Lab report

Task Set 1: Using Tools to Sniff and Spoof Packets

Task 1.1: Sniffing Packets

Task 1.1A.

可以观察到，当 root 权限运行时，并且打开浏览器的网站时，会有大量的数据包打印在屏幕上，可以证明能收到网络包。若没有在 root 权限下运行程序，则程序无法运行，因为网络数据包在网卡上接受，要获取数据包需要通过底层的硬件，而这是内核程序控制，一般程序在无权限的情况中，无法直接与底层硬件交互，所以需要在 root 权限下运行此程序。

```
^C[09/23/18]seed@VM:~/.../lab0$ cat mycode.py

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='tcp',prn=print_pkt)


[09/23/18]seed@VM:~/.../lab0$ python mycode.py
Traceback (most recent call last):
  File "mycode.py", line 7, in <module>
    pkt = sniff(filter='tcp',prn=print_pkt)
  File "/usr/local/lib/python2.7/dist-packages/scapy/sendrecv.py", line 731
    *arg, **karg)] = iface
  File "/usr/local/lib/python2.7/dist-packages/scapy/arch/linux.py", line 5
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.hton
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
[09/23/18]seed@VM:~/.../lab0$ sudo python mycode.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:85:78:69
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 6465
```

```
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 242
     id        = 6467
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xceb4
     src       = 10.0.2.4
     dst       = 115.239.210.27
     \options   \
###[ TCP ]###
        sport     = 34746
        dport     = https
        seq       = 3393209847L
        ack       = 12602
        dataofs   = 5
        reserved  = 0
        flags     = PA
        window    = 29200
        chksum    = 0x52f3
        urgptr    = 0
        options   = []
###[ Raw ]###
           load      = '\x16\x03\x01\x00\xc5\x01\x00\x00\xc1\x03\x03\x80\xb9\x81\x9aWt3Z\x9
c\xb5\x17"\x80\xdfS2d\xa7\xf4\xd5\xac\xaf\x00\x00\x1e\xc0+\xc0/\xcc\xa9\xcc\xa8\xc0,\xc00\x
x00/\x005\x00\n\x01\x00\x00z\x00\x00\x00\x12\x00\x10\x00\x00\rwww.baidu.com\x00\x17\x00\x00
x08\x00\x1d\x00\x17\x00\x18\x00\x19\x00\x0b\x00\x02\x01\x00\x00#\x00\x00\x00\x10\x00\x0e\x0
05\x01\x00\x00\x00\x00\xff\x03\x00\x00\x00\r\x00 \x00\x1e\x04\x03\x05\x03\x06\x03\x02\x03\x
1\x06\x01\x02\x01\x04\x02\x05\x02\x06\x02\x02\x02'
```

Task 1.1B

Capture only the ICMP packet

代码:

```
^C[09/23/18]seed@VM:~/.../lab0$ cat mycode.py

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

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

通过 ping 124.0.0.5（即另一台未启动的虚拟机网址）获取 icmp 包。

```
[09/23/18]seed@VM:~$ ping 124.0.0.5
PING 124.0.0.5 (124.0.0.5) 56(84) bytes of data.
```

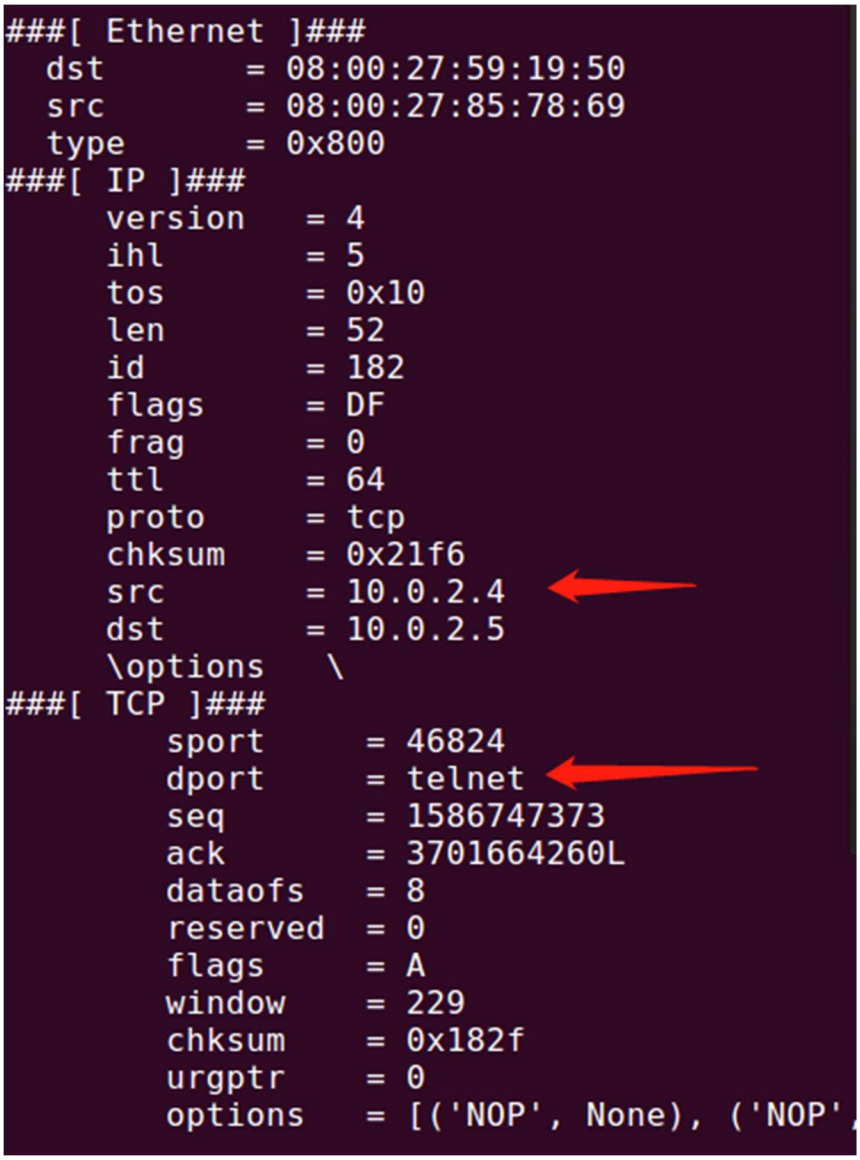
```
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:85:78:69
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 27426
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x477e
     src       = 10.0.2.4
     dst       = 124.0.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x56f5
        id        = 0xa5e
        seq       = 0xe
###[ Raw ]###
           load      = '\xdcj\xa7[\x1e\xd5
1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

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

抓取网络包，sniff 的 filter 函数，IP:10.0.2.4 port:23 以及是 tcp 报文　　Code:

```
^C[10/02/18]seed@VM:~/.../实验报告$ cat task1.1Btcp_port23.py
from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='host 10.0.2.4 and tcp port 23',prn=print_pkt)
```

抓取的结果是：



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.

选择的子网是 source 202.120.224.0/24

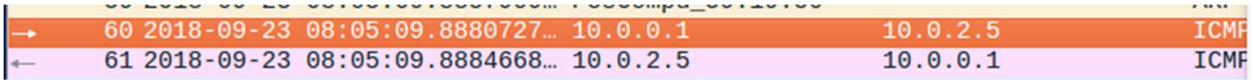
```
[09/23/18]seed@VM:~/.../lab0$ cat mycode.py

from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(filter='src net 202.120.224',prn=print_pkt)


[09/23/18]seed@VM:~/.../lab0$ vim mycode.py
[09/23/18]seed@VM:~/.../lab0$ sudo python mycode.py
###[ Ethernet ]###
  dst       = 08:00:27:85:78:69
  src       = 52:54:00:12:35:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 166
     id        = 10650
     flags     =
     frag      = 0
     ttl       = 255
     proto     = udp
     chksum    = 0xdb15
     src       = 202.120.224.26
     dst       = 10.0.2.4
     \options   \
```

```
###[ Ethernet ]###
  dst       = 08:00:27:85:78:69
  src       = 52:54:00:12:35:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 44
     id        = 10712
     flags     =
     frag      = 0
     ttl       = 255
     proto     = tcp
     chksum    = 0xdb04
     src       = 202.120.224.114
     dst       = 10.0.2.4
     \options   \
###[ TCP ]###
        sport     = http
        dport     = 43094
        seq       = 309138
        ack       = 909034845
        dataofs   = 6
        reserved  = 0
        flags     = SA
        window    = 32768
        chksum    = 0x55e
        urgptr    = 0
        options   = [('MSS', 1460)]
```

Task 1.2: Spoofing ICMP Packets

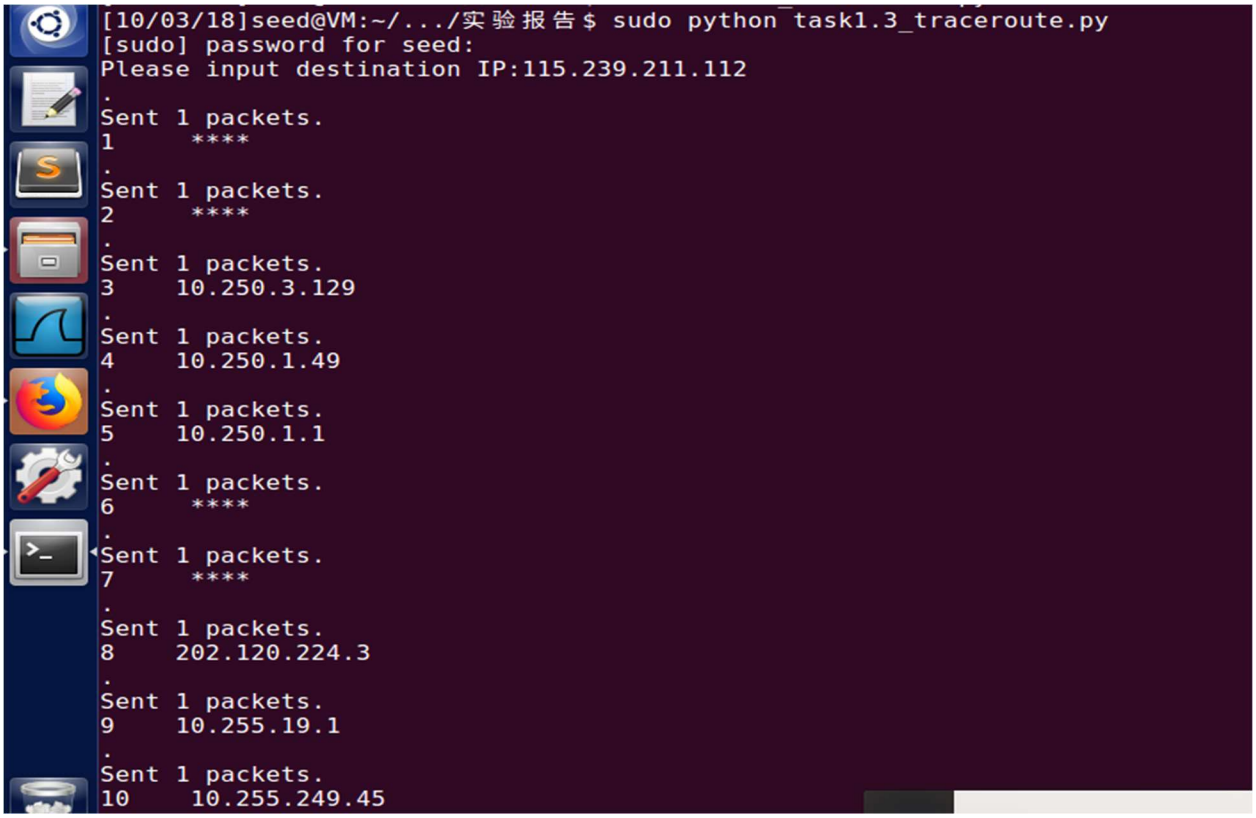
简单的欺骗，代码以及获取的结果：

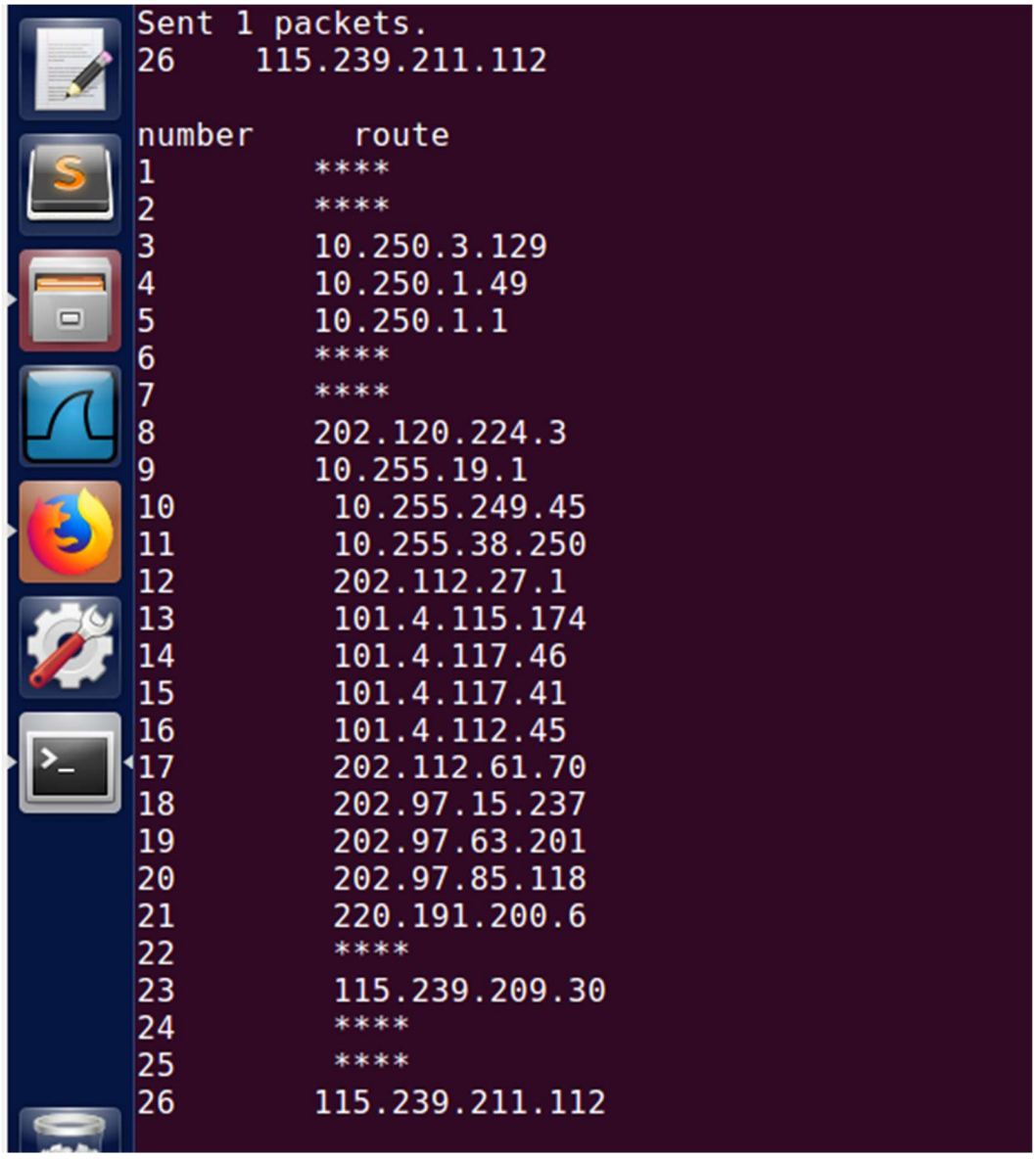
```
[09/23/18]seed@VM:~/.../lab0$ cat task1.py
from scapy.all import *

a = IP()
a.src = "10.0.0.1"
a.dst = "10.0.2.5"

b = ICMP()

p=a/b
send(p)
```



Task 1.3: Traceroute

为了处理收包和发包以及程序的结束，我使用了线程以及信号量来处理细节，通过查阅 scapy 文档，发现 scapy 有自带的路由函数，可以直接使用。下图是我获取 www.baidu.com 的一个 IPv4 的地址来获取其路由路径。发现其发送包时有 echo，最后我通过文件的形式保存数据，得到其完整的路由路径。相关程序：task1.3_traceroute.py





Task 1.4: Sniffing and-then Spoofing

代码如下图:

```
[10/04/18]seed@VM:~/.../实验报告$ cat task1.4_Sniffing_Spoofing.py
from scapy.all import *
import commands

def sendICMP(pkt):
    temp = pkt[IP].dst
    pkt[IP].dst = pkt[IP].src
    pkt[IP].src = temp
    del pkt[ICMP].chksum
    a = IP(pkt[IP])
    a[ICMP].type = 0
    del a[ICMP].chksum
    send(a)


def print_pkt(pkt):
    if pkt[ICMP].type == 8:
        sendICMP(pkt)


def recICMP():
    pkt = sniff(filter='icmp', prn=print_pkt)

def main():
    recICMP()

if __name__ == '__main__':
    main()
```

在虚拟机 B（ip 10.0.2.5）上 ping 222.222.222.222，在虚拟机 A(Ip 10.0.2.4)上运行伪造报文程序，得到如图所示回复：

```
[10/04/18]seed@VM:~$ ping -c 4 222.222.222.222
PING 222.222.222.222 (222.222.222.222) 56(84) bytes of data.
64 bytes from 222.222.222.222: icmp_seq=1 ttl=64 time=7.41 ms
64 bytes from 222.222.222.222: icmp_seq=2 ttl=64 time=10.1 ms
64 bytes from 222.222.222.222: icmp_seq=3 ttl=64 time=9.47 ms
64 bytes from 222.222.222.222: icmp_seq=4 ttl=64 time=7.56 ms

--- 222.222.222.222 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 7.415/8.647/10.137/1.189 ms
[10/04/18]seed@VM:~$
```

通过 wireshark 获取包的信息如图：

No.	Time	Source	Destination	Protocol	Length	Info
→ 1	2018-10-04 00:21:43…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x0f3c, seq=1/256, ttl=64 (reply in 2)
← 2	2018-10-04 00:21:43…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x0f3c, seq=1/256, ttl=64 (request in 1)
3	2018-10-04 00:21:44…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x0f3c, seq=2/512, ttl=64 (reply in 4)
4	2018-10-04 00:21:44…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x0f3c, seq=2/512, ttl=64 (request in 3)
5	2018-10-04 00:21:45…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x0f3c, seq=3/768, ttl=64 (reply in 6)
6	2018-10-04 00:21:45…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x0f3c, seq=3/768, ttl=64 (request in 5)
7	2018-10-04 00:21:46…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x0f3c, seq=4/1024, ttl=64 (reply in 8)
8	2018-10-04 00:21:46…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x0f3c, seq=4/1024, ttl=64 (request in 7)

观察 VMA 上的程序的回显：

```
^C[10/04/18]seed@VM:~/.../实验报告$ sudo python task1.4_Sniffing_Spoofing.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

说明伪造过程完成。

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

Task 2.1: Writing Packet Sniffing Program

Task 2.1A: Understanding How a SnifferWorks

代码 task2.1A_capture_pkt.c，效果如图：

```
[10/04/18]seed@VM:~/.../实验报告$ gcc -o sniff task2.1A_capture_pkt.c -lpcap
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:61.129.42.6
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:61.129.42.6
src_address:61.129.42.6        dst_address:10.0.2.4
src_address:61.129.42.6        dst_address:10.0.2.4
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:61.129.42.6
src_address:10.0.2.4        dst_address:61.129.42.6
src_address:61.129.42.6        dst_address:10.0.2.4
src_address:61.129.42.6        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:203.208.41.72
src_address:10.0.2.4        dst_address:203.208.41.190
src_address:203.208.41.72        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:203.208.41.72
src_address:203.208.41.190        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:203.208.41.190
src_address:10.0.2.4        dst_address:203.208.41.72
src_address:10.0.2.4        dst_address:61.129.42.6
```

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

首先，我们需要设置抓包的网卡，让其在获得包后，能够交给sniff去处理，所以接着必须要sniff能够与网卡建立连接，即会话机制。然后根据所设置的filter来选择相关的报文，将过滤好的报文交给sniff的相关程序，然后把报文解析后，根据其程序来进行不同的操作，最后当我们完成我们的需求或者完成相关的程序后，结束我们的会话。关闭程序，恢复其网卡的设置。

• Question 2. Why do you need the root privilege to run a sniffer program? Where does the program
fail if it is executed without the root privilege?

需要root权限时因为他要访问网络接口，获取其底层的硬件交互。前面已经说明Linux中没有root访问权限时不可能网卡的。而Sniff需要原始套接字，允许应用程序直接发送数据包括，绕过操作系统网络软件中的所有应用程序。

• Quesiton 3. Please turn on and turn off the promiscuous mode in your sniffer program. Can you
demonstrate the difference when this mode is on and off? Please describe how you can demonstrate
this.

混杂模式的设置在 pcap_open_live（）函数的第三个参数，0 为混杂模式，1 为非混杂模式，所以我们可以通过修改参数来开启混杂模式。我们可以通过其抓包的源地址和目的地址是否有本机 ip 来证明它，如下图一个中所受到的包地址中必含有 10.0.2.4，而另一个是没有这中情况的，且源地址和目的地址更加多样化。

```
[10/04/18]seed@VM:~/.../实验报告$ gcc -o sniff task2.1A_capture_pkt.c -lpcap
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
src_address:10.0.2.5        dst_address:10.0.2.255
src_address:10.0.2.5        dst_address:10.0.2.255
src_address:10.0.2.5        dst_address:10.0.2.255
src_address:10.0.2.5        dst_address:10.0.2.255
src_address:10.0.2.5        dst_address:224.0.0.251
src_address:10.0.2.5        dst_address:224.0.0.251
src_address:10.0.2.5        dst_address:255.255.255.255
src_address:10.0.2.5        dst_address:255.255.255.255
src_address:10.0.2.5        dst_address:224.0.0.251
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
^C
```

```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
[sudo] password for seed:
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:61.129.42.6
src_address:10.0.2.4        dst_address:202.120.224.26
src_address:10.0.2.4        dst_address:202.120.224.6
src_address:61.129.42.6        dst_address:10.0.2.4
src_address:202.120.224.26        dst_address:10.0.2.4
src_address:202.120.224.6       dst_address:10.0.2.4
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:10.0.2.4        dst_address:128.230.208.76
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:128.230.208.76        dst_address:10.0.2.4
src_address:10.0.2.4        dst_address:128.230.208.76
^C
```

Task 2.1B: Writing Filters

ICMP:使用 BPF "icmp and host 115.239.210.27 and 10.0.2.4"来过滤，（代码：task2.1B_filter_ICMP.c）,结果如下图：

```
[10/04/18]seed@VM:~$ ping 115.239.210.27
PING 115.239.210.27 (115.239.210.27) 56(84) bytes of data.
64 bytes from 115.239.210.27: icmp_seq=1 ttl=42 time=34.9 ms
64 bytes from 115.239.210.27: icmp_seq=2 ttl=42 time=40.2 ms
64 bytes from 115.239.210.27: icmp_seq=3 ttl=42 time=34.4 ms
64 bytes from 115.239.210.27: icmp_seq=4 ttl=42 time=33.7 ms
64 bytes from 115.239.210.27: icmp_seq=5 ttl=42 time=36.3 ms
64 bytes from 115.239.210.27: icmp_seq=6 ttl=42 time=36.9 ms
64 bytes from 115.239.210.27: icmp_seq=7 ttl=42 time=35.6 ms
64 bytes from 115.239.210.27: icmp_seq=8 ttl=42 time=35.1 ms
64 bytes from 115.239.210.27: icmp_seq=9 ttl=42 time=36.8 ms
64 bytes from 115.239.210.27: icmp_seq=10 ttl=42 time=37.4 ms
64 bytes from 115.239.210.27: icmp_seq=11 ttl=42 time=43.7 ms
```

```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
src_address:10.0.2.4      dst_address:115.239.210.27      proto number:1
src_address:115.239.210.27      dst_address:10.0.2.4      proto number:1
```

TCP :使用"tcp dst portrange 10-100"来过滤，（代码：task2.1B_filter_TCP.c）结果如图：

```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:23
dst port:80
dst port:80
dst port:80
dst port:80
dst port:80
dst port:80
dst port:80
dst port:80
dst port:80
```

Task 2.1C: Sniffing Passwords.

通过打印所抓的包，我们可以看到每次写入是，请求连接后输入命令后会有回显，但是密码没有回显，通过打印 telnet 的 data，我们可以得到敏感信息，包括登录账号和密码：如下图密码：dees，账号：seed

```
[10/04/18]seed@VM:~/.../实验报告$ gcc -o sniff task2.1C.c -lpcap
[10/04/18]seed@VM:~/.../实验报告$ sudo ./sniff
                                                                                
                                                                                
Ubuntu 16.04.5 LTS
VM login: ss@ee@ee@dd
@assword: d@e@e@s
@Last login: Thu Oct  4 05:45:01 EDT 2018 from 10.0.2.4 on pts/17@@
@Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-36-generic i686)@@

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

0 packages can be updated.@
0 updates are security updates.
```

Task 2.2: Spoofing

Task 2.2A: Write a spoofing program.

```
    1 2018-10-0… 10.0.2.4    56 10.0.2.5    TCP    80 → 80 [SYN] Seq=1 Win=32767 Len=0
    2 2018-10-0… 10.0.2.4    56 10.0.2.5    TCP    [TCP Retransmission] 80 → 80 [SYN] Seq=1 Win=32767 Len=0
    3 2018-10-0… 10.0.2.4    56 10.0.2.5    TCP    [TCP Retransmission] 80 → 80 [SYN] Seq=1 Win=32767 Len=0
```

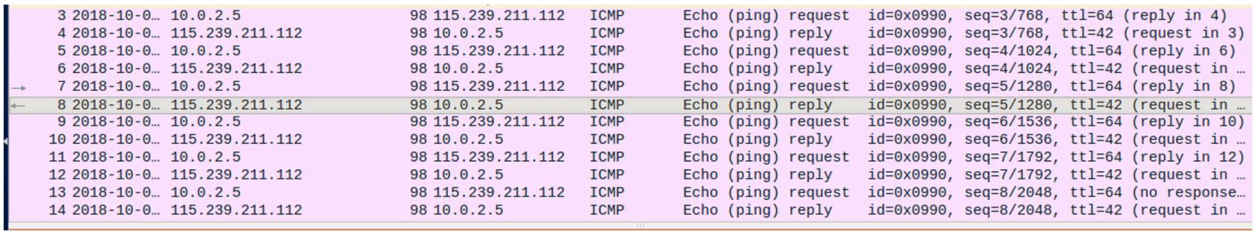
```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./task 10.0.2.4 80 10.0.2.5 80
socket()-SOCK_RAW and tcp protocol is OK.
84017162
setsockopt() is OK
Using:::::Source IP: 10.0.2.4 port: 80, Target IP: 10.0.2.5 port: 80.
Count #0 - sendto() is OK
Count #1 - sendto() is OK
Count #2 - sendto() is OK
Count #3 - sendto() is OK
Count #4 - sendto() is OK
Count #5 - sendto() is OK
Count #6 - sendto() is OK
```

Task 2.2B: Spoof an ICMP Echo Request.

发送报文:

```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./task 10.0.2.5 115.239.211.112
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
Using:::::Source IP: 10.0.2.5 , Target IP: 115.239.211.112 .
Count #0 - sendto() is OK
Count #1 - sendto() is OK
Count #2 - sendto() is OK
Count #3 - sendto() is OK
Count #4 - sendto() is OK
Count #5 - sendto() is OK
Count #6 - sendto() is OK
Count #7 - sendto() is OK
Count #8 - sendto() is OK
Count #9 - sendto() is OK
[10/04/18]seed@VM:~/.../实验报告$
```

收到报文：



3	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=3/768, ttl=64 (reply in 4)
4	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=3/768, ttl=42 (request in 3)
5	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=4/1024, ttl=64 (reply in 6)
6	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=4/1024, ttl=42 (request in …
7	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=5/1280, ttl=64 (reply in 8)
8	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=5/1280, ttl=42 (request in …
9	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=6/1536, ttl=64 (reply in 10)
10	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=6/1536, ttl=42 (request in …
11	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=7/1792, ttl=64 (reply in 12)
12	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=7/1792, ttl=42 (request in …
13	2018-10-0…	10.0.2.5	98	115.239.211.112	ICMP	Echo (ping) request id=0x0990, seq=8/2048, ttl=64 (no response…
14	2018-10-0…	115.239.211.112	98	10.0.2.5	ICMP	Echo (ping) reply id=0x0990, seq=8/2048, ttl=42 (request in …

Questions. Please answer the following questions.

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

不可以，因为当长度太长或与实际不符合是，系统会认为报文已经错误，不会把它继续传下去。

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

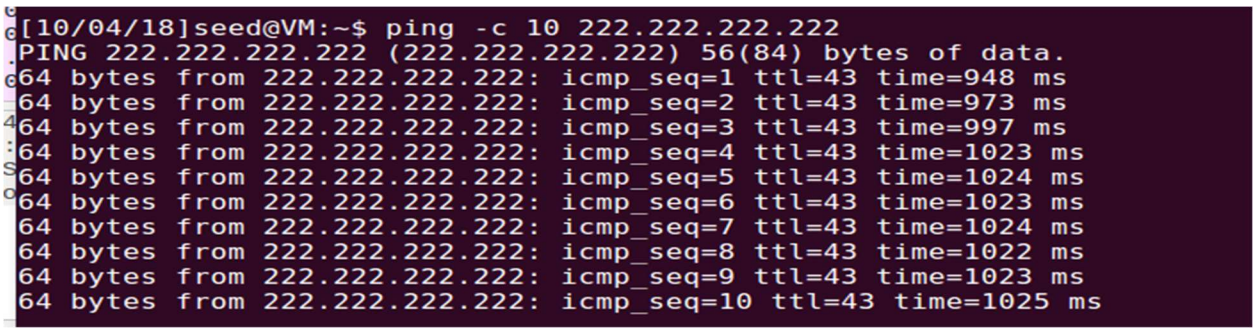
需要 ICMP 的校验和，我并不需要计算 IPheader 的校验和。

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

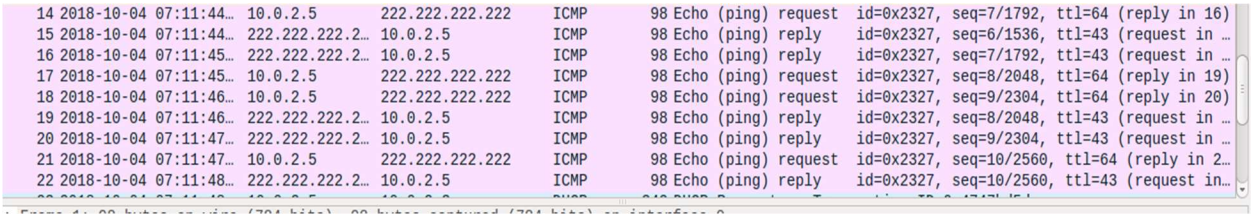
因为我们需要直接访问网卡，直接发送报文，而不是应用程序。并且 raw_socket 规则我们可以任意写，如果不用 root 权限，将会是我们的网络无法正常运转。且一般低于 1024 的端口号，如果没有 root 权限是无法绑定的。

Task2.3: Sniff and then Spoof

程序代码在 task2.3.c 中，我用 VMB ping 222.222.222.222 这个没有注册的 IP 地址，可以发现它能够正常回复显示此正常地址，说明我们伪造的报文是正确的：



```
[10/04/18]seed@VM:~$ ping -c 10 222.222.222.222
PING 222.222.222.222 (222.222.222.222) 56(84) bytes of data.
64 bytes from 222.222.222.222: icmp_seq=1 ttl=43 time=948 ms
64 bytes from 222.222.222.222: icmp_seq=2 ttl=43 time=973 ms
64 bytes from 222.222.222.222: icmp_seq=3 ttl=43 time=997 ms
64 bytes from 222.222.222.222: icmp_seq=4 ttl=43 time=1023 ms
64 bytes from 222.222.222.222: icmp_seq=5 ttl=43 time=1024 ms
64 bytes from 222.222.222.222: icmp_seq=6 ttl=43 time=1023 ms
64 bytes from 222.222.222.222: icmp_seq=7 ttl=43 time=1024 ms
64 bytes from 222.222.222.222: icmp_seq=8 ttl=43 time=1022 ms
64 bytes from 222.222.222.222: icmp_seq=9 ttl=43 time=1023 ms
64 bytes from 222.222.222.222: icmp_seq=10 ttl=43 time=1025 ms
```



14	2018-10-04 07:11:44…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x2327, seq=7/1792, ttl=64 (reply in 16)
15	2018-10-04 07:11:44…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x2327, seq=6/1536, ttl=43 (request in …
16	2018-10-04 07:11:45…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x2327, seq=7/1792, ttl=43 (request in …
17	2018-10-04 07:11:45…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x2327, seq=8/2048, ttl=64 (reply in 19)
18	2018-10-04 07:11:46…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x2327, seq=9/2304, ttl=64 (reply in 20)
19	2018-10-04 07:11:46…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x2327, seq=8/2048, ttl=43 (request in …
20	2018-10-04 07:11:47…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x2327, seq=9/2304, ttl=43 (request in …
21	2018-10-04 07:11:47…	10.0.2.5	222.222.222.222	ICMP	98	Echo (ping) request id=0x2327, seq=10/2560, ttl=64 (reply in 2…
22	2018-10-04 07:11:48…	222.222.222.2…	10.0.2.5	ICMP	98	Echo (ping) reply id=0x2327, seq=10/2560, ttl=43 (request in…

发送信息:

```
[10/04/18]seed@VM:~/.../实验报告$ sudo ./task
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
pacek_len :54 5400
Count # - sendto() is OK
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
pacek_len :54 5400
Count # - sendto() is OK
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
pacek_len :54 5400
Count # - sendto() is OK
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
pacek_len :54 5400
Count # - sendto() is OK
socket()-SOCK_RAW and icmp protocol is OK.
setsockopt() is OK
pacek_len :54 5400
Count # - sendto() is OK
socket()-SOCK_RAW and icmp protocol is OK.
```