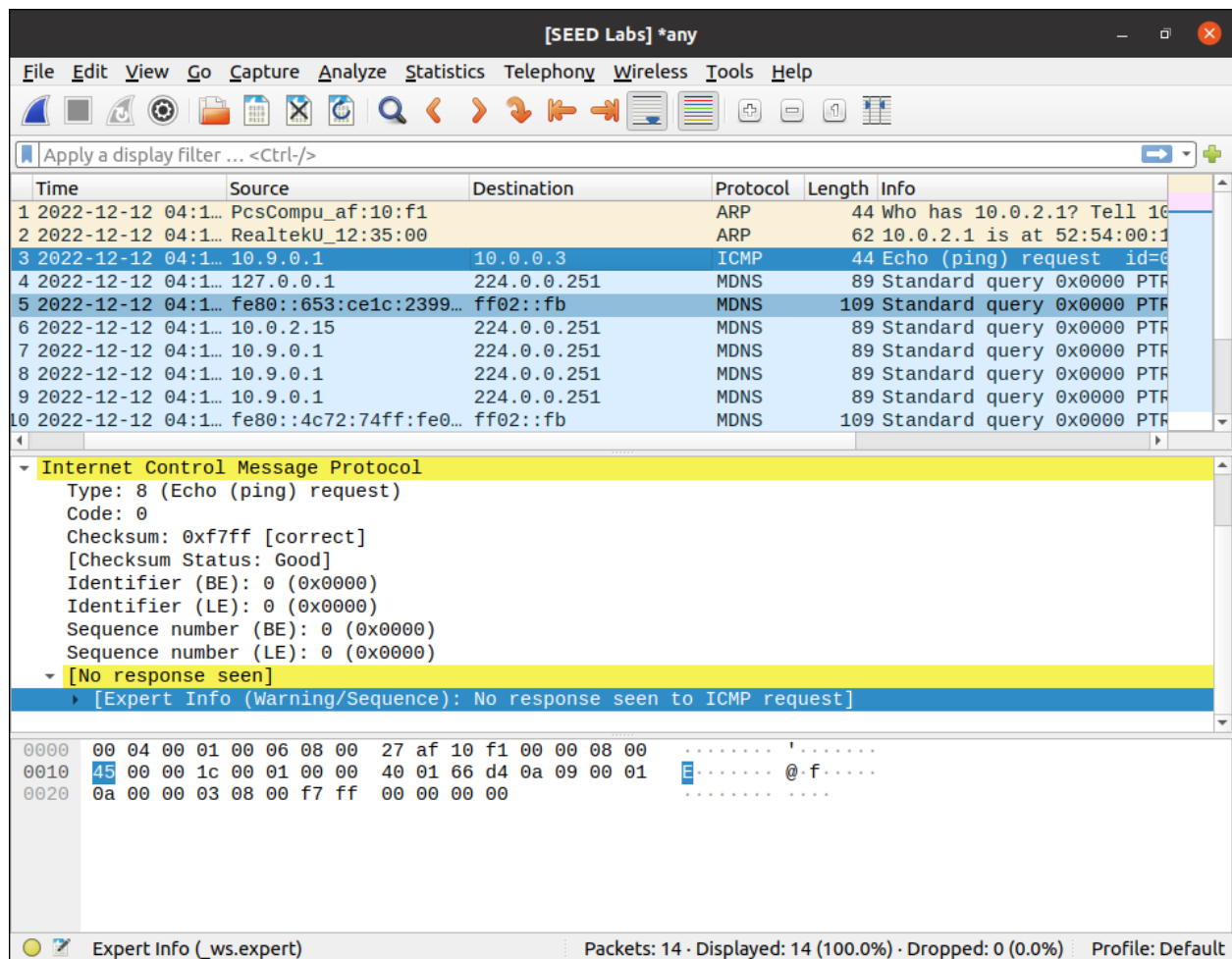
Trong đoạn code trên đầu tiên ta tạo một IP object từ IP class, tiếp theo ta gán cho object 2 địa chỉ IP là địa chỉ nguồn và địa chỉ đích. Tiếp theo ta tạo một ICMP object và



gắn 2 object a và b để tạo thành một packet hoàn chỉnh, thực thi chương trình ta được kết quả

```
[12/12/22] seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1-2.py  
.  
Sent 1 packets.
```

Thực hiện kiểm tra với Wireshark ta thấy rằng gói tin được gửi đi nhưng không có gói tin phản hồi



## Video minh họa

Task1.2.mp4

 <https://drive.google.com/file/d/1izGbWH1VsXamXCvgPFKxqtBCwkpY1WZV/view?usp=sharing>

## Task 1.3: Traceroute

Thực hiện kiểm tra đường đi từ máy chính đến một host bất kì. Ta có đoạn code như sau

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

argumentList = sys.argv

MAX_TTL = 255
dstHostName = argumentList[1]
dstIP = socket.gethostbyname(dstHostName) # return ip of a domain name

continuousLost = 0
MAX_LOST = 6

print("dst IP:", dstIP)

a = IP()
a.dst = dstIP
a.ttl = 1

b = ICMP()

while ip.ttl <= MAX_TTL:
    reply = sr1(a/b, verbose=0, timeout=3)
    print(str(a.ttl), end="\t")

    if reply == None:
        print("---PACKET LOST---")
        continuousLost += 1
        if continuousLost >= MAX_LOST:
            print("Unable to reach", dstIP)
            break
    else:
        print(reply.src)
        continuousLost = 0
        if reply.src == dstIP:
            break
    a.ttl += 1
```

Ở đoạn code trên thực hiện với `MAX_TTL = 255` trong đó `MAX_LOST = 6`, nếu vượt qua số lần mất gói tin đó, chương trình sẽ dừng thực hiện route. Chạy chương trình thực hiện ping đến 8.8.8.8, ta được kết quả

```
[12/12/22]seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1-4.py 8.8.8
.8
dst IP: 8.8.8.8
1      10.0.2.1
2      192.168.50.1
3      192.168.12.11
4      192.168.1.1
5      113.172.192.1
6      172.17.5.5
7      113.171.17.205
8      113.171.7.209
9      113.171.50.222
10     113.171.36.53
11     72.14.209.174
12     72.14.235.235
13     142.251.241.3
14     8.8.8.8
```

```
[12/12/22]seed@VM:~/.../Sniffing-Spoofing$ sudo python3 Task1-4.py 1.2.3
.4
dst IP: 1.2.3.4
1      10.0.2.1
2      192.168.50.1
3      192.168.12.11
4      192.168.1.1
5      113.172.192.1
6      172.17.5.1
7      ---PACKET LOST---
8      ---PACKET LOST---
9      ---PACKET LOST---
10     ---PACKET LOST---
11     ---PACKET LOST---
12     ---PACKET LOST---
Unable to reach 1.2.3.4
```

## Task 1.4: Sniffing and then Spoofing

Kết hợp giữa Sniffing và Spoofing đã được thực hiện ở trên, ở đây ta sẽ sử dụng 1 máy ảo VM để ping đến một host bất kì, ở đây một máy ảo khác cùng card mạng sẽ thực hiện bắt gói tin đó và gửi lại cho máy chủ VM, lúc này bất kể host đó có tồn tại hay không thì sẽ đều nhận được gói tin phản hồi

Ta có đoạn code như sau

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

```
def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        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
        send(newpkt, verbose=0)

pkt = sniff(iface='br-c1812fd1ba36', filter='icmp', prn=spoof_pkt)
```

Đoạn code thực hiện sniff một gói tin bất kì từ một thiết bị, tạo lại một gói tin mới với các trường tương tự như gói tin, thay đổi ip.src và ip.dst (đảo ngược lại với nhau). Lúc này ta sẽ được kết quả

```
root@81211927e4a2:/# ping -c 1 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=14.6 ms
```

```
--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.643/14.643/14.643/0.000 ms
```

Ở đây khi thực hiện thực hiện ping từ host A đến 1.2.3.4 ta nhận được gói tin phản hồi

```
root@81211927e4a2:/# ping -c 1 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
```

```
--- 10.9.0.99 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

Ở đây ta có thể thấy rằng do mạng 10.9.0.99 không nằm trong mạng local nên khi thực hiện ping đến nó packet sẽ không đi qua default gateway nên vì thế không thể sniff được

```
root@81211927e4a2:/# ping -c 2 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=22.6 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=40.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=17.4 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, +1 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 17.387/26.736/40.196/9.754 ms
```


Ta thấy rằng ở đây do khi thực hiện ping đến 8.8.8.8 do host đó có thực sự tồn tại nên khi nhận các gói tin trả về sẽ có sự trùng lặp (1 gói tin đến từ host chính và 1 gói tin bị spoof)

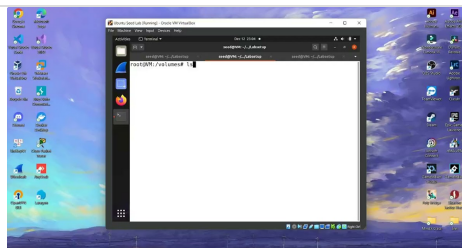
```
root@81211927e4a2:/# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
root@81211927e4a2:/# ip route get 8.8.8.8
8.8.8.8 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
root@81211927e4a2:/# ip route get 10.9.0.99
10.9.0.99 dev eth0 src 10.9.0.5 uid 0
    cache
root@81211927e4a2:/# █
```

**Lưu ý:** Như ở đây ta có thể thấy rằng khi thực hiện kiểm tra route giữa 3 địa chỉ ip trên thì 2 địa chỉ đầu tiên là 1.2.3.4 và 8.8.8.8 sẽ đi qua default gateway là 10.9.0.1 (do đây là địa chỉ IP public) để đi qua ngoài mạng. Đối với IP 10.9.0.99 do đây là địa chỉ local trong cùng mạng con nên nó sẽ được kết nối thẳng với nhau mà không thông qua default gateway

**Video minh họa**

Task1.4.mp4

 [https://drive.google.com/file/d/1gqBg9RYatX1gIFSQuh6GeQVzejzxmY\\_e/view?usp=sharing](https://drive.google.com/file/d/1gqBg9RYatX1gIFSQuh6GeQVzejzxmY_e/view?usp=sharing)



## Task 2.1: Writing Packet Sniffing Program

Ta có đoạn code sau:

### Task 2.1A: Understanding How a Sniffer Works

Thực hiện bắt gói tin bất kì và in ra địa chỉ nguồn và địa chỉ đích của nó, ta có đoạn code sau

```
#include <stdlib.h>
#include <stdio.h>
#include <pcap.h>

/* Ethernet header */
struct ethheader
{
    u_char ether_dhost[6]; /* destination host address */
    u_char ether_shost[6]; /* source host address */
    u_short ether_type;    /* protocol type (IP, ARP, RARP, etc) */
};

/* IP Header */
struct ipheader
{
    unsigned char iph_ihl : 4,          // IP header length
        iph_ver : 4;                    // IP version
    unsigned char iph_tos;               // Type of service
    unsigned short int iph_len;          // IP Packet length (data + header)
    unsigned short int iph_ident;        // Identification
    unsigned short int iph_flag : 3,    // Fragmentation flags
        iph_offset : 13;                // Flags offset
    unsigned char iph_ttl;               // Time to Live
    unsigned char iph_protocol;          // Protocol type
    unsigned short int iph_checksum;     // IP datagram checksum
    struct in_addr iph_sourceip;         // Source IP address
    struct in_addr iph_destip;           // Destination IP address
};

void got_packet(u_char *args, const struct pcap_pkthdr *header,
               const u_char *packet)
{
```

```

    struct ethheader *eth = (struct ethheader *)packet;

    if (ntohs(eth->ether_type) == 0x0800)
    { // 0x0800 is IPv4 type
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));

        printf("Source: %s  ", inet_ntoa(ip->iph_sourceip));
        printf("Destination: %s\n", inet_ntoa(ip->iph_destip));
    }
}

int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp";
    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with name br-c1812fd1ba36
    handle = pcap_open_live("br-c1812fd1ba36", BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    if (pcap_setfilter(handle, &fp) != 0)
    {
        pcap_perror(handle, "Error:");
        exit(EXIT_FAILURE);
    }

    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);

    pcap_close(handle); // Close the handle
    return 0;
}

```

Thực hiện chạy chương trình trên ta được kết quả:

```

root@81211927e4a2:/# ping -c 4 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=113 time=43.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=43.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=40.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=41.1 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 40.403/42.087/43.501/1.341 ms
root@81211927e4a2:/# █

```


```

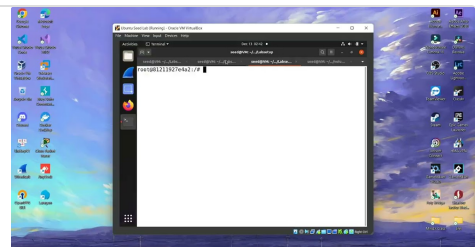
root@VM:/volumes# ls
Task1-4.py Task2-1.cpp sniff
root@VM:/volumes# ./sniff
Source: 10.9.0.5 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.9.0.5
Source: 10.9.0.5 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.9.0.5
Source: 10.9.0.5 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.9.0.5
Source: 10.9.0.5 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.9.0.5
█

```

### Video minh họa:

Task2.1.mp4

 <https://drive.google.com/file/d/15UX9Aoip9hFVHN8bMA4ePCJOFpIAqvK3/view?usp=sharing>





1. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs ?

Dựa vào chương trình trên ta có thể thấy rằng đầu tiên ta thực hiện gọi các thư viện cần thiết cho chương trình, tiếp theo đó xác định network interface mà ở đó chương trình của ta sẽ thực hiện sniff. Trong ví dụ trên là interface `br-c1812fd1ba36` .

Tiếp theo ta định nghĩa loại gói tin mà ta muốn nhận thông qua biến `filter_exp` và thực hiện biên dịch nó (cần thực hiện biên dịch về dạng mà pcap có thể đọc được, ở đây là dưới dạng mã giả BPF)

Cuối cùng chạy chương trình trong vòng lặp, đợi đến khi bắt được gói tin nào đó và gọi đến một callback function `got_packet()`

2. Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Cần thực hiện chạy dưới quyền quản trị viên khi thực thi hàm `pcap_open_live()` cần truy cập vào adapter để chạy dưới chế độ *promiscuous*

3. Turn on and turn off the promiscuous mode in your sniffer program. The value 1 of the third parameter in pcap open live() turns on the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when this mode is on and off ?

Khi tắt chế độ promiscuous, chương trình sẽ chỉ bắt các gói tin, dữ liệu được gửi đến/từ hoặc thông qua host đó mới bị sniff

Ngược lại, khi bật chế độ promiscuous chương trình sẽ bắt tất cả gói tin cũng như luồng dữ liệu trên đường dây cũng như những gói tin mà nó nhìn thấy

## Task 2.1B: Writing Filters

Tương tự như đoạn code trên nhưng ta thay đổi ở hàm `got_packet()` và biến `char filter_exp[] = "ip proto icmp";`

**Yêu cầu:** Bắt các gói tin ICMP đến từ 2 host nhất định, ta có đoạn code sau:

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet){
    struct ethheader *eth = (struct ethheader *)packet;
```

```

if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *) (packet + sizeof(struct ethheader));

    printf("Source: %s  ", inet_ntoa(ip->iph_sourceip));
    printf("Destination: %s", inet_ntoa(ip->iph_destip));

    /* determine protocol */
    switch(ip->iph_protocol) {
        case IPPROTO_ICMP:
            printf("  Protocol: ICMP\n");
            return;
        default:
            printf("  Protocol: others\n");
            return;
    }
}
}

```

Thực hiện chương trình ta có kết quả

```

root@81211927e4a2:/# ping -c 4 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=113 time=40.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=40.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=41.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=42.6 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 40.444/41.349/42.628/0.843 ms
root@81211927e4a2:/# telnet 8.8.8.8
Trying 8.8.8.8...
^C
root@81211927e4a2:/# █

```


```

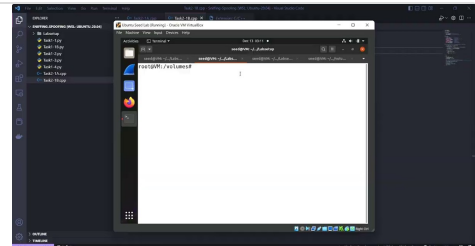
root@VM:/volumes# ls
Task1-4.py Task2-1.cpp Task2-1B.cpp sniff
root@VM:/volumes# ./sniff
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: ICMP
Source: 8.8.8.8 Destination: 10.9.0.5 Protocol: ICMP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: ICMP
Source: 8.8.8.8 Destination: 10.9.0.5 Protocol: ICMP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: ICMP
Source: 8.8.8.8 Destination: 10.9.0.5 Protocol: ICMP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: ICMP
Source: 8.8.8.8 Destination: 10.9.0.5 Protocol: ICMP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: others
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: others
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: others

```

### Video minh họa:

Task2.1B.mp4

 <https://drive.google.com/file/d/1hcsBo-vBnaUW4JuxQbxM37WNwPTeVbXy/view?usp=sharing>



**Yêu cầu:** Bắt các gói tin đến TCP với port đích trong khoảng từ 10 đến 100, tương tự bài trên ta thay đổi biến `filter_exp` và hàm `got_packet()`, ta có đoạn code sau:

```
char filter_exp[] = "proto TCP and dst portrange 10-100";
```

```

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    struct ethheader *eth = (struct ethheader *)packet;

    if (ntohs(eth->ether_type) == 0x0800)
    { // 0x0800 is IP type
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));

        printf("Source: %s  ", inet_ntoa(ip->iph_sourceip));
        printf("Destination: %s", inet_ntoa(ip->iph_destip));
        /* determine protocol */
        switch (ip->iph_protocol)
        {
            case IPPROTO_TCP:
                printf("    Protocol: TCP\n");

```

```

        return;
    default:
        printf("    Protocol: others\n");
        return;
    }
}
}

```

Để bắt được các gói tin có port từ 10-100, ở đây ta sử dụng giao thức telnet chạy ở port 23, chạy chương trình trên ta được kết quả

```

root@81211927e4a2:/# telnet 8.8.8.8
Trying 8.8.8.8...
^C
root@81211927e4a2:/# ping -c 2 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=113 time=40.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=46.9 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 40.573/43.741/46.909/3.168 ms
root@81211927e4a2:/# █

```


```

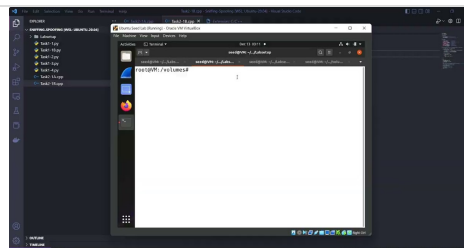
root@VM:/volumes# ls
Task1-4.py Task2-1.cpp Task2-1B.cpp Task2-1B_range.cpp sniff
root@VM:/volumes# ./sniff
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: TCP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: TCP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: TCP
Source: 10.9.0.5 Destination: 8.8.8.8 Protocol: TCP
^C
root@VM:/volumes# ./sniff
█

```

**Video minh họa:**

Task2.1B.mp4

 <https://drive.google.com/file/d/1hcsBo-vBnaUW4JuxQbxM37WNwPTeVbXy/view?usp=sharing>



## Task 2.1C: Sniffing Passwords

Tương tự như đoạn code trên nhưng ta thay đổi ở hàm `got_packet()` và hàm `print_payload()`

```
void print_payload(const u_char *payload, int len)
{
    const u_char *ch;
    ch = payload;
    printf("Payload: \n\t\t");

    for (int i = 0; i < len; i++)
    {
        if (isprint(*ch))
        {
            if (len == 1)
            {
                printf("\t%c", *ch);
            }
            else
            {
                printf("%c", *ch);
            }
        }
        ch++;
    }
    printf("\n\n");
}

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    const struct sniff_tcp *tcp;
    u_char *payload;
    int size_ip;
    int size_tcp;
    int size_payload;

    struct ethheader *eth = (struct ethheader *)packet;

    if (ntohs(eth->ether_type) == 0x0800)
    {
        // 0x0800 is IPv4 type
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
```

```

size_ip = IP_HL(ip) * 4;

/* determine protocol */
switch (ip->iph_protocol)
{
    case IPPROTO_TCP:
        tcp = (struct sniff_tcp *) (packet + SIZE_ETHERNET + size_ip);
        size_tcp = TH_OFF(tcp) * 4;

        payload = (u_char *) (packet + SIZE_ETHERNET + size_ip + size_tcp);
        size_payload = ntohs(ip->iph_len) - (size_ip + size_tcp);

        if (size_payload > 0)
        {
            printf("Source: %s Port: %d\n", inet_ntoa(ip->iph_sourceip), ntohs(tcp->th_sport));
            printf("Destination: %s Port: %d\n", inet_ntoa(ip->iph_destip), ntohs(tcp->th_dport));
            printf("    Protocol: TCP\n");
            print_payload(payload, size_payload);
        }

        return;
    default:
        printf("    Protocol: others\n");
        return;
}
}
}


```

Ở đây chương trình thực hiện việc telnet từ host A đến máy Attacker (ta hoàn toàn có thể sniff được khi telnet từ host A đến host B)

Chương trình thực hiện việc bắt các gói tin tcp telnet đến và in ra dưới dạng chữ

### Video minh họa:

Task2.1C.mp4

 <https://drive.google.com/file/d/115orKMjpPJUGHMHujCcz8GTjOD33JnHu/view?usp=sharing>

