

# BÁO CÁO BÀI TẬP

Môn học: AN TOÀN MẠNG  
PACKET SNIFFING AND SPOOFING LAB

**Nhóm: 05**

## 1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT140.P11.ANTT

STT	Họ và tên	MSSV	Email
1	Hồ Vĩnh Khánh	22520633	22520633@gm.uit.edu.vn
2	Trần Anh Khôi	22520701	22520701@gm.uit.edu.vn
3	Nguyễn Hồ Nhật Khoa	22520677	22520677@gm.uit.edu.vn
4	Diệp Tấn Phát	22521066	22521066@gm.uit.edu.vn

## 2. NỘI DUNG THỰC HIỆN:<sup>1</sup>

STT	Nội dung	Tình trạng
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%
Điểm tự đánh giá		10/10

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

<sup>1</sup> Ghi nội dung công việc, các kịch bản trong bài Thực hành

# BÁO CÁO CHI TIẾT

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

**Task 1.1.A:** In the above program, for each captured packet, the callback function `print_pkt()` will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

- Code python dùng để bắt các gói tin ICMP:

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-5903fba25e43', filter='icmp', prn=print_pkt)
```

- Chạy code với quyền root của trên máy seed-attacker

```
nhnkhoea@nhnKhoea: /mnt/wsl  X  +  v
root@docker-desktop:/# cd volumes/
root@docker-desktop:/volumes# ls
task1.1.py
root@docker-desktop:/volumes# chmod a+x task1.1.py
root@docker-desktop:/volumes# python3 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       = 63298
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x2f4a
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xa46e
  id       = 0x1
  seq      = 0x1
###[ Raw ]###
  load     = '\x17g\x00\x00\x00\xfa\xe9\x06\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

- Thực hiện ping từ host A đến host B

```
nhnkhoea@nhnKhoea: /mnt/wsl x + v
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

E:\UIT\Hoc_ki_5\ATM\BtapBuoi5\Labsetup\Labsetup\volumes>wsl
nhnkhoea@nhnKhoea:/mnt/wsl/docker-desktop-bind-mounts/Ubuntu-22.04/c6d8e124e50abf783efd868ca7270449e5d34f7d72acc2757775344
0c4e1e6e1$ docker exec -it hostA-10.9.0.5 /bin/bash
root@183c91d20ecf:/# ping 10.9.0.6
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=0.512 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.158 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.153 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.159 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.150 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.148 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.120 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.128 ms
^C
--- 10.9.0.6 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8356ms
rtt min/avg/max/mdev = 0.089/0.179/0.512/0.119 ms
root@183c91d20ecf:/#
```

- Ta có thể nhìn thấy một số thông tin của các gói tin này trên máy seed-attacker khi chạy code bắt các gói tin ICMP

```
###[ 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       = 63298
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x2f4a
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xa46e
  id       = 0x1
  seq      = 0x1
###[ Raw ]###
  load     = '|k\x17g\x00\x00\x00\x00\xfa\xe9\x06\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

- Chạy đoạn code python trên nhưng không sử dụng quyền root:

```
nhnkhoe@nhnKhoe: /mnt/wsl x + v
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

E:\UIT\Hoc_ki_5\ATM\BtapBuoi5\Labsetup\Labsetup\volumes>wsl
nhnkhoe@nhnKhoe: /mnt/wsl/docker-desktop-bind-mounts/Ubuntu-22.04/c6d8e124e50abf783efd868ca7270449e5d34f7d72acc27577753440c4e1e6e1$ python3 task1.1.py
Traceback (most recent call last):
  File "/mnt/wsl/docker-desktop-bind-mounts/Ubuntu-22.04/c6d8e124e50abf783efd868ca7270449e5d34f7d72acc27577753440c4e1e6e1/task1.1.py", line 6, in <module>
    pkt = sniff(iface='br-5903fba25e43', filter='icmp', prn=print_pkt)
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 906, in _run
    sniff_socket(L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/lib/python3/dist-packages/scapy/arch/linux.py", line 410, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # noqa: E501
  File "/usr/lib/python3.10/socket.py", line 232, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

- Ta gặp lỗi “Operation not permitted”, vì vậy khi muốn chặn bắt các gói tin chúng ta cần có quyền root để có thể xem lưu lượng và chặn bắt các gói tin liên quan.

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

- Thực hiện chạy code python để bắt gói tin TCP ở Port 23 của host 10.9.0.6

```
nhnkhoe@nhnKhoe: /mnt/wsl x + v
root@docker-desktop: /volumes# python3 task1.1.py

===== packet: 1 =====

###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:0a:09:00:06
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 55304
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x4e87
  src      = 10.9.0.6
  dst      = 10.9.0.5
  \options \
###[ TCP ]###
  sport     = 40386
  dport     = telnet
  seq       = 498449607
  ack       = 0
  dataofs   = 10
  reserved  = 0
  flags     = S

nhnkhoe@nhnKhoe: /mnt/wsl x + v
root@87f88bfd982c: /# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^'.
Ubuntu 20.04.1 LTS
183c91d20ecf login: seed
Password:
```

- Thực hiện chạy code python để bắt các gói tin khi host 10.9.0.5 ping tới một địa chỉ ip bất kì thuộc mạng 128.230.0.0/16

```

nhnkhoea@nhnKhoea: /mnt/wsl x + v
root@docker-desktop:/volumes# python3 task1.1.py

===== packet:1 =====

###[ Ethernet ]###
dst      = 02:42:a0:9e:e5:a6
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 41105
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xf19
src      = 10.9.0.5
dst      = 128.230.0.11
\options
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x7d2e
id       = 0x3
seq      = 0x1
###[ Raw ]###
load     = 'f}\x17g\x00\x00\x00\x00;\x16\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a'

nhnkhoea@nhnKhoea: /mnt/wsl x + v
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

E:\UIT\Hoc_ki_5\ATM\BtapBuoi5\Labsetup\Labsetup\volumes>wsl
nhnkhoea@nhnKhoea: /mnt/wsl/docker-desktop-bind-mounts/Ubuntu-22.04/c6d8e124e50abf783efd868ca7270449e5d34f7d72acc2757775344
0c4e1e6e1$ docker exec -it hostA-10.9.0.5 /bin/bash
root@183c91d20ecf:/# ping 128.230.0.11
PING 128.230.0.11 (128.230.0.11) 56(84) bytes of data.
^C
--- 128.230.0.11 ping statistics ---
25 packets transmitted, 0 received, 100% packet loss, time 24942ms

root@183c91d20ecf:/# |

```

- Đoạn code python được sử dụng trong bài tập này:

```

1  #!/usr/bin/python
2  from scapy.all import *
3
4  def print_pkt(pkt):
5      print_pkt.num_packets += 1
6      print ("\n===== packet:{} =====\n".format(print_pkt.num_packets))
7      pkt.show()
8      print_pkt.num_packets = 0
9  #A. Capture only the ICMP packet
10 pkt = sniff(iface='br-5903fba25e43', filter='icmp', prn=print_pkt)
11
12 #B. Capture any TCP packet that comes from a particular IP and with a destination port number 23
13 pkt = sniff(iface='br-5903fba25e43', filter='tcp && src host 10.9.0.6 && dst port 23', prn=print_pkt)
14
15 #C. 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
16 pkt = sniff(iface='br-5903fba25e43', filter='net 128.230.0.0/16', prn=print_pkt)

```

## Task 1.2: Spoofing ICMP Packets

- Đoạn code bên dưới được dùng để gửi một gói tin ICMP từ một địa chỉ IP giả mạo đến một địa chỉ IP bất kì

```

1  #!/usr/bin/python
2  from scapy.all import *
3
4  a = IP()
5
6  a.src = '1.2.3.4'
7  a.dst = '10.0.2.6'
8  send(a/ICMP())
9  ls(a)

```

- Thực thi đoạn code:

```

nhnkhoea@nhnkhoea: /mnt/wsl
root@docker-desktop:/volumes# python3 task1.2.py
Sent 1 packets.
version : BitField (4 bits) = 4 (4)
ihl : BitField (4 bits) = None (None)
tos : XByteField = 0 (0)
len : ShortField = None (None)
id : ShortField = 1 (1)
flags : FlagsField (3 bits) = <Flag 0 (> (<Flag 0 (>))
frag : BitField (13 bits) = 0 (0)
ttl : ByteField = 64 (64)
proto : ByteEnumField = 0 (0)
chksum : XShortField = None (None)
src : SourceIPField = '1.2.3.4' (None)
dst : DestIPField = '10.0.2.6' (None)
options : PacketListField = [] ([])
root@docker-desktop:/volumes#

```

### Task 1.3: Traceroute

- Nhiệm vụ của task này là sử dụng Scapy để ước tính khoảng cách, theo số lượng định tuyến, giữa VM của ta và máy đích đã chọn

- Trong bài này sẽ chọn VM và host A

- Đoạn code demo ban đầu:

```

a = IP()
a.dst = '1.2.3.4'
a.ttl = 3
b = ICMP()
send(a/b)

```

- Code công cụ được tạo:

```

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

# Task 1.3
def jump(ttl):
    # Create an IP packet
    a = IP()
    a.dst = '8.8.8.8' # Destination address (Google Public DNS)
    a.ttl = int(ttl) # Time to live from function argument

    # Create an ICMP packet
    b = ICMP()

    # Send the packet and receive the response
    response = sr1(a / b)

    # Print the source IP address if a response is received
    if response:
        print("Source:", response.src)
    else:
        print("No response received.")

# Check if a TTL argument is provided
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python3 task1.3.py <ttl>")
        sys.exit(1)

    # Call the jump function with the TTL argument
    jump(sys.argv[1])

```

- Đoạn mã Python trên sử dụng thư viện Scapy để tạo và gửi một gói tin ICMP tới địa chỉ IP 8.8.8.8 (DNS server của Google). Dưới đây là chức năng của từng phần trong mã:

- Mô-đun sys được sử dụng để truy cập các tham số dòng lệnh. Trong trường hợp này, nó được sử dụng để lấy giá trị TTL (Time to Live) từ dòng lệnh.
- Hàm jump nhận một tham số ttl, đại diện cho giá trị Time to Live của gói tin IP. Hàm này sẽ thực hiện tất cả các bước để tạo và gửi gói tin ICMP.
- Tạo một gói tin IP mới (a) với địa chỉ đích là 8.8.4.4 và TTL được lấy từ tham số dòng lệnh (biến ttl). TTL xác định số lượng bước mà gói tin có thể đi qua trước khi bị loại bỏ.
- Tạo một gói ICMP (b). ICMP được sử dụng để gửi thông báo lỗi và kiểm tra kết nối mạng (chẳng hạn như trong lệnh ping).
- Kết hợp gói IP và gói ICMP, sau đó gửi gói tin đi bằng cách sử dụng hàm sr1. Hàm này gửi gói tin và chờ nhận phản hồi đầu tiên từ gói đã gửi.
- In ra địa chỉ IP nguồn của gói tin phản hồi mà đã được nhận. Nếu gói tin đã gửi không nhận được phản hồi, a sẽ là None, và sẽ gây ra lỗi khi cố gắng truy cập thuộc tính src.

**Chức năng tổng quát:** Chức năng tổng quát của mã này là gửi một gói tin ICMP tới một địa chỉ IP cụ thể (trong trường hợp này là Google DNS) với TTL được chỉ định và sau đó in ra địa chỉ IP nguồn của gói tin phản hồi. Đây có thể được xem như là một cách để kiểm tra kết nối mạng đến một địa chỉ cụ thể, tương tự như lệnh ping trong hệ điều hành.

#### **Task 1.4: Sniffing and-then Spoofing**

- Bài này yêu cầu bạn kết hợp các kỹ thuật sniffing (nghe lén gói tin) và spoofing (giả mạo gói tin) để triển khai chương trình.

- Từ container người dùng, ta thực hiện lệnh ping đến một địa chỉ IP X. Điều này sẽ tạo ra gói tin yêu cầu ICMP echo. Nếu IP X còn hoạt động, chương trình ping sẽ nhận được phản hồi và hiển thị kết quả. Chương trình sniff-and-then-spoof chạy trên VM, giám sát mạng LAN bằng cách nghe lén gói tin.

- Bắt đầu với địa chỉ ip 1.2.3.4

```

1#!/usr/bin/env python3
2from scapy.all import *
3#Task 1.4
4
5def spoof_pkt(pkt):
6    # sniff and print out icmp echo request packet
7    if ICMP in pkt and pkt[ICMP].type == 8:
8        print("Original Packet.....")
9        print("Source IP : ", pkt[IP].src)
10       print("Destination IP :", pkt[IP].dst)
11
12       # spoof an icmp echo reply packet
13       # swap srcip and dstip
14       ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
15       icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
16       data = pkt[Raw].load
17       newpkt = ip/icmp/data
18
19       print("Spoofed Packet.....")
20       print("Source IP : ", newpkt[IP].src)
21       print("Destination IP :", newpkt[IP].dst)
22
23       send(newpkt, verbose=0)
24#ping 1.2.3.4 # a non-existing host on the Internet
25
26filter = 'icmp and host 1.2.3.4'
27# print("filter: {}".format(filter))
28#ping 10.9.0.99 # a non-existing host on the LAN
29
30#ping 8.8.8.8 # an existing host on the Internet
31pkt = sniff(iface = 'br-6257e8301eb6', filter=filter, prn=spoof_pkt)

```

```
anhkhloi@anhkhloi-ubuntu-0312:~/Tran Anh Khoi/Labsetup$ sudo docker exec -it hostA
-10.9.0.5 /bin/bash
[sudo] password for ankhkhloi:
root@ff51ebd7018b:/# ping 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=60.1 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.4 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=16.6 ms
```

```
task1.3.py task1.4.py
root@anhkhoi-ubuntu-0312:/volumes# python3 ./task1.4.py
Original Packet.....
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 Packet.....
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 Packet.....
```



## Nhóm 05

- Tiếp theo là địa chỉ IP 10.9.0.99, do địa chỉ này không hoạt động nên không thấy gói tin nào trả về

```
1#!/usr/bin/env python3
2from scapy.all import *
3#Task 1.4
4
5def spoof_pkt(pkt):
6    # sniff and print out icmp echo request packet
7    if ICMP in pkt and pkt[ICMP].type == 8:
8        print("Original Packet.....")
9        print("Source IP : ", pkt[IP].src)
10       print("Destination IP :", pkt[IP].dst)
11
12       # spoof an icmp echo reply packet
13       # swap srcip and dstip
14       ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
15       icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
16       data = pkt[Raw].load
17       newpkt = ip/icmp/data
18
19       print("Spoofed Packet.....")
20       print("Source IP : ", newpkt[IP].src)
21       print("Destination IP :", newpkt[IP].dst)
22
23       send(newpkt, verbose=0)
24#ping 1.2.3.4 # a non-existing host on the Internet
25
26filter = 'icmp and host 10.9.0.99'
27# print("filter: {}".format(filter))
28#ping 10.9.0.99 # a non-existing host on the LAN
29
30#ping 8.8.8.8 # an existing host on the Internet
31pkt = sniff(iface = 'br-6257e8301eb6', filter=filter, prn=spoof_pkt)
```

```
root@ff51ebd7018b:/# 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
ping: sendmsg: No route to host
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
```

```
root@anhkhoi-ubuntu-0312:/volumes# python3 ./task1.4.py
```

- Cuối cùng là địa chỉ IP 8.8.8.8

```
1#!/usr/bin/env python3
2from scapy.all import *
3#Task 1.4
4
5def spoof_pkt(pkt):
6    # sniff and print out icmp echo request packet
7    if ICMP in pkt and pkt[ICMP].type == 8:
8        print("Original Packet.....")
9        print("Source IP : ", pkt[IP].src)
10       print("Destination IP : ", pkt[IP].dst)
11
12       # spoof an icmp echo reply packet
13       # swap srcip and dstip
14       ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
15       icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
16       data = pkt[Raw].load
17       newpkt = ip/icmp/data
18
19       print("Spoofed Packet.....")
20       print("Source IP : ", newpkt[IP].src)
21       print("Destination IP : ", newpkt[IP].dst)
22
23       send(newpkt, verbose=0)
24 #ping 1.2.3.4 # a non-existing host on the Internet
25
26 filter = 'icmp and host 8.8.8.8'
27 # print("filter: {}".format(filter))
28 #ping 10.9.0.99 # a non-existing host on the LAN
29
30 #ping 8.8.8.8 # an existing host on the Internet
31 pkt = sniff(iface = 'br-6257e8301eb6', filter=filter, prn=spoof_pkt)
```

```
root@ff51ebd7018b:/# 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=64 time=38.8 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=54.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=15.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=54.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=22.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=54.2 ms (DUP!)
```

```
root@anhkhoi-ubuntu-0312:/volumes# python3 ./task1.4.py
^Croot@anhkhoi-ubuntu-0312:/volumes# python3 ./task1.4.py
Original Packet.....
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 Packet.....
```

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

## Task 2.1: Writing Packet Sniffing Program

- Sniffer programs can be easily written using the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. At the end of the sequence, packets will be put in buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the pcap library.

## Task 2.1A: Understanding How a Sniffer Works

```
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/if_ether.h>

void packet_handler(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
    printf("Received packet of length: %d\n", header->len);

    // Lấy thông tin Ethernet
    struct ether_header *eth_header = (struct ether_header *)packet;
    printf("Source MAC: ");
    for (int i = 0; i < 6; i++) printf("%02x:", eth_header->ether_shost[i]);
    printf("\nDestination MAC: ");
    for (int i = 0; i < 6; i++) printf("%02x:", eth_header->ether_dhost[i]);
    printf("\n");

    // Kiểm tra nếu gói tin là IPv4
    if (ntohs(eth_header->ether_type) == ETHertype_IP) {
        struct ip *ip_header = (struct ip *)(packet + sizeof(struct ether_header));
        printf("Source IP: %s\n", inet_ntoa(ip_header->ip_src));
        printf("Destination IP: %s\n", inet_ntoa(ip_header->ip_dst));
    }
    printf("\n");
}

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

    // Mở thiết bị mạng để sniffing
    handle = pcap_open_live("ens33", BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        fprintf(stderr, "Could not open device: %s\n", errbuf);
        return 2;
    }

    // Bắt gói tin
    pcap_loop(handle, 10, packet_handler, NULL);

    pcap_close(handle);
    return 0;
}
```

```
nachtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$ sudo ./
Received packet of length: 93
Source MAC: 00:50:56:fc:53:ec:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 3.233.158.26
Destination IP: 192.168.81.132

Received packet of length: 54
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 3.233.158.26

Received packet of length: 93
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 3.233.158.26

Received packet of length: 78
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 3.233.158.26

Received packet of length: 54
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 3.233.158.26

Received packet of length: 60
Source MAC: 00:50:56:fc:53:ec:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 3.233.158.26
Destination IP: 192.168.81.132

Received packet of length: 60
Source MAC: 00:50:56:fc:53:ec:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 3.233.158.26
Destination IP: 192.168.81.132

Received packet of length: 60
Source MAC: 00:50:56:fc:53:ec:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 3.233.158.26
Destination IP: 192.168.81.132

Received packet of length: 78
Source MAC: 00:50:56:fc:53:ec:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 3.233.158.26
Destination IP: 192.168.81.132

Received packet of length: 54
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 3.233.158.26
nachtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$
```

*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.*

Các lệnh thư viện chính:

- `pcap_open_live`: Mở một phiên để bắt gói tin.
- `pcap_compile`: Chuyển bộ lọc thành mã để áp dụng.
- `pcap_setfilter`: Áp dụng bộ lọc lên phiên bắt gói tin.
- `pcap_loop`: Liên tục bắt gói tin và xử lý qua hàm `got_packet`.
- `pcap_close`: Đóng phiên bắt gói tin.

*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?*

- Chương trình cần quyền root vì thao tác bắt gói tin yêu cầu truy cập trực tiếp vào giao diện mạng. Nếu không có quyền root, `pcap_open_live` sẽ gặp lỗi.

*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.*

- Để kiểm tra chế độ promiscuous, thử thay đổi tham số thứ ba trong `pcap_open_live` giữa 1 (bật) và 0 (tắt) và dùng lệnh sau để kiểm tra:  
**`ip -d link show dev eth0`**

## Task 2.1B: Writing Filters.

```
// Thiết lập biểu thức lọc cho các yêu cầu của Task 2.2
struct bpf_program fp;

// Biểu thức lọc cho các gói tin ICMP giữa hai host cụ thể
char filter_exp_icmp[] = "icmp and host 192.168.81.1 and host 192.168.81.132";
if (pcap_compile(handle, &fp, filter_exp_icmp, 0, PCAP_NETMASK_UNKNOWN) == -1) {
    fprintf(stderr, "Could not parse filter %s: %s\n", filter_exp_icmp, pcap_geterr(handle));
    return 2;
}
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Could not install filter %s: %s\n", filter_exp_icmp, pcap_geterr(handle));
    return 2;
}

printf("Bắt gói ICMP giữa hai host 192.168.81.1 và 192.168.81.132\n");
pcap_loop(handle, 5, packet_handler, NULL);

// Thiết lập biểu thức lọc cho các gói TCP có cổng đích trong khoảng từ 10 đến 100
char filter_exp_tcp[] = "tcp and dst portrange 10-100";
if (pcap_compile(handle, &fp, filter_exp_tcp, 0, PCAP_NETMASK_UNKNOWN) == -1) {
    fprintf(stderr, "Could not parse filter %s: %s\n", filter_exp_tcp, pcap_geterr(handle));
    return 2;
}
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Could not install filter %s: %s\n", filter_exp_tcp, pcap_geterr(handle));
    return 2;
}

printf("Bắt gói TCP có cổng đích trong khoảng từ 10 đến 100\n");
pcap_loop(handle, 5, packet_handler, NULL);

pcap_close(handle);
```



```
machtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$ gcc sniffer.c -o sniffer2.2 -lpcap
machtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$ sudo ./sniffer2.2
Bắt gói ICMP giữa hai host 192.168.81.1 và 192.168.81.132
Received packet of length: 74
Source MAC: 00:50:56:c0:00:08:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.1
Destination IP: 192.168.81.132

Received packet of length: 74
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:c0:00:08:
Source IP: 192.168.81.132
Destination IP: 192.168.81.1

Received packet of length: 74
Source MAC: 00:50:56:c0:00:08:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.1
Destination IP: 192.168.81.132

Received packet of length: 74
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:c0:00:08:
Source IP: 192.168.81.132
Destination IP: 192.168.81.1

Received packet of length: 74
Source MAC: 00:50:56:c0:00:08:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.1
Destination IP: 192.168.81.132

Bắt gói TCP có cổng đích trong khoảng từ 10 đến 100
Received packet of length: 74
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 185.125.190.49

Received packet of length: 54
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 185.125.190.49

Received packet of length: 142
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 185.125.190.49

Received packet of length: 54
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:50:56:fc:53:ec:
Source IP: 192.168.81.132
Destination IP: 185.125.190.49
```

## Task 2.1C: Sniffing Passwords.

```
// Thiết lập biểu thức lọc để chỉ bắt gói tin Telnet (TCP port 23)
struct bpf_program fp;
char filter_exp[] = "tcp port 23";
if (pcap_compile(handle, &fp, filter_exp, 0, PCAP_NETMASK_UNKNOWN) == -1) {
    fprintf(stderr, "Could not parse filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return 2;
}
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Could not install filter %s: %s\n", filter_exp, pcap_geterr(handle));
    return 2;
}

printf("Listening for Telnet packets...\n");
pcap_loop(handle, -1, packet_handler, NULL);

pcap_close(handle);
return 0;
```

## - Tạo server và kết nối:

```
nachtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$
sudo python3 telnet_server.py
Server listening on 0.0.0.0:23
Connection from ('192.168.81.128', 57922)
Username entered: admin
Password entered: password
Connection with ('192.168.81.128', 57922) closed.
```

```
(root@kali)-[~]
# telnet 192.168.81.132 23
Trying 192.168.81.132 ...
Connected to 192.168.81.132.
Escape character is '^]'.
Welcome to the Telnet server!
Enter username: admin
Enter password: password
Login successful! Type 'exit' to quit.
> exit
Goodbye!
Connection closed by foreign host.

(root@kali)-[~]
```



```
import socket

HOST = '0.0.0.0' # Địa chỉ IP lắng nghe
PORT = 23        # Cổng Telnet server

def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # Cho phép sử dụng lại địa chỉ và cổng
    server.bind((HOST, PORT))
    server.listen(1)
    print(f"Server listening on {HOST}:{PORT}")

    while True:
        client_socket, addr = server.accept()
        print(f"Connection from {addr}")
        client_socket.sendall(b"Welcome to the Telnet server!\n")

        # Yêu cầu nhập username
        client_socket.sendall(b"Enter username: ")
        username = read_line(client_socket)
        print(f"Username entered: {username}")

        # Yêu cầu nhập password sau khi có username
        client_socket.sendall(b"Enter password: ")
        password = read_line(client_socket)
        print(f"Password entered: {password}")

        # Xác nhận đăng nhập thành công sau khi nhận cả username và password
        client_socket.sendall(b>Login successful! Type 'exit' to quit.\n")

        # Vòng lặp chính để nhận lệnh từ người dùng
        while True:
            client_socket.sendall(b"> ") # Hiển thị prompt
            command = read_line(client_socket)
            if command.lower() == "exit":
                client_socket.sendall(b"Goodbye!\n")
                break
            else:
                client_socket.sendall(b"Command received: " + command.encode() + b"\n")

        client_socket.close()
        print(f"Connection with {addr} closed.")
```

- Chạy code để bắt gói tin và hiển thị nếu bắt được mật khẩu:

```
nachtrangcuc@dtphat-VMware-Virtual-Platform:~/Documents/NT140/Sniffing_Spoofing$ sudo ./sniffer_2.3
Listening for Telnet packets...
Received packet of length: 74
Source MAC: 00:0c:29:70:1a:6c:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.128
Destination IP: 192.168.81.132

Received packet of length: 74
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:0c:29:70:1a:6c:
Source IP: 192.168.81.132
Destination IP: 192.168.81.128
```

```
Received packet of length: 66
Source MAC: 00:0c:29:70:1a:6c:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.128
Destination IP: 192.168.81.132
```

```
Received packet of length: 73
Source MAC: 00:0c:29:70:1a:6c:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.128
Destination IP: 192.168.81.132
Payload:
admin..
```

```
Received packet of length: 66
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:0c:29:70:1a:6c:
Source IP: 192.168.81.132
Destination IP: 192.168.81.128
```

```
Received packet of length: 82
Source MAC: 00:0c:29:87:f9:58:
Destination MAC: 00:0c:29:70:1a:6c:
Source IP: 192.168.81.132
Destination IP: 192.168.81.128
Payload:
Enter password:
```

```
Received packet of length: 66
Source MAC: 00:0c:29:70:1a:6c:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.128
Destination IP: 192.168.81.132
```

```
Received packet of length: 76
Source MAC: 00:0c:29:70:1a:6c:
Destination MAC: 00:0c:29:87:f9:58:
Source IP: 192.168.81.128
Destination IP: 192.168.81.132
Payload:
password..
```

## Task 2.2A: Write a spoofing program.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <netinet/ip.h>
6 #include <netinet/ip_icmp.h>
7 #include <unistd.h>
8
9 // Hàm để tính toán checksum
10 unsigned short checksum(void *b, int len) {
11     unsigned short *buf = b;
12     unsigned int sum = 0;
13     unsigned short result;
14
15     for (sum = 0; len > 1; len -= 2)
16         sum += *buf++;
17     if (len == 1)
18         sum += *(unsigned char *)buf;
19     sum = (sum >> 16) + (sum & 0xFFFF);
20     sum += (sum >> 16);
21     result = ~sum;
22     return result;
23 }
24
25 int main() {
26     int sd;
27     struct sockaddr_in sin;
28     char buffer[1024]; // Bộ đệm cho gói
29
30     // Tạo một socket thô với giao thức IP
31     sd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
32     if(sd < 0) {
33         perror("socket() error");
34         exit(-1);
35     }
36
37     // Thông tin điểm đến
38     sin.sin_family = AF_INET;
39     sin.sin_port = htons(0); // Không sử dụng cho ICMP
40     sin.sin_addr.s_addr = inet_addr("192.168.81.1"); // Thay đổi thành IP mục tiêu
41
42     sin.sin_addr.s_addr = inet_addr("192.168.81.1"); // Thay đổi thành IP mục tiêu
43
44     // Điền vào tiêu đề IP
45     struct iphdr *iph = (struct iphdr *)buffer;
46     iph->ihl = 5; // Kích thước tiêu đề
47     iph->version = 4; // IPv4
48     iph->tos = 0; // Loại dịch vụ
49     iph->tot_len = sizeof(struct iphdr) + sizeof(struct icmphdr); // Tổng chiều dài
50     iph->id = htonl(54321); // ID duy nhất
51     iph->frag_off = 0; // Độ lệch phân mảnh
52     iph->ttl = 64; // Thời gian sống
53     iph->protocol = IPPROTO_ICMP; // Loại giao thức
54     iph->check = 0; // Checksum (ban đầu là 0)
55     iph->saddr = inet_addr("192.168.81.128"); // Địa chỉ IP giả mạo
56     iph->daddr = sin.sin_addr.s_addr;
57
58     // Điền vào tiêu đề ICMP
59     struct icmphdr *icmph = (struct icmphdr *)(buffer + sizeof(struct iphdr));
60     icmph->type = ICMP_ECHO; // Loại ICMP
61     icmph->code = 0; // Mã
62     icmph->checksum = 0; // Checksum (ban đầu là 0)
63     icmph->un.echo.id = htons(1234); // ID
64     icmph->un.echo.sequence = htons(1); // Số thứ tự
65
66     // Tính toán checksum
67     icmph->checksum = checksum((unsigned short *)icmph, sizeof(struct icmphdr));
68     iph->check = checksum((unsigned short *)iph, sizeof(struct iphdr));
69
70     // Gửi gói
71     if(sendto(sd, buffer, iph->tot_len, 0, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
72         perror("sendto() error");
73         exit(-1);
74     }
75
76     printf("Gói đã được gửi\n");
77     close(sd);
78     return 0;
79 }

```

## Task 2.2B: Spoof an ICMP Echo Request.

```
(kali@kali)-[~/Documents/NT140/Sniffing_Spoofing]
$ gcc -o spoof_icmp spoof_icmp.c

(kali@kali)-[~/Documents/NT140/Sniffing_Spoofing]
$ sudo ./spoof_icmp
[sudo] password for kali:
Gói đã được gửi
```

No.	Time	Source	Destination	Protocol	Length	Info
21	1.960543689	192.168.81.128	192.168.81.1	ICMP	62	Unknown ICMP (obsolete or malformed?)

```

    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not Set)
    Total Length: 48
    Identification: 0xfe36 (65078)
    ▶ 010. .... = Flags: 0x2, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
    Header Checksum: 0x18c4 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.81.128
    Destination Address: 192.168.81.1
    ▶ Internet Control Message Protocol
    Type: 69 (Unknown ICMP (obsolete or malformed?))
    Code: 0
    Checksum: 0x1c00 [correct]
    [Checksum Status: Good]
  
```

Internet Control Message Protocol: Protocol      Packets: 141 · Displayed: 1 (0.7%) · Dropped: 0 (0.0%) · Profile: Default

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

- Về lý thuyết, có thể đặt nó thành bất kỳ giá trị nào, nhưng nó phải phù hợp với kích thước gói thực tế. Nếu không, gói có thể bị loại bỏ bởi máy nhận hoặc các thiết bị trung gian.

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

- Có, phải tính toán checksum cho tiêu đề IP để đảm bảo rằng gói được hiểu đúng bởi hệ thống nhận.

*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?*

- Socket thô cho phép xây dựng các gói thủ công, điều này có thể được sử dụng cho các hoạt động độc hại như giả mạo gói. Do đó, quyền root là cần thiết để ngăn người dùng không có quyền gửi các gói được chế tạo. Nếu không có quyền root, việc tạo socket sẽ thất bại và sẽ nhận được thông báo lỗi "Permission denied".

**Task 2.3: Sniff and then Spoof**

- Chương trình "sniff-and-then-spoof" chạy trên máy tấn công, giám sát mạng thông qua việc bắt gói tin. Khi nó phát hiện một gói tin yêu cầu ICMP echo, chương trình sẽ ngay lập tức gửi một phản hồi echo giả mạo. Như vậy, dù máy X có hoạt động hay không, chương trình ping sẽ luôn nhận được phản hồi, cho thấy máy X đang hoạt động

**Code:**

```
void send_raw_ip_packet(struct ipheader *ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;
    // Bước 1: Tạo một raw socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    // Bước 2: Thiết lập các option của socket.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
               &enable, sizeof(enable));
    // Bước 3: Thông tin destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;
    // Bước 4: Gửi gói tin ra ngoài.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

```
void send_echo_reply(struct ipheader *ip)
{
    int ip_header_len = ip->iph_ihl * 4;
    const char buffer[PACKET_LEN];
    // Tạo một bản sao từ gói tin gốc vào bộ đệm (gói tin giả)
    memset((char *)buffer, 0, PACKET_LEN);
    memcpy((char *)buffer, ip, ntohs(ip->iph_len));
    struct ipheader *newip = (struct ipheader *)buffer;
    struct icmpheader *newicmp = (struct icmpheader *) (buffer + ip_header_len);
    // SWAP src và dest trong gói ICMP giả mạo
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 64;
    // Điền tất cả thông tin tiêu đề ICMP cần thiết.
    // ICMP Type: 8 is request, 0 is reply.
    newicmp->icmp_type = 0;
    send_raw_ip_packet(newip);
}
```

```

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(" From: %s\n", inet_ntoa(ip->iph_sourceip));
        printf(" To: %s\n", inet_ntoa(ip->iph_destip));
        /* determine protocol */
        switch (ip->iph_protocol)
        {
            case IPPROTO_TCP:
                printf(" Protocol: TCP\n");
                return;
            case IPPROTO_UDP:
                printf(" Protocol: UDP\n");
                return;
            case IPPROTO_ICMP:
                printf(" Protocol: ICMP\n");
                send_echo_reply(ip);
                return;
            default:
                printf(" Protocol: others\n");
                return;
        }
    }
}

```

```

int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp[icmptype] = 8";
    bpf_u_int32 net;
    // Bước 1: Mở phiên pcap trực tiếp trên NIC với tên eth3
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    // Bước 2: Biên dịch filter_exp thành mã BPF
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
    // Bước 3: Bắt gói tin
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);
    return 0;
}

```

- Máy tấn công được đặt ở chế độ "promiscuous", nghĩa là nó có thể nhận tất cả các gói tin đi qua mạng mà không chỉ gói tin gửi đến nó. Khi chạy chương trình giả mạo (spoofing), thẻ mạng (NIC) của máy attacker sẽ bắt tất cả các gói tin đến.
- Chương trình sẽ xử lý các gói tin này theo cách mà nó thay đổi địa chỉ đích thành địa chỉ nguồn và địa chỉ nguồn thành địa chỉ đích. Sau khi tạo gói tin mới, nó sẽ gửi gói tin ra ngoài. Kết quả là, nạn nhân sẽ nhận được gói tin đó. Như vậy, chúng ta đã giả mạo thành công yêu cầu echo ICMP.

### Screenshots:

#### The Victim

```
seed@VM: ~  
seed@VM: ~/.../Labsetup x seed@VM: ~ x seed@VM: ~/.../volumes x  
[10/28/24]seed@VM:~$ ping -c 3 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=255 time=26.7 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=255 time=27.7 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=255 time=28.4 ms  
  
--- 8.8.8.8 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2006ms  
rtt min/avg/max/mdev = 26.716/27.630/28.448/0.710 ms  
[10/28/24]seed@VM:~$
```

#### The Attacker

```
seed@VM: ~/.../volumes  
seed@VM: ~/.../Labsetup x seed@VM: ~/.../volumes x seed@VM: ~/.../volumes x  
[10/28/24]seed@VM:~/.../volumes$ sudo ./sniffspoof  
From: 10.0.2.15  
To: 8.8.8.8  
Protocol: ICMP  
From: 10.0.2.15  
To: 8.8.8.8  
Protocol: ICMP  
From: 10.0.2.15  
To: 8.8.8.8  
Protocol: ICMP  
█
```



[SEED Labs] task2.3.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2024-10-28 05:0...	10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=
2	2024-10-28 05:0...	8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=
3	2024-10-28 05:0...	10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=
4	2024-10-28 05:0...	8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=
5	2024-10-28 05:0...	10.0.2.15	8.8.8.8	ICMP	98	Echo (ping) request id=
6	2024-10-28 05:0...	8.8.8.8	10.0.2.15	ICMP	98	Echo (ping) reply id=
7	2024-10-28 05:0...	PcsCompu_b8:06:15	52:55:0a:00:02:02	ARP	42	Who has 10.0.2.2? Tell 1
8	2024-10-28 05:0...	52:55:0a:00:02:02	PcsCompu_b8:06:15	ARP	64	10.0.2.2 is at 52:55:0a:
9	2024-10-28 05:0...	10.0.2.15	34.117.121.53	TLSv1.2	93	Application Data
10	2024-10-28 05:0...	34.117.121.53	10.0.2.15	TCP	60	443 → 59354 [ACK] Seq=10
11	2024-10-28 05:0...	34.117.121.53	10.0.2.15	TLSv1.2	93	Application Data
12	2024-10-28 05:0...	10.0.2.15	34.117.121.53	TCP	54	59354 → 443 [ACK] Seq=27

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0

Ethernet II, Src: PcsCompu\_b8:06:15 (08:00:27:b8:06:15), Dst: 52:55:0a:00:02:02 (52:55:0a:00:02:02)

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8

Internet Control Message Protocol

```

0000  52 55 0a 00 02 02 08 00 27 b8 06 15 08 00 45 00  RU.....E.
0010  00 54 97 f4 40 00 40 01 86 96 0a 00 02 0f 08 08  .T..@..
0020  08 08 08 00 f9 54 00 04 00 01 79 53 1f 67 00 00  ....T...yS.g..
0030  00 00 9c 18 0b 00 00 00 00 00 10 11 12 13 14 15  ....
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  ....!"$%&'()*+,-./012345
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37                                           67
    
```

-- Hết --