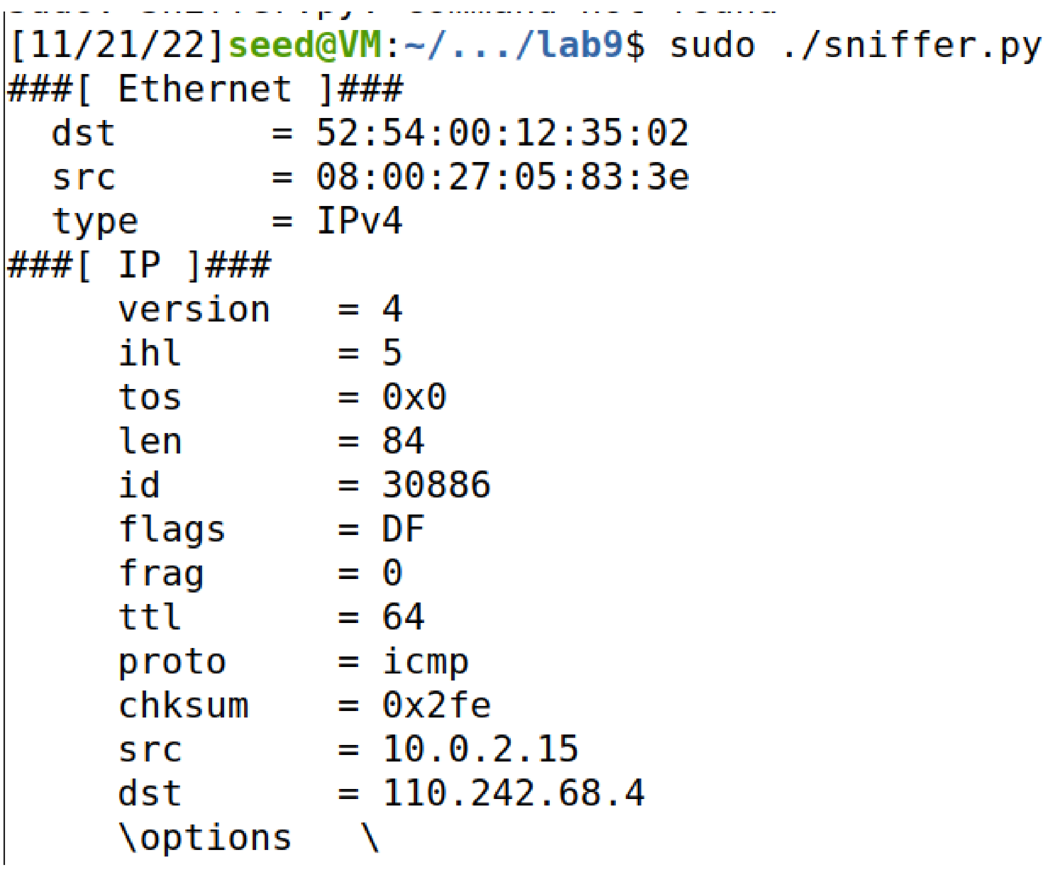
Task1

1.1A sniffer.py

sniffer.py捕获指定网卡的ICMP包并打印信息：ICMP包可由ping任意主机产生

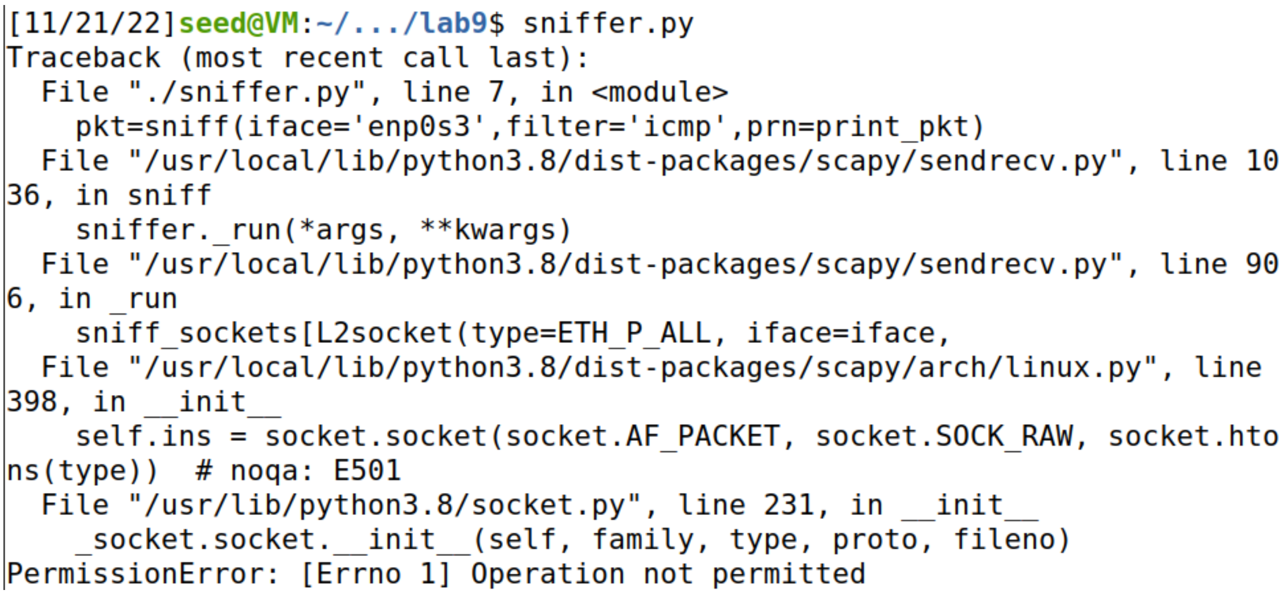
```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()

pkt=sniff(iface='enp0s3',filter='icmp',prn=print_pkt)
```

赋予可执行权限：`chmod a+x sniffer.py`，以root权限执行：



```
[11/21/22]seed@VM:~/.../lab9$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:05:83:3e
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 30886
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2fe
     src       = 10.0.2.15
     dst       = 110.242.68.4
     \options   \
```

以seed用户权限执行：出现权限错误，因为创建原始套接字需要root权限



```
[11/21/22]seed@VM:~/.../lab9$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt=sniff(iface='enp0s3',filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 10
36, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 90
6, 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.hto
ns(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
```

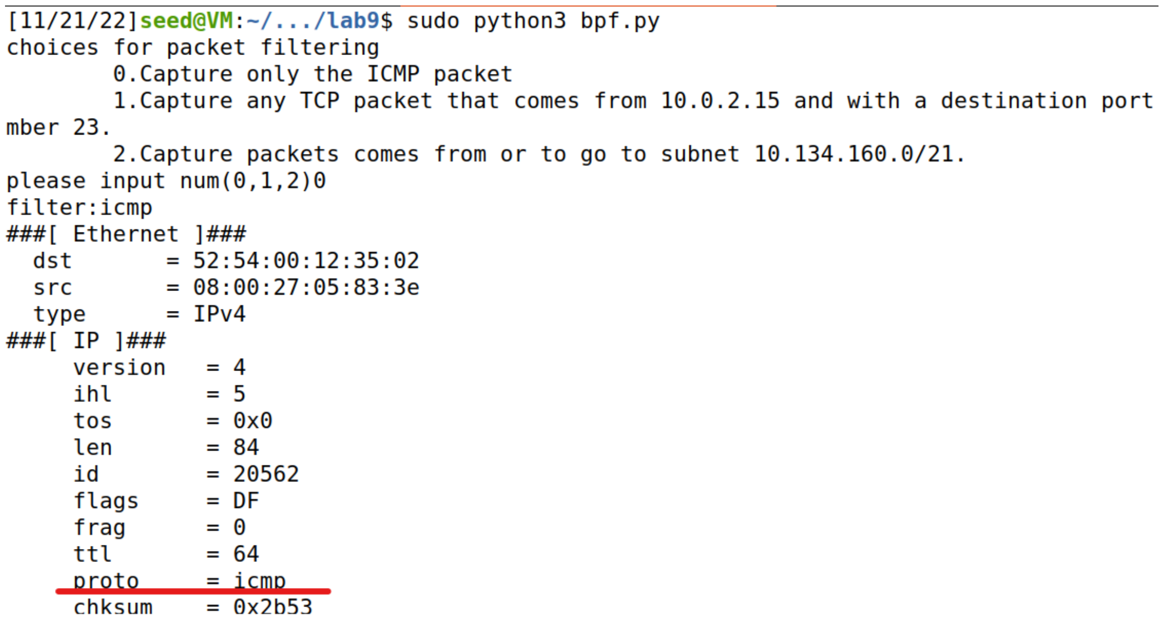
1.1B BPF

编写BPF过滤语法

1. 只捕获ICMP

```
filter='icmp'
```

`ping www.baidu.com`

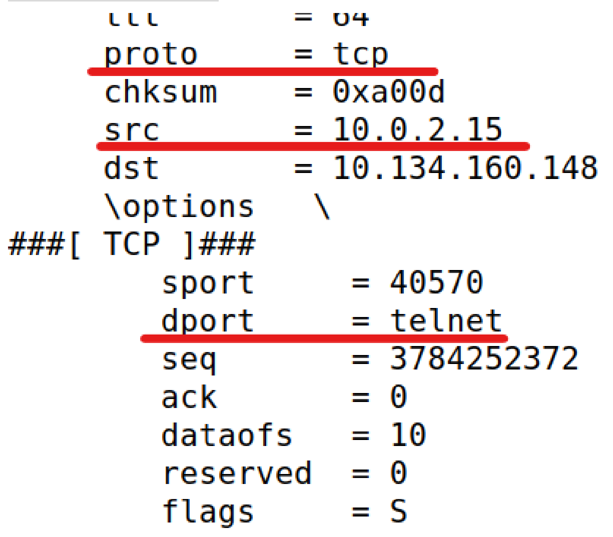


```
[11/21/22]seed@VM:~/.../lab9$ sudo python3 bpf.py
choices for packet filtering
        0.Capture only the ICMP packet
        1.Capture any TCP packet that comes from 10.0.2.15 and with a destination port
mber 23.
        2.Capture packets comes from or to go to subnet 10.134.160.0/21.
please input num(0,1,2)0
filter:icmp
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:05:83:3e
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 20562
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2b53
```

2. 捕获特定源IP和目的端口的TCP包

```
filter='tcp and src net 10.0.2.15 and dst port 23'
```

`telnet IP 23`通过telnet远程连接主机可以生成满足条件的包：

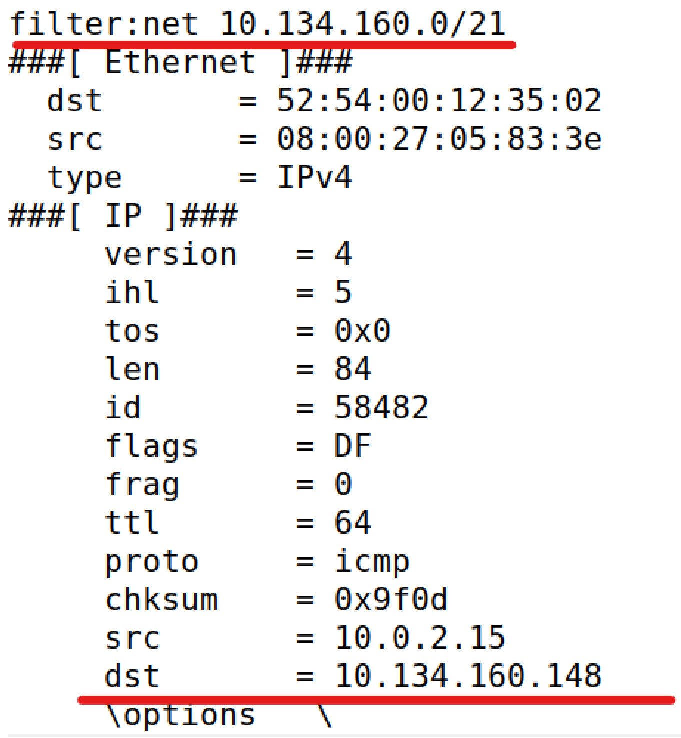


```
        ttl        = 64
        proto      = tcp
        chksum     = 0xa00d
        src        = 10.0.2.15
        dst        = 10.134.160.148
        \options    \
###[ TCP ]###
           sport      = 40570
           dport      = telnet
           seq        = 3784252372
           ack        = 0
           dataofs    = 10
           reserved   = 0
           flags      = S
```

3. 捕获源或目的IP属于某特定子网

```
filter='net 10.134.160.0/21'
```

根据宿主机IP和子网掩码算出网络号如上，开始捕获后ping百度和宿主机，查看接收到的包只有目标为宿主机的icmp包：



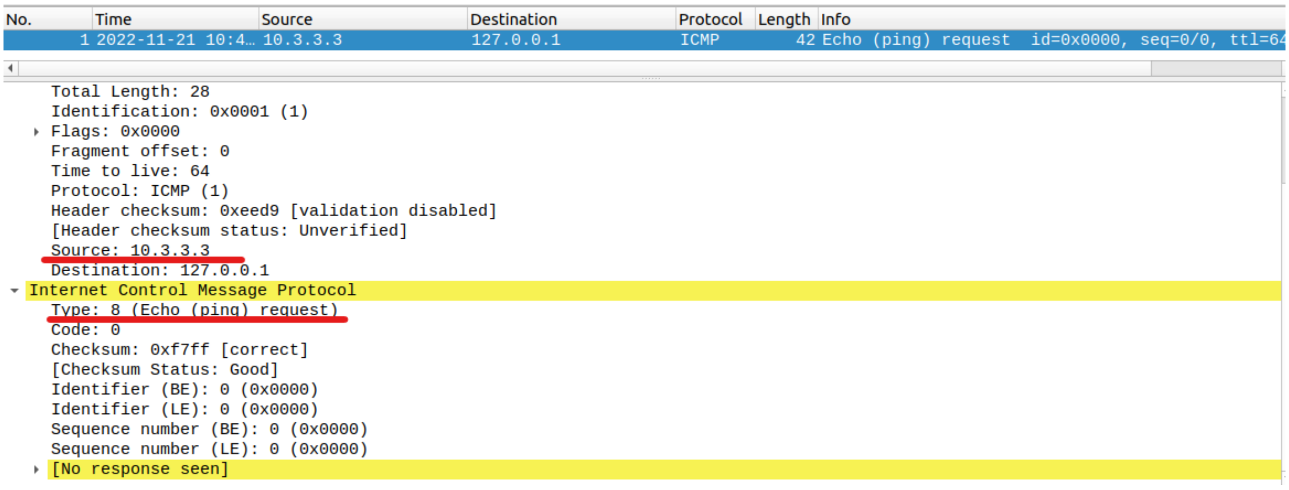
```
filter:net 10.134.160.0/21
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:05:83:3e
  type      = IPv4
###[ IP ]###
        version    = 4
        ihl        = 5
        tos        = 0x0
        len        = 84
        id         = 58482
        flags      = DF
        frag       = 0
        ttl        = 64
        proto      = icmp
        chksum     = 0x9f0d
        src        = 10.0.2.15
        dst        = 10.134.160.148
        \options    \
```

1.2 Spoofing ICMP

spoof.py制作虚假源IP的ICMP包并发送：

```
from scapy.all import *
a=IP()
srcip=input('please input fake srcIP:')
a.src=srcip
b=ICMP()
p=a/b
send(p)
```

由用户输入虚假IP来设置ICMP包的源IP，默认目的地址是`127.0.0.1`，利用wireshark查看环回地址捕获到的ICMP包：



```
No.   Time               Source      Destination  Protocol  Length  Info
    1 2022-11-21 10:4…   10.3.3.3    127.0.0.1    ICMP          42  Echo (ping) request  id=0x0000, seq=0/0, ttl=64

    Total Length: 28
    Identification: 0x0001 (1)
  › Flags: 0x0000
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xeed9 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.3.3.3
    Destination: 127.0.0.1
⌄ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xf7ff [correct]
    [Checksum Status: Good]
    Identifier (BE): 0 (0x0000)
    Identifier (LE): 0 (0x0000)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
  › [No response seen]
```

1.3 TTL

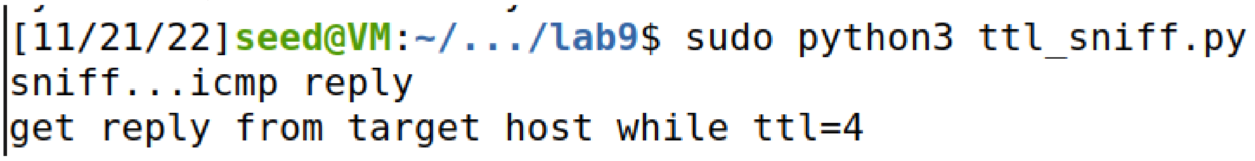
ttl.py 自动ttl增1并发送ICMP包

```
a=IP()
a.dst='...'
b=ICMP()
for i in range(1,65):
    a.ttl=i
    send(a/b)
```

ttl_sniff.py 捕获ICMP应答包并计算成功收到目标主机的应答前已经收到多少个路由器的错误反馈，根据此得出真正的ttl距离：

```
global ttl
ttl=0
def count_ttl(pkt):
    global ttl
    ttl=ttl+1
    if(pkt[ICMP].type==0 and pkt[IP].src=='...'): # type=0是reply
        print('get reply from target host while ttl='+str(ttl))
        exit(0)

print('sniff...icmp reply')
sniff(iface='enp0s3',filter='icmp',prn=count_ttl)
```

启动ttl_sniff.py监听后开始发包，得到结果：



```
[11/21/22]seed@VM:~/.../lab9$ sudo python3 ttl_sniff.py
sniff...icmp reply
get reply from target host while ttl=4
```

1.4 Sniff&Spoof

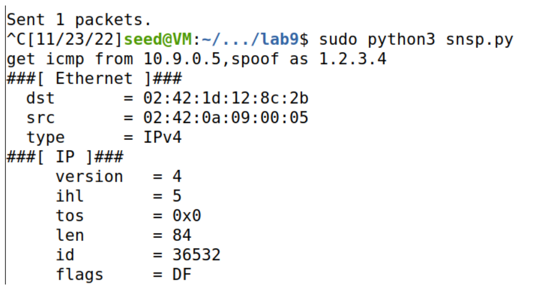
10.9.0.1(虚拟主机和attack都是)为主机A和B的网关，即10.9.0.1对应的网卡能接收到A和B的ICMP包。主机中运行嗅探和欺骗程序snsp.py:

```
def snsf(pkt):
    a=IP(src=pkt[IP].dst,dst=pkt[IP].src)
    b=ICMP(type='echo-
reply',code=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
    p=a/b/pkt[Raw]
    send(p)
```

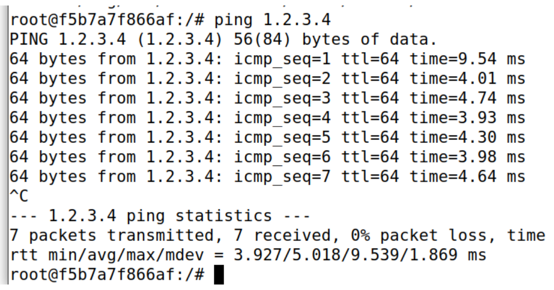
Raw是应用层的数据，如果不加的话，被欺骗的主机能收到欺骗包，但是**ping**程序无法识别是响应包

容器hostA作为被欺骗者，分别ping：

1. 互联网中不存在的主机`1.2.3.4`：hostA能接收到响应包

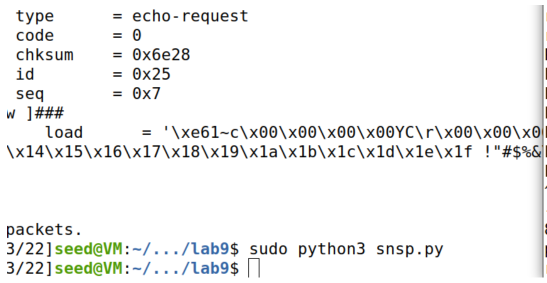


```
Sent 1 packets.
^C[11/23/22]seed@VM:~/.../lab9$ sudo python3 snsp.py
get icmp from 10.9.0.5,spoof as 1.2.3.4
###[ Ethernet ]###
  dst       = 02:42:1d:12:8c:2b
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 36532
     flags     = DF
```

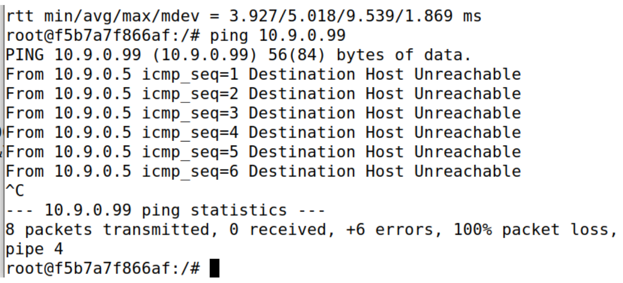


```
root@f5b7a7f866af:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=9.54 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=4.01 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=4.74 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=3.93 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=4.30 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=3.98 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=4.64 ms
^C
--- 1.2.3.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time
rtt min/avg/max/mdev = 3.927/5.018/9.539/1.869 ms
root@f5b7a7f866af:/# 
```

2. 局域网中不存在的主机`10.9.0.99`：unreachable，hostA向局域网内的主机发包不会经过网关，因此攻击机不会收到ICMP包，也就不会返回欺骗包。由于hostA找不到`10.9.0.99`的MAC地址，导致包不可达。

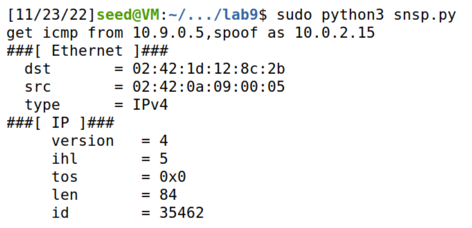


```
 type      = echo-request
 code      = 0
 chksum    = 0x6e28
 id        = 0x25
 seq       = 0x7
w ]###
    load      = '\xe61~c\x00\x00\x00\x00YC\r\x00\x00\x0
\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&


packets.
3/22]seed@VM:~/.../lab9$ sudo python3 snsp.py
3/22]seed@VM:~/.../lab9$ 
```

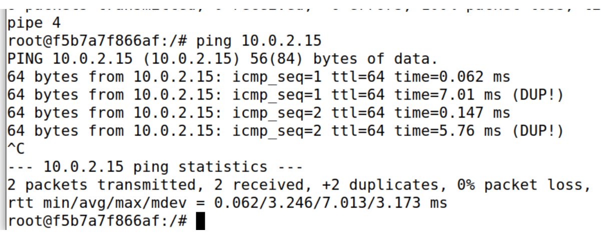


```
rtt min/avg/max/mdev = 3.927/5.018/9.539/1.869 ms
root@f5b7a7f866af:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
8 packets transmitted, 0 received, +6 errors, 100% packet loss,
pipe 4
root@f5b7a7f866af:/# 
```

3. 互联网中存在的主机：虚拟主机。由于目标主机和攻击机都能收到ICMP包并做出响应，导致出现冗余包



```
[11/23/22]seed@VM:~/.../lab9$ sudo python3 snsp.py
get icmp from 10.9.0.5,spoof as 10.0.2.15
###[ Ethernet ]###
  dst       = 02:42:1d:12:8c:2b
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 35462
```

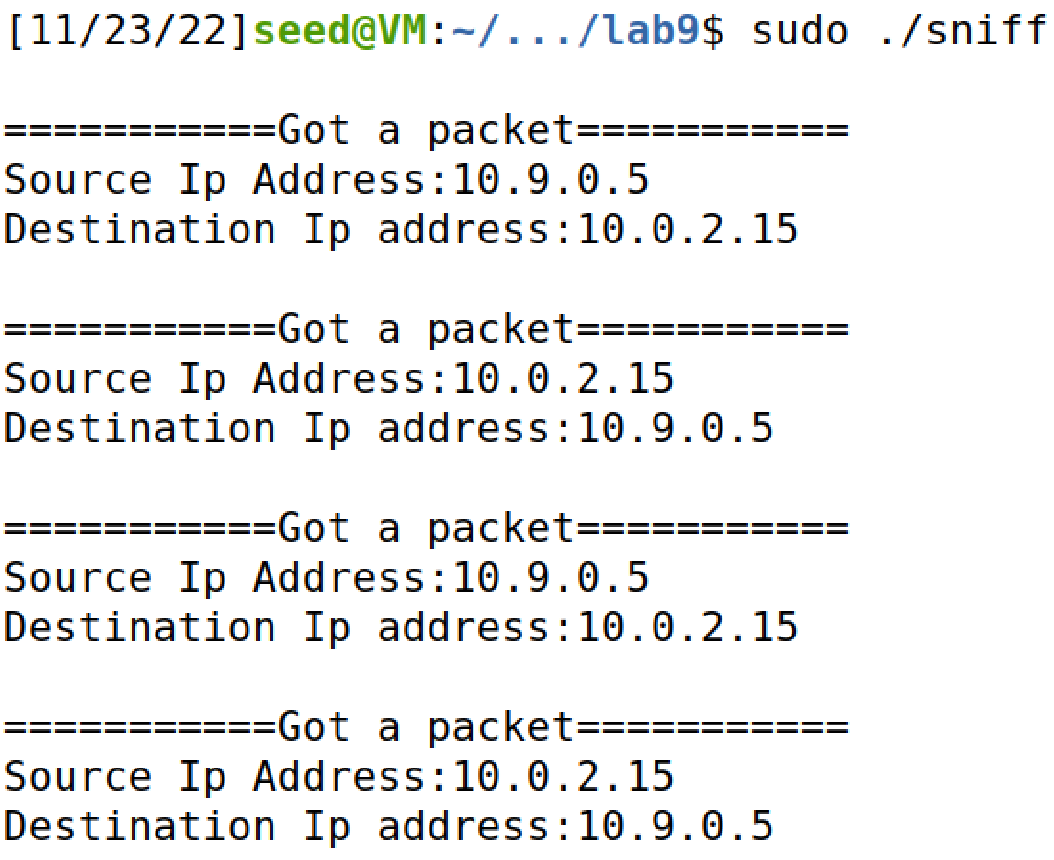


```
pipe 4
root@f5b7a7f866af:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=7.01 ms (DUP!)
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.147 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=5.76 ms (DUP!)
^C
--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, 0% packet loss,
rtt min/avg/max/mdev = 0.062/3.246/7.013/3.173 ms
root@f5b7a7f866af:/# 
```

Task2

2.1A sniff

sniff.c：嗅探局域网中的ICMP包，并打印源IP和目的IP。



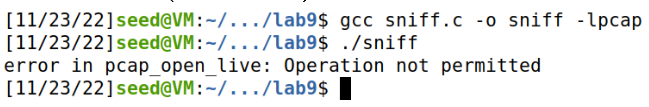
```
[11/23/22]seed@VM:~/.../lab9$ sudo ./sniff

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.0.2.15

===========Got a packet===========
Source Ip Address:10.0.2.15
Destination Ip address:10.9.0.5

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.0.2.15

===========Got a packet===========
Source Ip Address:10.0.2.15
Destination Ip address:10.9.0.5
```

Question1：

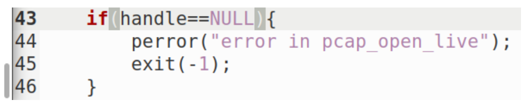
- `pcap_open_live`：打开指定网卡设备
- `pcap_compile,pcap_setfilter`：编译BPF过滤条件并绑定到本次处理；
- `pcap_loop`：开始循环捕获，通过回调函数对捕获到的每一个包进行处理
- `pcap_close`：关闭本次捕获

Question2：

对网卡设备(底层硬件)的操作需要root权限，从第一步打开网卡就需要权限：



```
[11/23/22]seed@VM:~/.../lab9$ gcc sniff.c -o sniff -lpcap
[11/23/22]seed@VM:~/.../lab9$ ./sniff
error in pcap_open_live: Operation not permitted
[11/23/22]seed@VM:~/.../lab9$ 
```

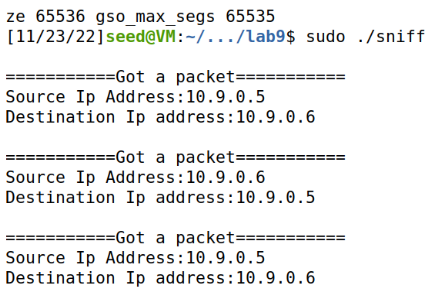


```
    if(handle==NULL){
        perror("error in pcap_open_live");
        exit(-1);
    }
```

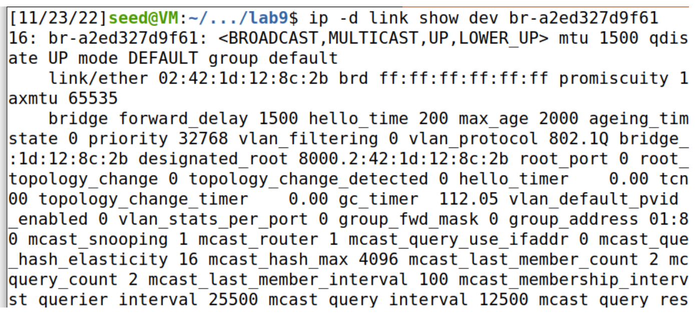
Question3

混杂模式开启后能接收目的地址不是本机的数据包。用`10.9.0.1`的网卡设备作为嗅探设备，用hostA ping hostB，A和B在同一局域网，不会发给网关。

混杂模式开启时：`10.9.0.1`能接收到ICMP包

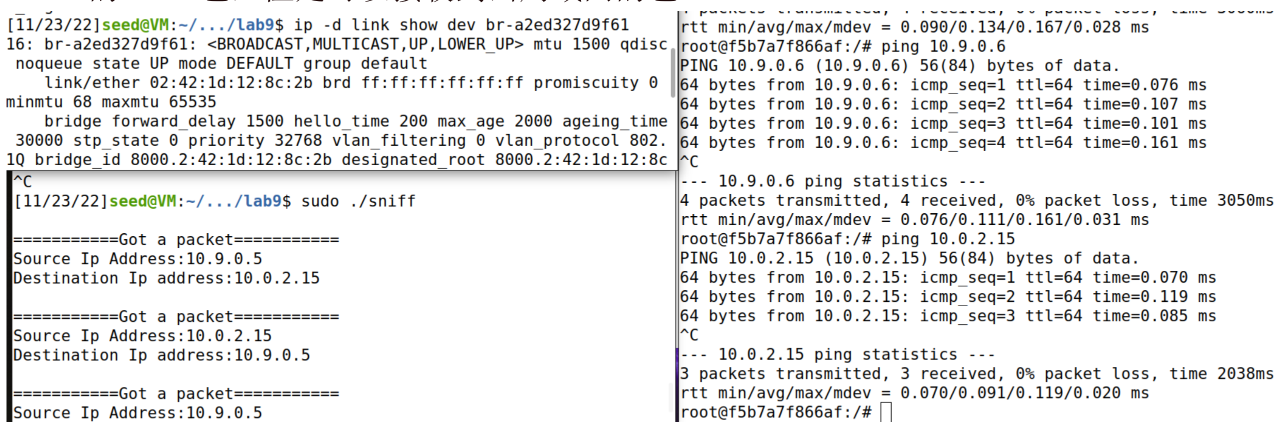


```
ze 65536 gso_max_segs 65535
[11/23/22]seed@VM:~/.../lab9$ sudo ./sniff

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.9.0.6

===========Got a packet===========
Source Ip Address:10.9.0.6
Destination Ip address:10.9.0.5

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.9.0.6
```



```
[11/23/22]seed@VM:~/.../lab9$ ip -d link show dev br-a2ed327d9f61
16: br-a2ed327d9f61: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdis
ate UP mode DEFAULT group default
    link/ether 02:42:1d:12:8c:2b brd ff:ff:ff:ff:ff:ff promiscuity 1
axmtu 65535
    bridge forward_delay 1500 hello_time 200 max_age 2000 ageing_tim
state 0 priority 32768 vlan_filtering 0 vlan_protocol 802.1Q bridge_
:1d:12:8c:2b designated_root 8000.2:42:1d:12:8c:2b root_port 0 root_
topology_change 0 topology_change_detected 0 hello_timer    0.00 tcn
00 topology_change_timer    0.00 gc_timer  112.05 vlan_default_pvid
_enabled 0 vlan_stats_per_port 0 group_fwd_mask 0 group_address 01:8
0 mcast_snooping 1 mcast_router 1 mcast_query_use_ifaddr 0 mcast_que
_hash_elasticity 16 mcast_hash_max 4096 mcast_last_member_count 2 mc
query_count 2 mcast_last_member_interval 100 mcast_membership_interv
st querier interval 25500 mcast query interval 12500 mcast query res
```

修改`pcap_live_open`参数中的1为0，关闭混杂模式：`10.9.0.1`不能接收到hostA ping hostB的ICMP包，但是可以接收到出局域网的包。



```
[11/23/22]seed@VM:~/.../lab9$ ip -d link show dev br-a2ed327d9f61
16: br-a2ed327d9f61: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
 noqueue state UP mode DEFAULT group default
    link/ether 02:42:1d:12:8c:2b brd ff:ff:ff:ff:ff:ff promiscuity 0
minmtu 68 maxmtu 65535
    bridge forward_delay 1500 hello_time 200 max_age 2000 ageing_time
 30000 stp_state 0 priority 32768 vlan_filtering 0 vlan_protocol 802.
1Q bridge_id 8000.2:42:1d:12:8c:2b designated_root 8000.2:42:1d:12:8c
```

```
^C
[11/23/22]seed@VM:~/.../lab9$ sudo ./sniff

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.0.2.15

===========Got a packet===========
Source Ip Address:10.0.2.15
Destination Ip address:10.9.0.5

===========Got a packet===========
Source Ip Address:10.9.0.5
```

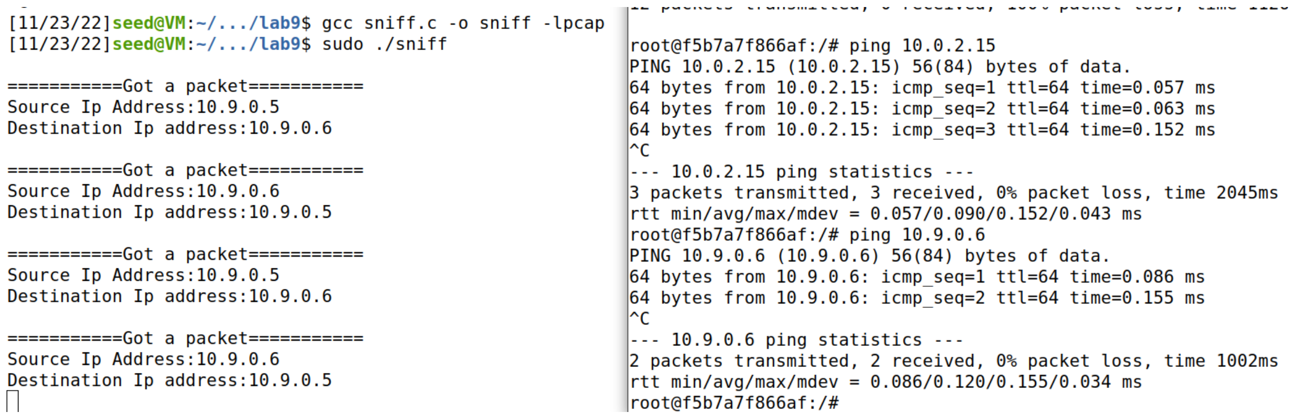
```
rtt min/avg/max/mdev = 0.090/0.134/0.167/0.028 ms
root@f5b7a7f866af:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.107 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.101 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.161 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.076/0.111/0.161/0.031 ms
root@f5b7a7f866af:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.070 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.119 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.085 ms
^C
--- 10.0.2.15 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.070/0.091/0.119/0.020 ms
root@f5b7a7f866af:/#
```

2.1B filter

1.捕获两个特定主机之间的ICMP数据包

```
filter="icmp and host 10.9.0.5 and host 10.9.0.6"
```

只捕获`10.9.05`和`10.9.0.6`之间的，ping其他IP时则不会收到



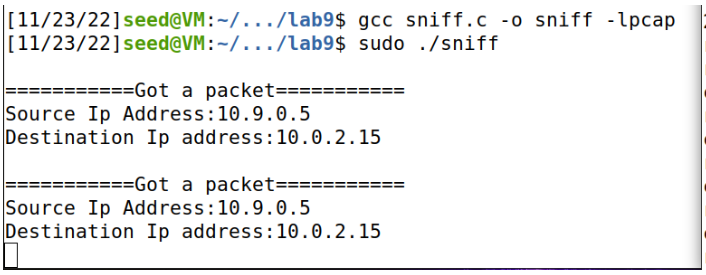
```
[11/23/22]seed@VM:~/.../lab9$ gcc sniff.c -o sniff -lpcap
[11/23/22]seed@VM:~/.../lab9$ sudo ./sniff

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.9.0.6

===========Got a packet===========
Source Ip Address:10.9.0.6
Destination Ip address:10.9.0.5

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.9.0.6

===========Got a packet===========
Source Ip Address:10.9.0.6
Destination Ip address:10.9.0.5
```

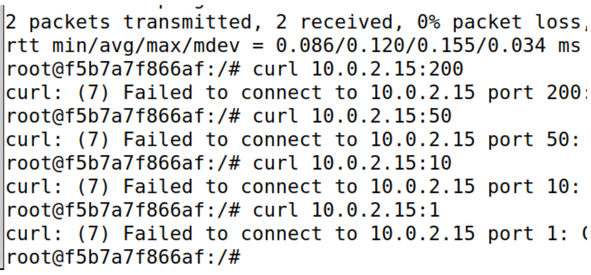
```
root@f5b7a7f866af:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.057 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.152 ms
^C
--- 10.0.2.15 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2045ms
rtt min/avg/max/mdev = 0.057/0.090/0.152/0.043 ms
root@f5b7a7f866af:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.086 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.155 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.086/0.120/0.155/0.034 ms
root@f5b7a7f866af:/#
```

2.捕获目标端口号在10到100范围内的TCP数据包。

```
filter="tcp and dst portrange 10-100"
```

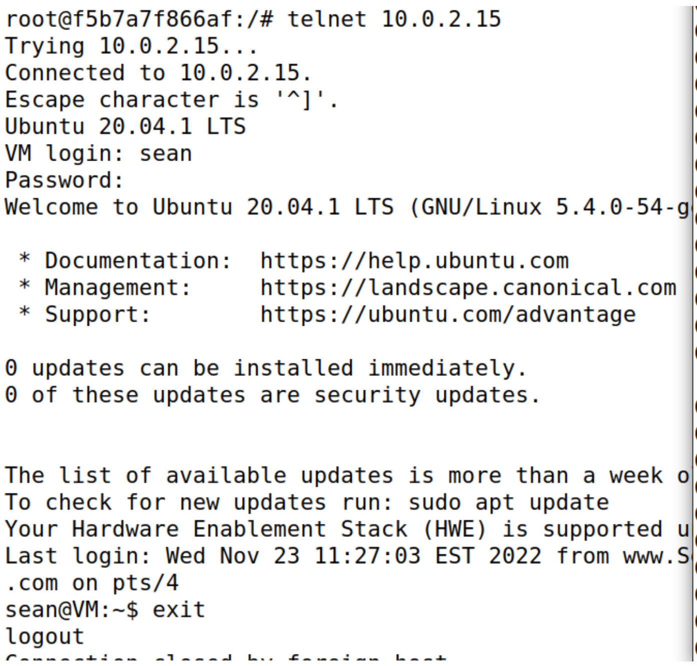
用curl对多个端口发起请求，只有10-100之间的才可以被捕获到

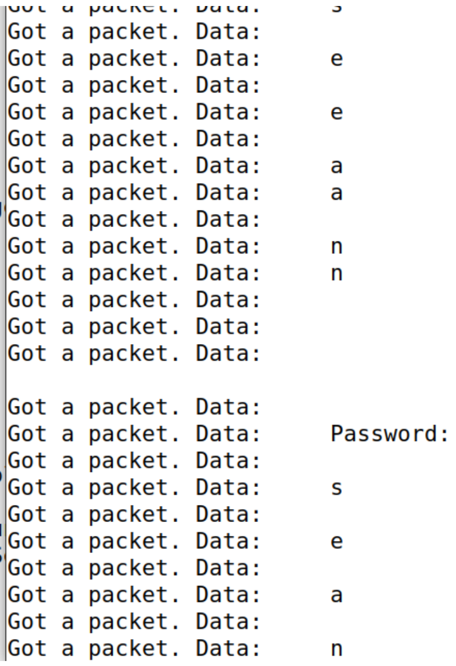

```
[11/23/22]seed@VM:~/.../lab9$ gcc sniff.c -o sniff -lpcap
[11/23/22]seed@VM:~/.../lab9$ sudo ./sniff

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.0.2.15

===========Got a packet===========
Source Ip Address:10.9.0.5
Destination Ip address:10.0.2.15
```


```
2 packets transmitted, 2 received, 0% packet loss,
rtt min/avg/max/mdev = 0.086/0.120/0.155/0.034 ms
root@f5b7a7f866af:/# curl 10.0.2.15:200
curl: (7) Failed to connect to 10.0.2.15 port 200:
root@f5b7a7f866af:/# curl 10.0.2.15:50
curl: (7) Failed to connect to 10.0.2.15 port 50:
root@f5b7a7f866af:/# curl 10.0.2.15:10
curl: (7) Failed to connect to 10.0.2.15 port 10:
root@f5b7a7f866af:/# curl 10.0.2.15:1
curl: (7) Failed to connect to 10.0.2.15 port 1: (
root@f5b7a7f866af:/#
```

2.1C telnet

嗅探TCP包中的敏感数据，以telnet为例：telnet远程连接，捕获其登录名和密码。


```
root@f5b7a7f866af:/# telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: sean
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-g

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

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week o
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported u
Last login: Wed Nov 23 11:27:03 EST 2022 from www.S
.com on pts/4
sean@VM:~$ exit
logout
Connection closed by foreign host
```


```
Got a packet. Data:     s
Got a packet. Data:
Got a packet. Data:     e
Got a packet. Data:
Got a packet. Data:     e
Got a packet. Data:
Got a packet. Data:     a
Got a packet. Data:     a
Got a packet. Data:
Got a packet. Data:     n
Got a packet. Data:     n
Got a packet. Data:
Got a packet. Data:
Got a packet. Data:

Got a packet. Data:
Got a packet. Data:     Password:
Got a packet. Data:
Got a packet. Data:     s
Got a packet. Data:
Got a packet. Data:     e
Got a packet. Data:
Got a packet. Data:     a
Got a packet. Data:
Got a packet. Data:     n
```

Telnet发送的数据固定在报文第**66**字节，可以直接输出内容

2.2A spoof IP

自定义报头结构体，根据需要指定字段内容：根据**wireshark**抓包可得，除了**IP**和**ICMP**的报文头，还有**8**个字节的时间戳和**48**个字节的填充数据

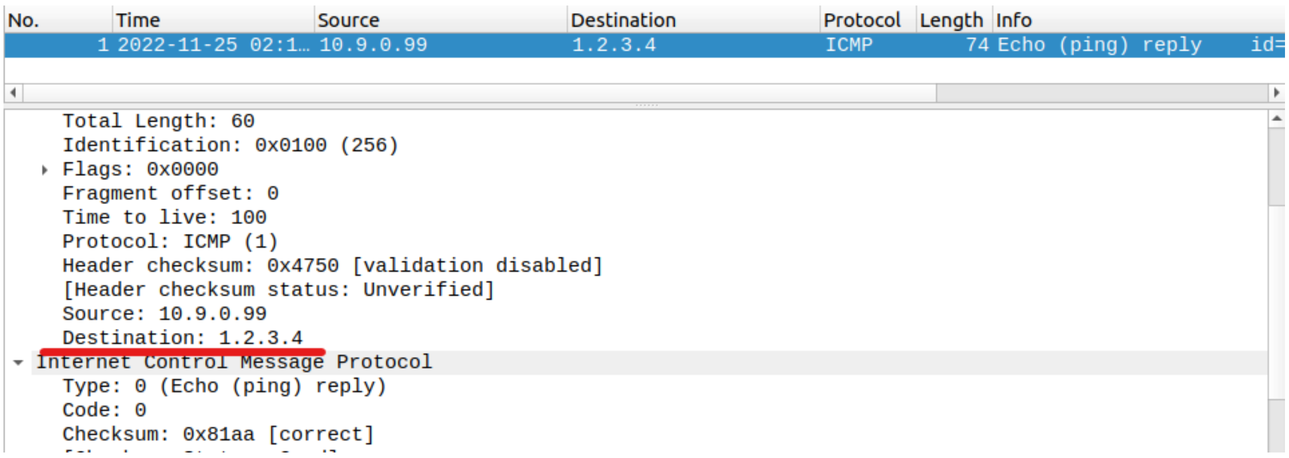
```
//IP报头
typedef struct IP_HEADER {
#if defined(__LITTLE_ENDIAN_BITFIELD)
    __u8    ihl:4,          //4位头部长度 一位4个字节，，最多15*4个字节(可选项)
        version:4;      //4位版本号
#elif defined (__BIG_ENDIAN_BITFIELD)
    __u8    version:4,
        ihl:4;
#else
#error  "Please fix <asm/byteorder.h>"
#endif
    __u8    tos;            //8位服务类型
    __be16  tot_len;        //16位总长度
    __be16  id;         //16位标识符
    __be16  frag_off;       //3位标志加13位片偏移
    __u8    ttl;            //8位生存时间
    __u8    protocol;       //8位上层协议号
    __sum16 check;      //16位校验和
    __be32  saddr;          //32位源IP地址
    __be32  daddr;          //32位目的IP地址
    /*The options start here. */
} IP_HEADER;

//ICMP报头
typedef struct ICMP_HEADER
{
    u_char type;    //8位类型字段
    u_char code;    //8位代码字段
    u_short cksum; //16位校验和
    u_short id;    //16位标识符
    u_short seq;   //16位序列号
} ICMP_HEADER;
```

使用原始套接字，根据数据包中的源**IP**构造对应的**sockaddr_in**，发送自定义IP数据包：

```
struct sockaddr_in sin;
sin.sin_family = AF_INET;
sin.sin_addr.s_addr=pIPHeader->daddr;
```

```
 sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
 if(sd < 0) {
         perror("socket() error"); exit(-1);
 }
 sin.sin_family = AF_INET;
 int ip_len = (sizeof(IP_HEADER) + sizeof(ICMP_HEADER) +
P_DATA_SIZE);
 if(sendto(sd, buffer, ip_len, 0, (struct sockaddr *)&sin,
sin)) < 0) {
         perror("sendto() error"); exit(-1);
```

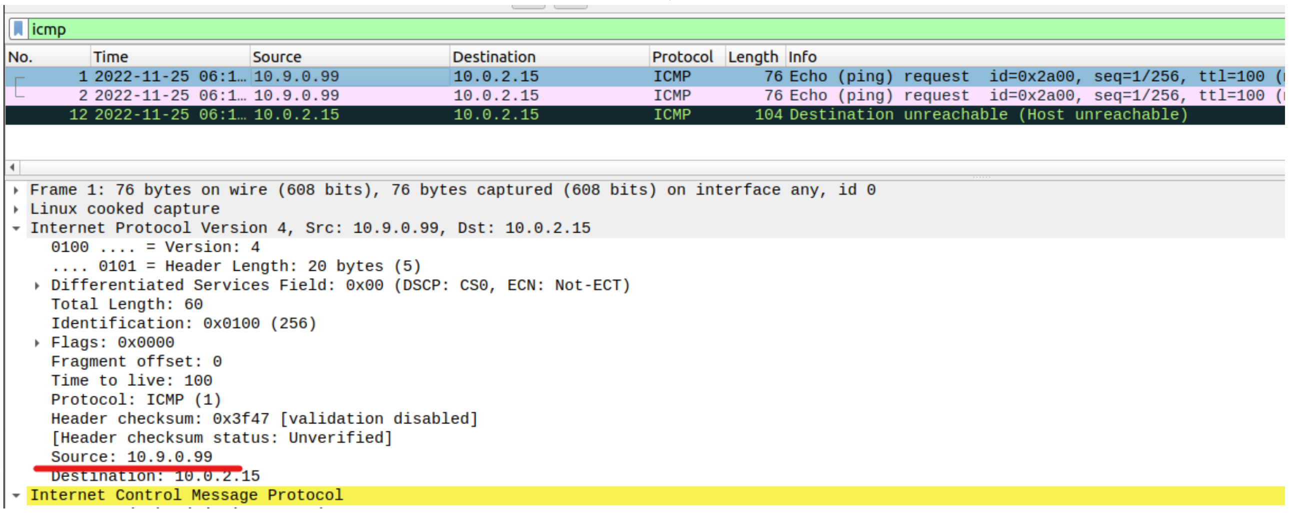
修改源和目的IP，用wireshark查看结果：



```
No.   Time                Source      Destination  Protocol Length Info
    1 2022-11-25 02:1… 10.9.0.99   1.2.3.4      ICMP         74 Echo (ping) reply    id=

    Total Length: 60
    Identification: 0x0100 (256)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 100
    Protocol: ICMP (1)
    Header checksum: 0x4750 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.9.0.99
    Destination: 1.2.3.4
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x81aa [correct]
```

2.2B spoof echo-request

构造ICMP包与2.1A类似，以局域网中不存在的主机为源IP向某存活主机发起request，由于源IP是不可达的，所以无法收到存活主机的reply



```
icmp
No.  Time                Source      Destination  Protocol Length Info
   1 2022-11-25 06:1… 10.9.0.99   10.0.2.15    ICMP         76 Echo (ping) request  id=0x2a00, seq=1/256, ttl=100 (
   2 2022-11-25 06:1… 10.9.0.99   10.0.2.15    ICMP         76 Echo (ping) request  id=0x2a00, seq=1/256, ttl=100 (
  12 2022-11-25 06:1… 10.0.2.15   10.0.2.15    ICMP        104 Destination unreachable (Host unreachable)

▸ Frame 1: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
▸ Linux cooked capture
▾ Internet Protocol Version 4, Src: 10.9.0.99, Dst: 10.0.2.15
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x0100 (256)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 100
    Protocol: ICMP (1)
    Header checksum: 0x3f47 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.9.0.99
    Destination: 10.0.2.15
▾ Internet Control Message Protocol
```

Question4:

可以随意设置IP报文头部的长度，但必须保证向socket发送的原始报文是正确的长度。

Question5:

必须要计算校验和，若校验不通过则该数据包会被丢弃。

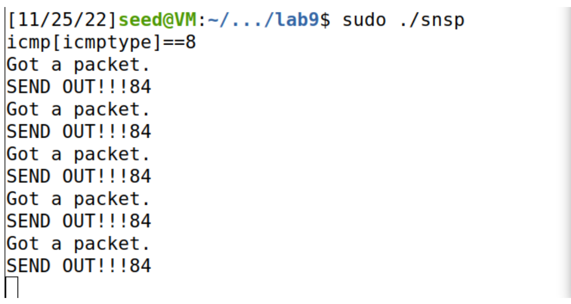
Question6:

创建原始套接字需要root权限。若没有权限， `socket(AF_INET, SOCK_RAW, IPPROTO_RAW);`这一步就无法成功创建套接字。

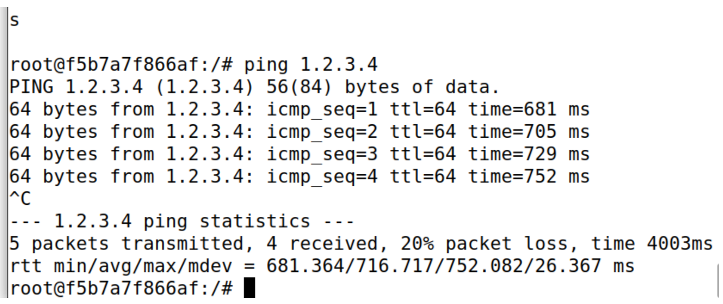
2.3 Sniff&Spoof

结合以上的代码，实现在嗅探的同时发送欺骗响应包。

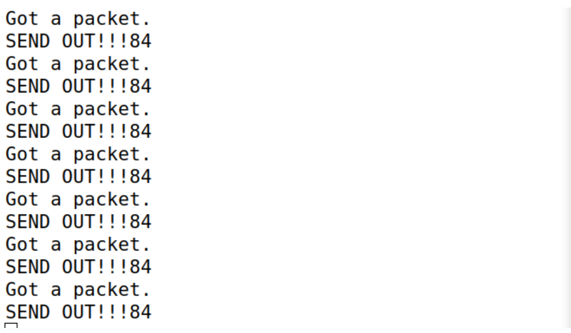
结果与Task1.4类似



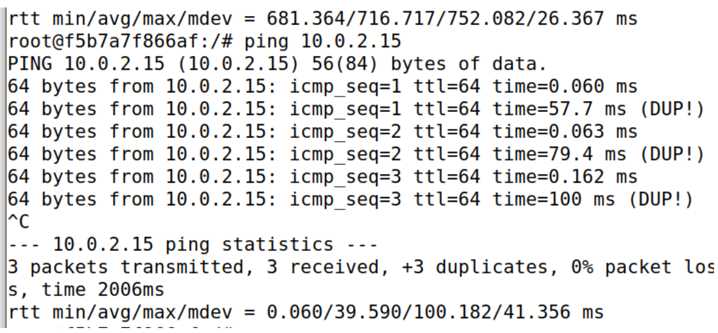
```
[11/25/22]seed@VM:~/.../lab9$ sudo ./snsp
icmp[icmptype]==8
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
```



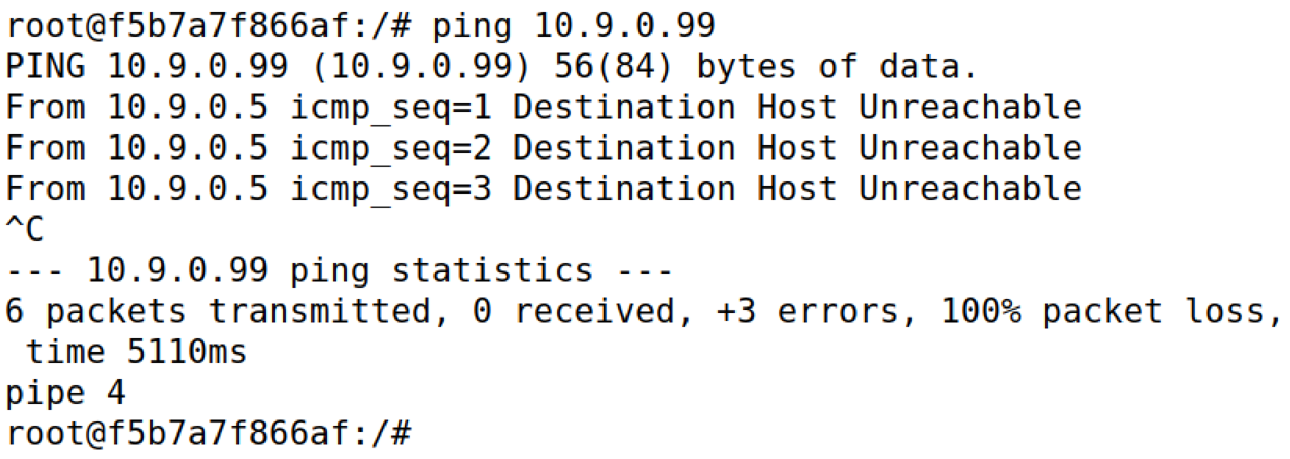
```
s

root@f5b7a7f866af:/# 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=681 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=705 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=729 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=752 ms
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4003ms
rtt min/avg/max/mdev = 681.364/716.717/752.082/26.367 ms
root@f5b7a7f866af:/# 
```



```
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
Got a packet.
SEND OUT!!!84
```



```
rtt min/avg/max/mdev = 681.364/716.717/752.082/26.367 ms
root@f5b7a7f866af:/# ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=57.7 ms (DUP!)
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=79.4 ms (DUP!)
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.162 ms
64 bytes from 10.0.2.15: icmp_seq=3 ttl=64 time=100 ms (DUP!)
^C
--- 10.0.2.15 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet los
s, time 2006ms
rtt min/avg/max/mdev = 0.060/39.590/100.182/41.356 ms
```



```
root@f5b7a7f866af:/# 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
^C
--- 10.9.0.99 ping statistics ---
6 packets transmitted, 0 received, +3 errors, 100% packet loss,
 time 5110ms
pipe 4
root@f5b7a7f866af:/#
```

总结

- 用C语言构造报文头比python难很多，但是可操作性强，可自定义每一个字节的数据；
- 接发包不只是要构造源和目的的地址，对于操作系统和特定程序来说，都有其必要的检验机制，如本次实验中ping检验了Raw数据。
- 对C语言的数据类型及其在内存中的具体情况还不是很清楚。