山东大学

网络空间安全学院

网络安全实验

Packet Sniffing and Spoofing Lab

学号	姓名	任务分工
202100460095	杜威	代码编写，资源汇总
202100460124	李旷达	环境配置，程序调试
202100460134	杨昊	程序修改，总结汇总

目录

实验目的：

1.嗅探和欺骗工具的使用。
2.使用 pcap 和 Scapy 进行数据包嗅探。
3.使用原始套接字和 Scapy 的数据包欺骗。
4.使用 Scapy 操作数据包。
5.编写程序进行嗅探和欺骗。

实验任务：

任务一：
Lab Task Set 1: Using Scapy to Sniff and Spoof Packets.
(使用工具进行数据包嗅探和欺骗。)
Task 1.1: Sniffing Packets.
Task 1.2: Spoofing ICMP Packets.
Task 1.3: Traceroute.
Task 1.4: Sniffing and-then Spoofing.
任务二：
Lab Task Set 2: Writing Programs to Sniff and Spoof Packets.
(编写程序进行数据包嗅探和欺骗。)
Task 2.1: Writing Packet Sniffing Program.
Task 2.2: Spoofing.
Taks 2.3: Sniff and then Spoof.

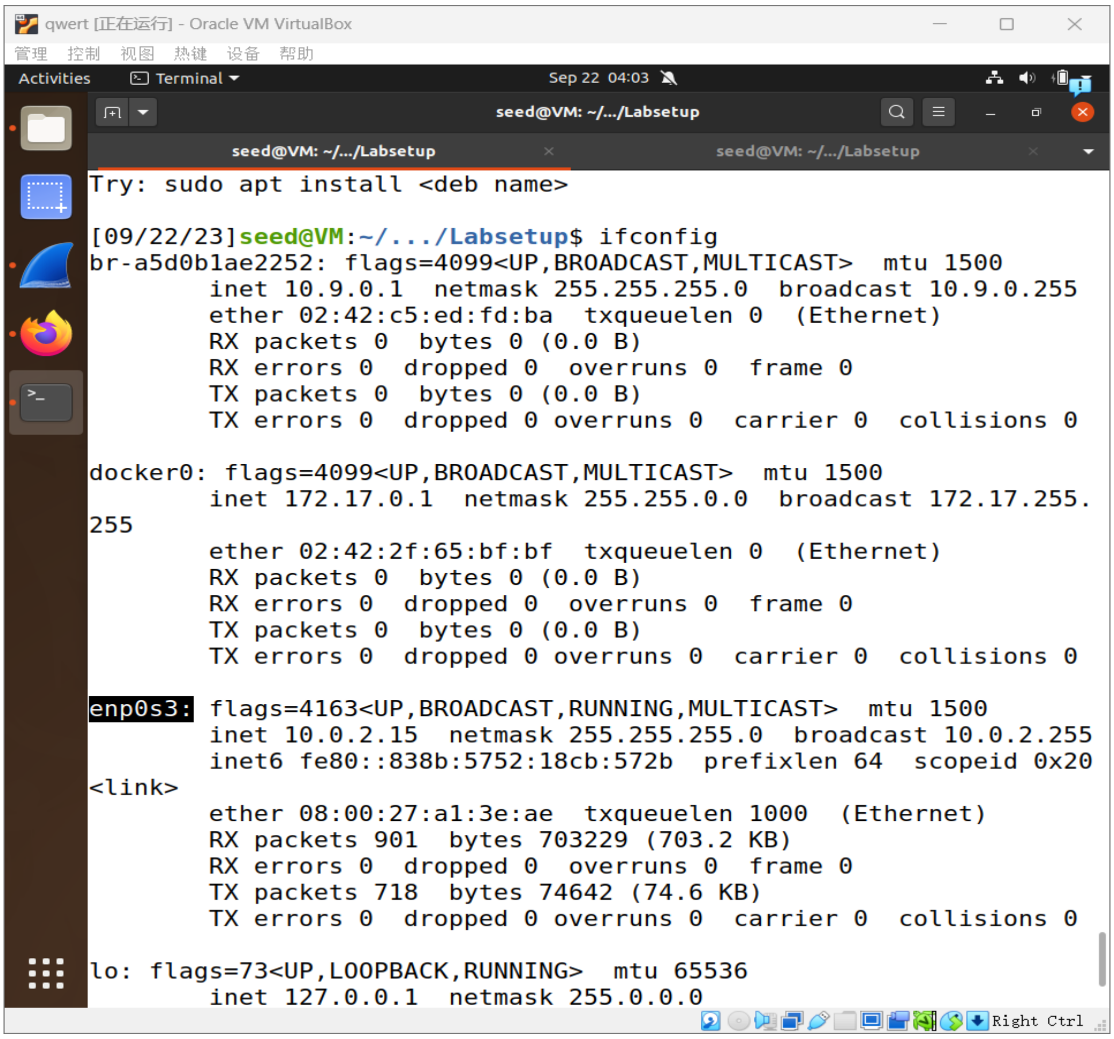
实验内容：

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

Task 1.1: Sniffing Packets
Task 1.1A
注：Scapy 对比于 Wireshark，更容易作为模块构建其他嗅探和欺骗工具。
编写嗅探程序，并使用管理员权限运行：

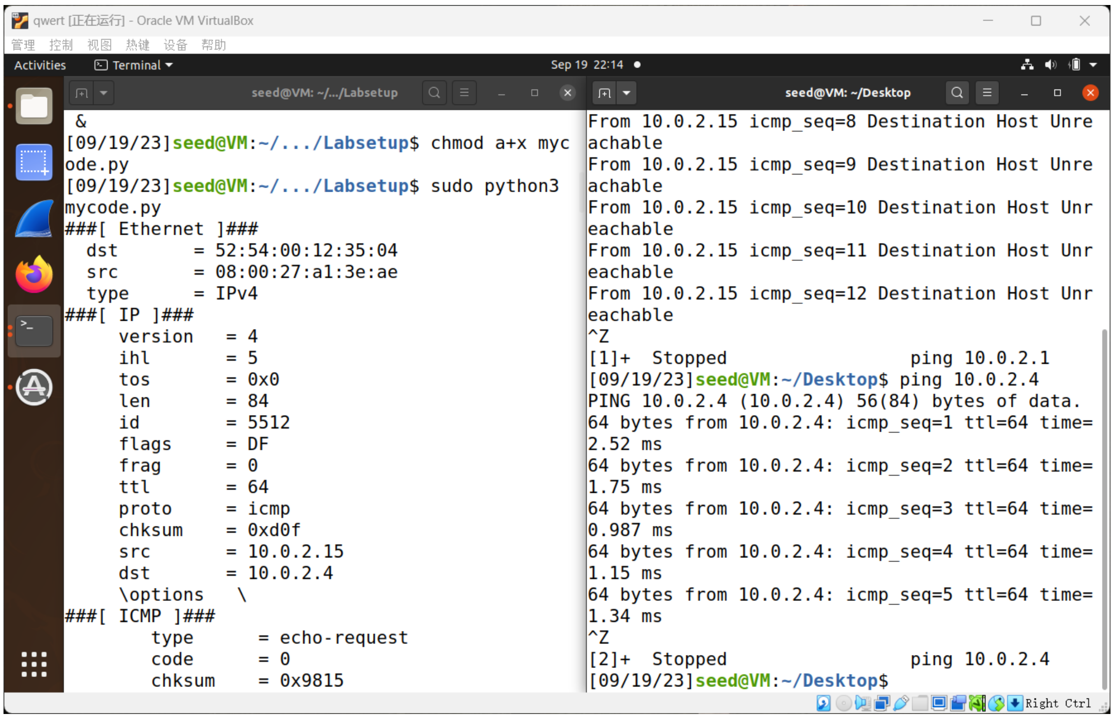
```python
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface=['enp0s3','bra5d0b1ae2252'], filter='icmp', prn
=print_pkt)
```

下图查询本机 IP 地址及网卡。



```
Try: sudo apt install <deb name>

[09/22/23]seed@VM:~/.../Labsetup$ ifconfig
br-a5d0b1ae2252: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:c5:ed:fd:ba  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.
255
        ether 02:42:2f:65:bf:bf  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::838b:5752:18cb:572b  prefixlen 64  scopeid 0x20
<link>
        ether 08:00:27:a1:3e:ae  txqueuelen 1000  (Ethernet)
        RX packets 901  bytes 703229 (703.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 718  bytes 74642 (74.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
```

下面输入命令，首先运行嗅探程序。接着手动 PING 一个地址，这里以 10.0.2.4 为例，可以看到，下面出现了截获的数据包。



```
&
[09/19/23]seed@VM:~/.../Labsetup$ chmod a+x myc
ode.py
[09/19/23]seed@VM:~/.../Labsetup$ sudo python3
mycode.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:04
  src       = 08:00:27:a1:3e:ae
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 5512
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xd0f
     src       = 10.0.2.15
     dst       = 10.0.2.4
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x9815
```

```
From 10.0.2.15 icmp_seq=8 Destination Host Unre
achable
From 10.0.2.15 icmp_seq=9 Destination Host Unre
achable
From 10.0.2.15 icmp_seq=10 Destination Host Unr
eachable
From 10.0.2.15 icmp_seq=11 Destination Host Unr
eachable
From 10.0.2.15 icmp_seq=12 Destination Host Unr
eachable
^Z
[1]+  Stopped                 ping 10.0.2.1
[09/19/23]seed@VM:~/Desktop$ ping 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=
2.52 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=
1.75 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=
0.987 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=
1.15 ms
64 bytes from 10.0.2.4: icmp_seq=5 ttl=64 time=
1.34 ms
^Z
[2]+  Stopped                 ping 10.0.2.4
[09/19/23]seed@VM:~/Desktop$
```

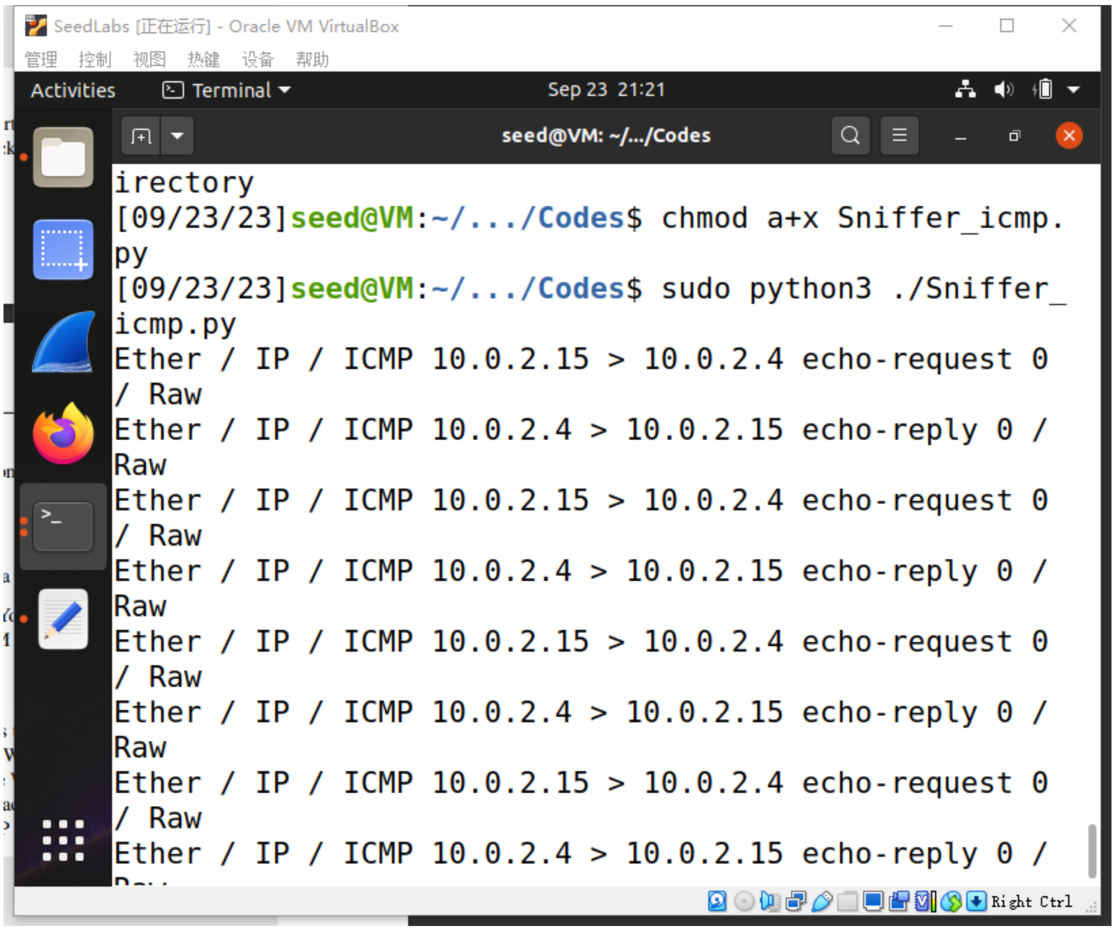
另外，该程序不能由普通用户权限执行，必须使用 ROOT 权限才能运行嗅探程序，可以得知嗅探这一行为只能由 ROOT 权限才能进行。

Task 1.1B

简单修改嗅探条件即可只嗅探 ICMP 数据包。

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

def print_pkt(pkt):
return pkt.summary()

pkt=sniff(filter="icmp",prn=print_pkt)
print(pkt)
```

手动 PING 一个地址，下面就是截获的 ICMP 数据包。


SeedLabs [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities
Terminal
Sep 23 21:21
seed@VM: ~/.../Codes
Right Ctrl

```
irectory
[09/23/23]seed@VM:~/.../Codes$ chmod a+x Sniffer_icmp.
py
[09/23/23]seed@VM:~/.../Codes$ sudo python3 ./Sniffer_
icmp.py
Ether / IP / ICMP 10.0.2.15 > 10.0.2.4 echo-request 0
/ Raw
Ether / IP / ICMP 10.0.2.4 > 10.0.2.15 echo-reply 0 /
Raw
Ether / IP / ICMP 10.0.2.15 > 10.0.2.4 echo-request 0
/ Raw
Ether / IP / ICMP 10.0.2.4 > 10.0.2.15 echo-reply 0 /
Raw
Ether / IP / ICMP 10.0.2.15 > 10.0.2.4 echo-request 0
/ Raw
Ether / IP / ICMP 10.0.2.4 > 10.0.2.15 echo-reply 0 /
Raw
Ether / IP / ICMP 10.0.2.15 > 10.0.2.4 echo-request 0
/ Raw
Ether / IP / ICMP 10.0.2.4 > 10.0.2.15 echo-reply 0 /
```

嗅探指定端口的数据包。

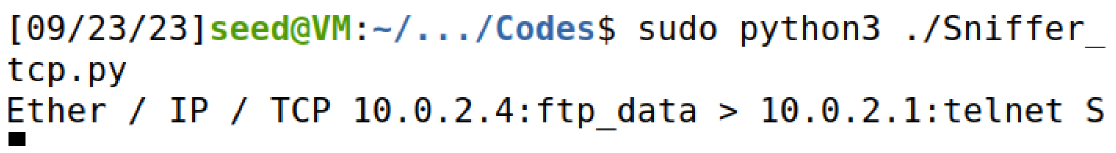
```python
#!usr/bin/python3
from scapy.all import *
```

```python

def print_pkt(pkt):
    return pkt.summary()

pkt=sniff(filter="tcp and src host 10.0.2.4 and dst port 23",prn=print_pkt)
print(pkt)
```

发送指定 TCP 端口的数据包。

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

ip=IP()
ip.src="10.0.2.4"
ip.dst="10.0.2.1"
tcp=TCP()
tcp.dport=23

send(ip/tcp)
```

下面先运行嗅探程序再运行发送程序，得到截获的数据包。



```
[09/23/23]seed@VM:~/.../Codes$ sudo python3 ./Sniffer_tcp.py
Ether / IP / TCP 10.0.2.4:ftp_data > 10.0.2.1:telnet S
```

嗅探任意子网的数据包。

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

def print_pkt(pkt):
    return pkt.summary()

pkt=sniff(filter="net 192.168.0.0/16 ",prn=print_pkt)
print(pkt)
```

发送到指定子网的数据包。

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

```

ip=IP()
ip.src="10.0.2.4"
ip.dst="192.168.0.1"
tcp=TCP()
tcp.dport=23

send(ip/tcp)
```

下面先运行嗅探程序再运行发送程序，得到截获的数据包。

```
[09/23/23]seed@VM:~/.../Codes$ sudo python3 ./Sniffer_
tcp2.py
Ether / IP / TCP 10.0.2.4:ftp_data > 192.168.0.1:telne
t S
```

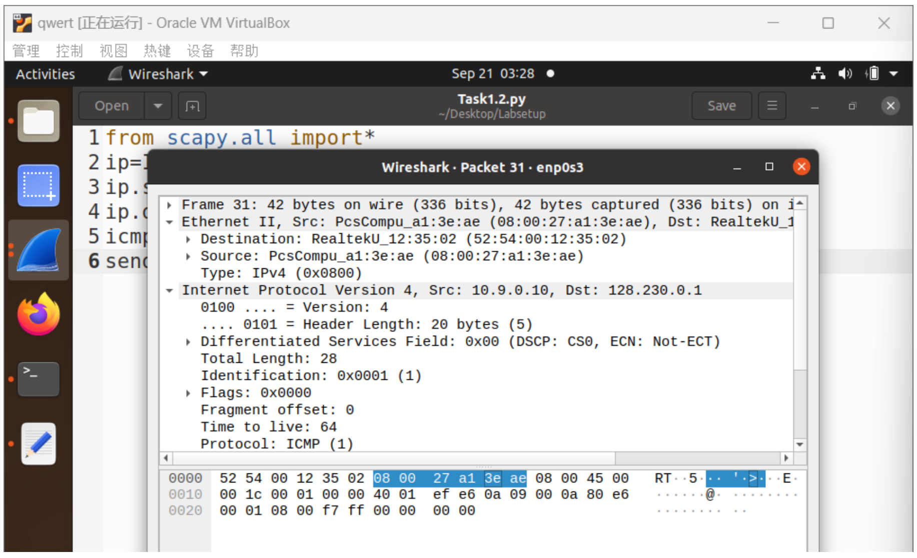
Task 1.2: Spoofing ICMP Packets

利用 Scapy 能设置 IP 数据包字段为任意值的功能，利用源 IP 地址欺骗数据包。我们将欺骗 IPCMP 回 应请求数据包，并将它们发送到同一网络上的另一个虚拟机。我们将使用 Wireshark 观察我们的请 求是否会被接收者接受，如果他被接受，一个应答包将被发送到欺骗的 IP 地址。

使用 Wireshark 抓包，我们需要的是 IP/ICMP 协议用于跟踪路径，做以修改。

```python
from scapy.all import *

ip=IP()
ip.src='10.9.0.10'
ip.dst='128.230.0.1'
icmp=ICMP()

send(ip/icmp)
```

抓包得到下图。


qwert [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities
Wireshark
Sep 21 03:28
Open
Task1.2.py
~/Desktop/Labsetup
Save
Wireshark · Packet 31 · enp0s3

```
from scapy.all import*
ip=I
ip.s
ip.d
icmp
send
```

```
▸ Frame 31: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on i
▾ Ethernet II, Src: PcsCompu_a1:3e:ae (08:00:27:a1:3e:ae), Dst: RealtekU_1
  ▸ Destination: RealtekU_12:35:02 (52:54:00:12:35:02)
  ▸ Source: PcsCompu_a1:3e:ae (08:00:27:a1:3e:ae)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 10.9.0.10, Dst: 128.230.0.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 28
    Identification: 0x0001 (1)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)

0000  52 54 00 12 35 02 08 00  27 a1 3e ae 08 00 45 00   RT··5··· '·>···E·
0010  00 1c 00 01 00 00 40 01  ef e6 0a 09 00 0a 80 e6   ······@· ········
0020  00 01 08 00 f7 ff 00 00  00 00                     ········ ··
```

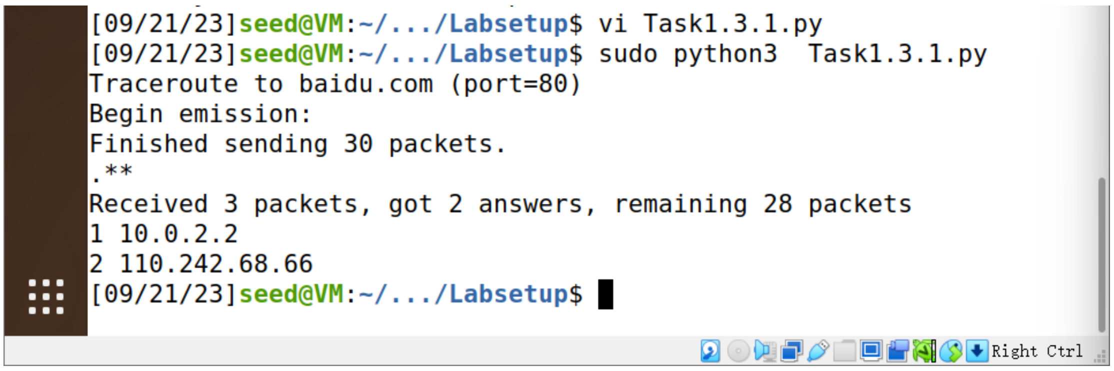
Task 1.3: Traceroute

使用 Scapy 来估计 VM 和你选的目标地址之间的距离即路由器数量，可以使用工具 traceroute 实现。

编写工具（原理）：发送一个数据包（任何类型）到目的地，首先将时间(TTL)字段设置为 1。发送到第一个路由器后 TTL 值-1，被丢弃，并发送给我们一个 ICMP 错误的信息，表示它已经超过运行时间，这样我们就得到了第一个路由器的 IP 地址。同理，我们将 TTL 字段增加到 2，发送另一个数据包，并获得第二个路由器的 IP 地址。我们将重复此过程，直到我们的数据包最终到达目的地时就得到了沿途的路由器地址和源和目的的距离。需要注意的是，这个实验只得到一个估计结果，因为理论上，不是所有这些包采取相同的路径传播（但实际有可能会在短时间内选择同一路径传播）。

```python
from scapy.all import *
import sys

def traceroute(target, minttl=1, maxttl=30, dport=80):
    print(f"Traceroute to {target} (port={dport})")

    ans, uans = sr(
        IP(dst=target, ttl=(minttl, maxttl), id=RandShort()) / TCP(flags=0x2, dport=dport),
        timeout=10
    )

    for snd, rcv in ans:
        print(snd.ttl, rcv.src)

if __name__ == '__main__':
    if len(sys.argv) <= 1:
        traceroute("baidu.com")
    else:
        traceroute(sys.argv[1])
```

下面是运行的结果，我们输出了收到回复的地址，显然是经过了两个路由器。

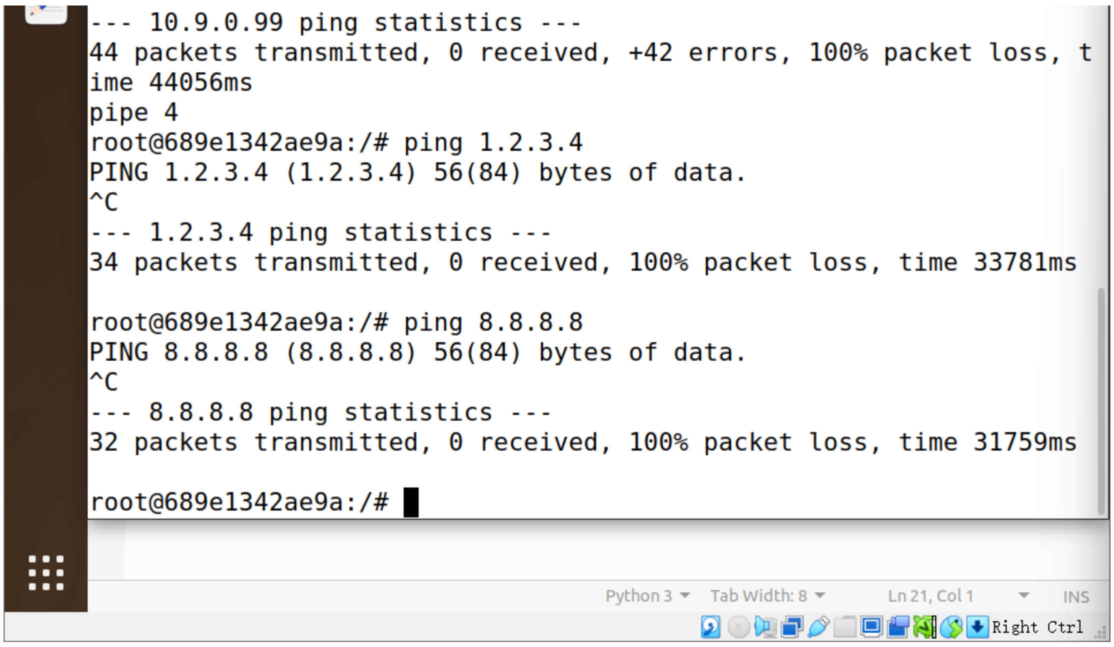


```
[09/21/23]seed@VM:~/.../Labsetup$ vi Task1.3.1.py
[09/21/23]seed@VM:~/.../Labsetup$ sudo python3  Task1.3.1.py
Traceroute to baidu.com (port=80)
Begin emission:
Finished sending 30 packets.
.**
Received 3 packets, got 2 answers, remaining 28 packets
1 10.0.2.2
2 110.242.68.66
[09/21/23]seed@VM:~/.../Labsetup$ 
```

Task 1.4: Sniffing and-then Spoofing

嗅探并欺骗的程序。

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

def spoofing(pkt):
    if pkt[ICMP].type == 8:
        dst = pkt[IP].dst
        src = pkt[IP].src
        ihl = pkt[IP].ihl
        idd = pkt[ICMP].id
        seqq = pkt[ICMP].seq
        load = pkt[Raw].load

        a = IP(src=dst, dst=src, ihl=ihl)
        b = ICMP(type=0, id=idd, seq=seqq)
        c = load

        ans = a / b / c
        send(ans)

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



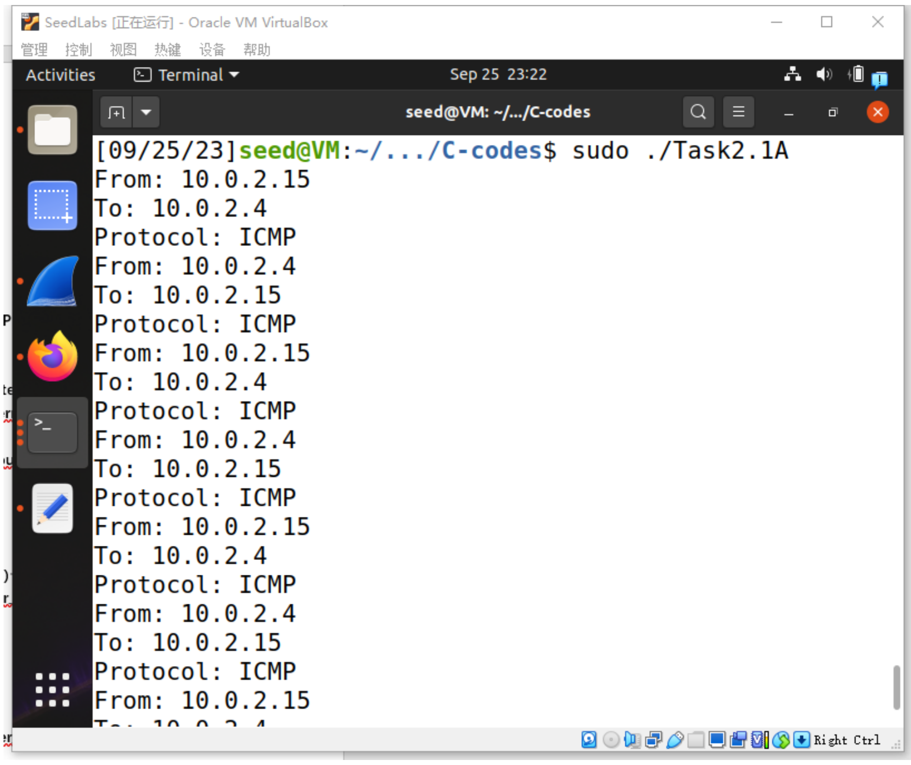
```
--- 10.9.0.99 ping statistics ---
44 packets transmitted, 0 received, +42 errors, 100% packet loss, t
ime 44056ms
pipe 4
root@689e1342ae9a:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
34 packets transmitted, 0 received, 100% packet loss, time 33781ms

root@689e1342ae9a:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
32 packets transmitted, 0 received, 100% packet loss, time 31759ms

root@689e1342ae9a:/# 
```

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

Task 2.1: Writing Packet Sniffing Program

Task 2.1A: Understanding How a Sniffer Works

使用 pcap 库可以轻松编写嗅探器程序。有了 pcap，嗅探器的任务变得简单起来。调用 pcap 库中的一个简单过程序列。在序列的末尾，数据包将在被捕获后立即放入缓冲区进行进一步处理。数据包捕获的所有细节都由 pcap 库处理。
我们使用 C 语言进行程序编写， 代码较长见附录。
使用 GCC 编译并执行之后见下图得到的结果， 可以发现嗅探程序成功截获了数据包。



SeedLabs [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities
Terminal
Sep 25 23:22
seed@VM: ~/.../C-codes
Right Ctrl

```
[09/25/23]seed@VM:~/.../C-codes$ sudo ./Task2.1A
From: 10.0.2.15
To: 10.0.2.4
Protocol: ICMP
From: 10.0.2.4
To: 10.0.2.15
Protocol: ICMP
From: 10.0.2.15
To: 10.0.2.4
Protocol: ICMP
From: 10.0.2.4
To: 10.0.2.15
Protocol: ICMP
From: 10.0.2.15
To: 10.0.2.4
Protocol: ICMP
From: 10.0.2.4
To: 10.0.2.15
Protocol: ICMP
From: 10.0.2.15
```

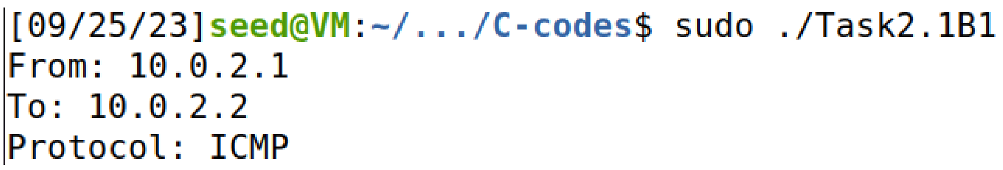
• Question 1. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.
Answer: 创建套接字，设置套接字选项，将套接字绑定到指定的网络接口上并设置过滤器，指定源或目的 IP 地址、端口号等，进行循环捕获，关闭嗅探。

• 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?
Answer: 因为在系统设定中，对于访问网络数据包有较高的限制，因此要进行此操作需要更高的 root 权限。

• Question 3. Please turn on and turn off the promiscuous mode in your sniffer program. The value 1 of the third parameter in pcap open live() turns on the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this. You can use the following command to check whether an interface's promiscuous mode is on or off (look at the promiscuity's value).
Answer: 若不启用混杂模式，则嗅探程序只会捕获发送给自己地址的数据包，而在混杂模式下，其可以捕获通过它传递的所有数据包，无论目的地址如何。

Task 2.1B: Writing Filters
修改 2.1A 部分的过滤器来捕获指定的数据包，修改的过滤器如下。

```c
char filter_exp[] = "icmp and src host 10.0.2.1 and dst host 10.0.2.2";
```



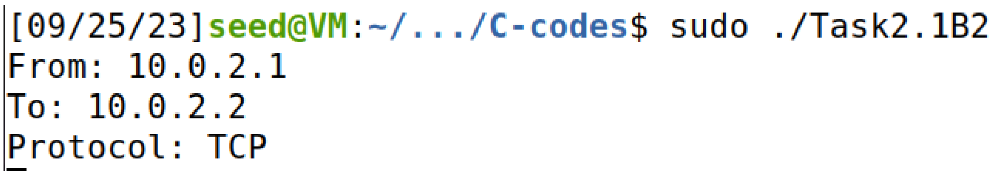
```
[09/25/23]seed@VM:~/.../C-codes$ sudo ./Task2.1B1
From: 10.0.2.1
To: 10.0.2.2
Protocol: ICMP
```

收到了上面截获的报文。
发送者代码如下。

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

ip=IP()
ip.src="10.0.2.1"
ip.dst="10.0.2.2"
icmp=ICMP()

send(ip/icmp)
```

为了截获端口 10-100 的数据包，过滤器修改如下。

```c
char filter_exp[] = "tcp and dst portrange 10-100";
```



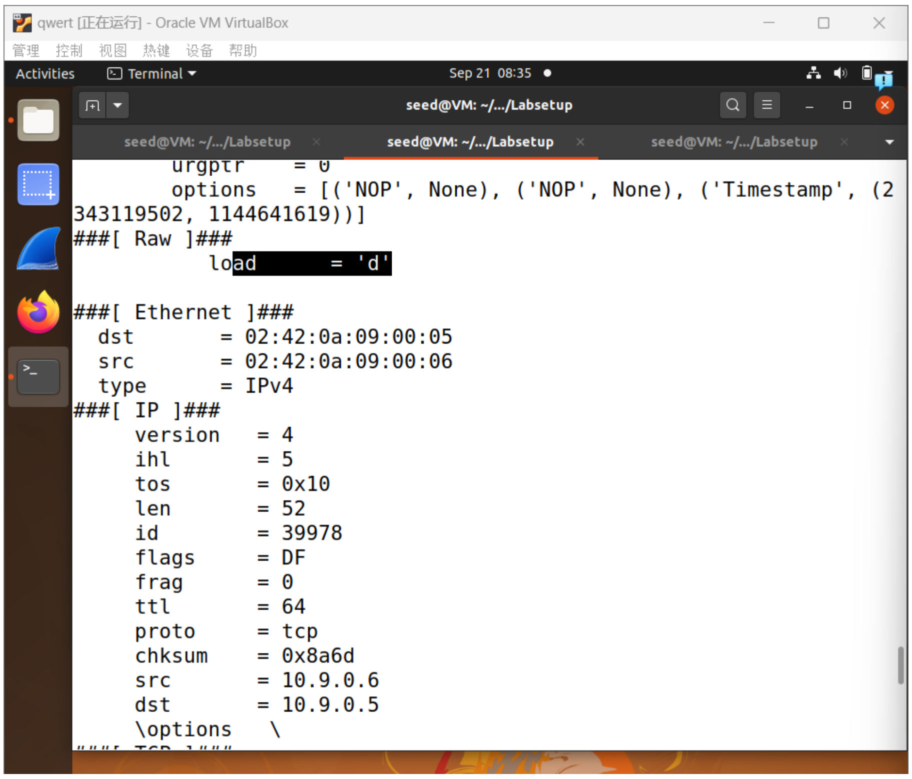
```
[09/25/23]seed@VM:~/.../C-codes$ sudo ./Task2.1B2
From: 10.0.2.1
To: 10.0.2.2
Protocol: TCP
```

收到了上面截获的报文。
发送者代码如下。

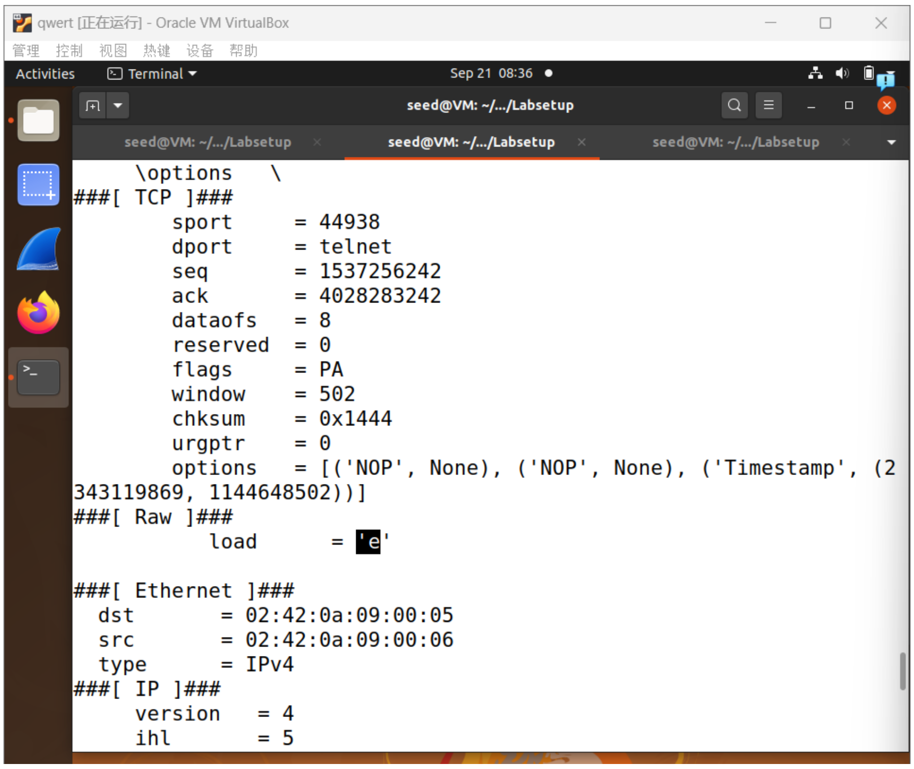
```python
#!usr/bin/python3
from scapy.all import *

ip=IP()
ip.src="10.0.2.1"
ip.dst="10.0.2.2"
tcp=TCP()
tcp.dport=23

send(ip/tcp)
```

Task 2.1C: Sniffing Passwords

代码较长，见附录，执行监听程序，结果如下。



```
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (2
343119502, 1144641619))]
###[ Raw ]###
           load      = 'd'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     id        = 39978
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x8a6d
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
```

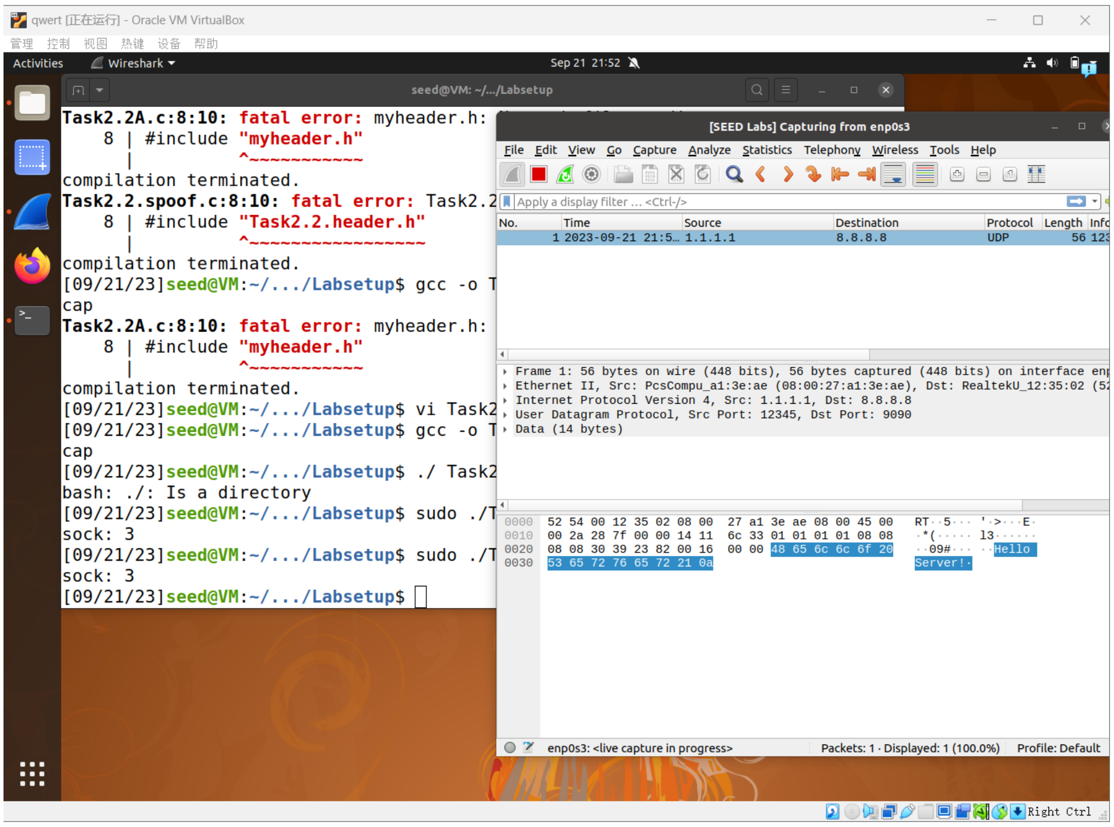


```
     \options   \
###[ TCP ]###
        sport     = 44938
        dport     = telnet
        seq       = 1537256242
        ack       = 4028283242
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 502
        chksum    = 0x1444
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (2
343119869, 1144648502))]
###[ Raw ]###
           load      = 'e'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
```

Task 2.2: Spoofing

Task 2.2A: Write a spoofing program

代码较长，见附录，运行伪造程序，如下图所示，我们从 wireshark 看到了我们发出的伪造数据包。



```
Task2.2A.c:8:10: fatal error: myheader.h:
    8 | #include "myheader.h"
      |          ^~~~~~~~~~~~
compilation terminated.
Task2.2.spoof.c:8:10: fatal error: Task2.2
    8 | #include "Task2.2.header.h"
      |          ^~~~~~~~~~~~~~~~~~
compilation terminated.
[09/21/23]seed@VM:~/.../Labsetup$ gcc -o T
cap
Task2.2A.c:8:10: fatal error: myheader.h:
    8 | #include "myheader.h"
      |          ^~~~~~~~~~~~
compilation terminated.
[09/21/23]seed@VM:~/.../Labsetup$ vi Task2
[09/21/23]seed@VM:~/.../Labsetup$ gcc -o T
cap
[09/21/23]seed@VM:~/.../Labsetup$ ./ Task2
bash: ./: Is a directory
[09/21/23]seed@VM:~/.../Labsetup$ sudo ./T
sock: 3
[09/21/23]seed@VM:~/.../Labsetup$ sudo ./T
sock: 3
[09/21/23]seed@VM:~/.../Labsetup$
```

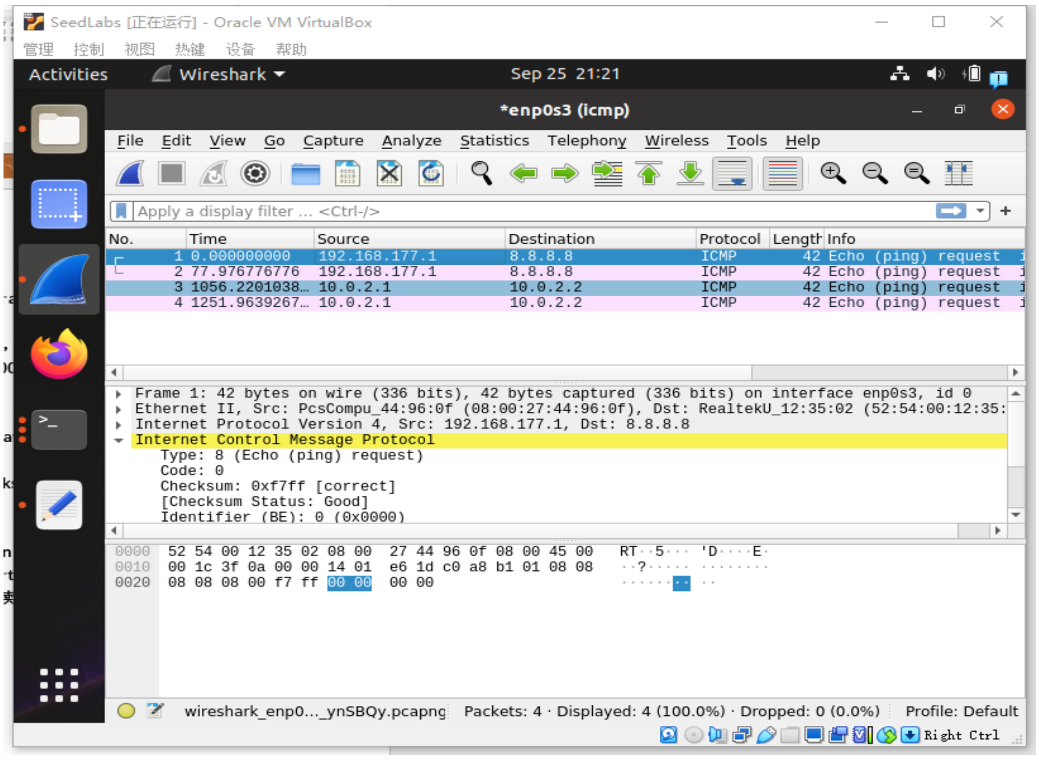
```
No.  Time                 Source    Destination  Protocol  Length  Info
   1 2023-09-21 21:5…     1.1.1.1   8.8.8.8      UDP          56  123

Frame 1: 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface enp
Ethernet II, Src: PcsCompu_a1:3e:ae (08:00:27:a1:3e:ae), Dst: RealtekU_12:35:02 (52
Internet Protocol Version 4, Src: 1.1.1.1, Dst: 8.8.8.8
User Datagram Protocol, Src Port: 12345, Dst Port: 9090
Data (14 bytes)

0000  52 54 00 12 35 02 08 00  27 a1 3e ae 08 00 45 00   RT··5···  '·>···E·
0010  00 2a 28 7f 00 00 14 11  6c 33 01 01 01 01 08 08   ·*(·····  l3······
0020  08 08 30 39 23 82 00 16  00 00 48 65 6c 6c 6f 20   ··09#···  ··Hello 
0030  53 65 72 76 65 72 21 0a                            Server!·

enp0s3: <live capture in progress>   Packets: 1 · Displayed: 1 (100.0%)   Profile: Default
```

Task 2.2B: Spoof an ICMP Echo Request

伪造一个 ICMP 回复数据包。

代码较长见附录。

运行之后通过 wireshark 我们看到了我们发出的伪造数据包。



```
No.  Time             Source          Destination  Protocol Length Info
   1 0.000000000      192.168.177.1   8.8.8.8      ICMP        42 Echo (ping) request
   2 77.976776776     192.168.177.1   8.8.8.8      ICMP        42 Echo (ping) request
   3 1056.2201038…    10.0.2.1        10.0.2.2     ICMP        42 Echo (ping) request
   4 1251.9639267…    10.0.2.1        10.0.2.2     ICMP        42 Echo (ping) request
```

```
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_44:96:0f (08:00:27:44:96:0f), Dst: RealtekU_12:35:02 (52:54:00:12:35:
Internet Protocol Version 4, Src: 192.168.177.1, Dst: 8.8.8.8
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xf7ff [correct]
    [Checksum Status: Good]
    Identifier (BE): 0 (0x0000)
```

```
0000  52 54 00 12 35 02 08 00  27 44 96 0f 08 00 45 00   RT··5···  'D····E·
0010  00 1c 3f 0a 00 00 14 01  e6 1d c0 a8 b1 01 08 08   ··?·····  ········
0020  08 08 08 00 f7 ff 00 00  00 00                     ········  ··
```

wireshark_enp0…_ynSBQy.pcapng   Packets: 4 · Displayed: 4 (100.0%) · Dropped: 0 (0.0%)   Profile: Default

SeedLabs [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities
Wireshark
Sep 25 21:21
*enp0s3 (icmp)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Right Ctrl

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

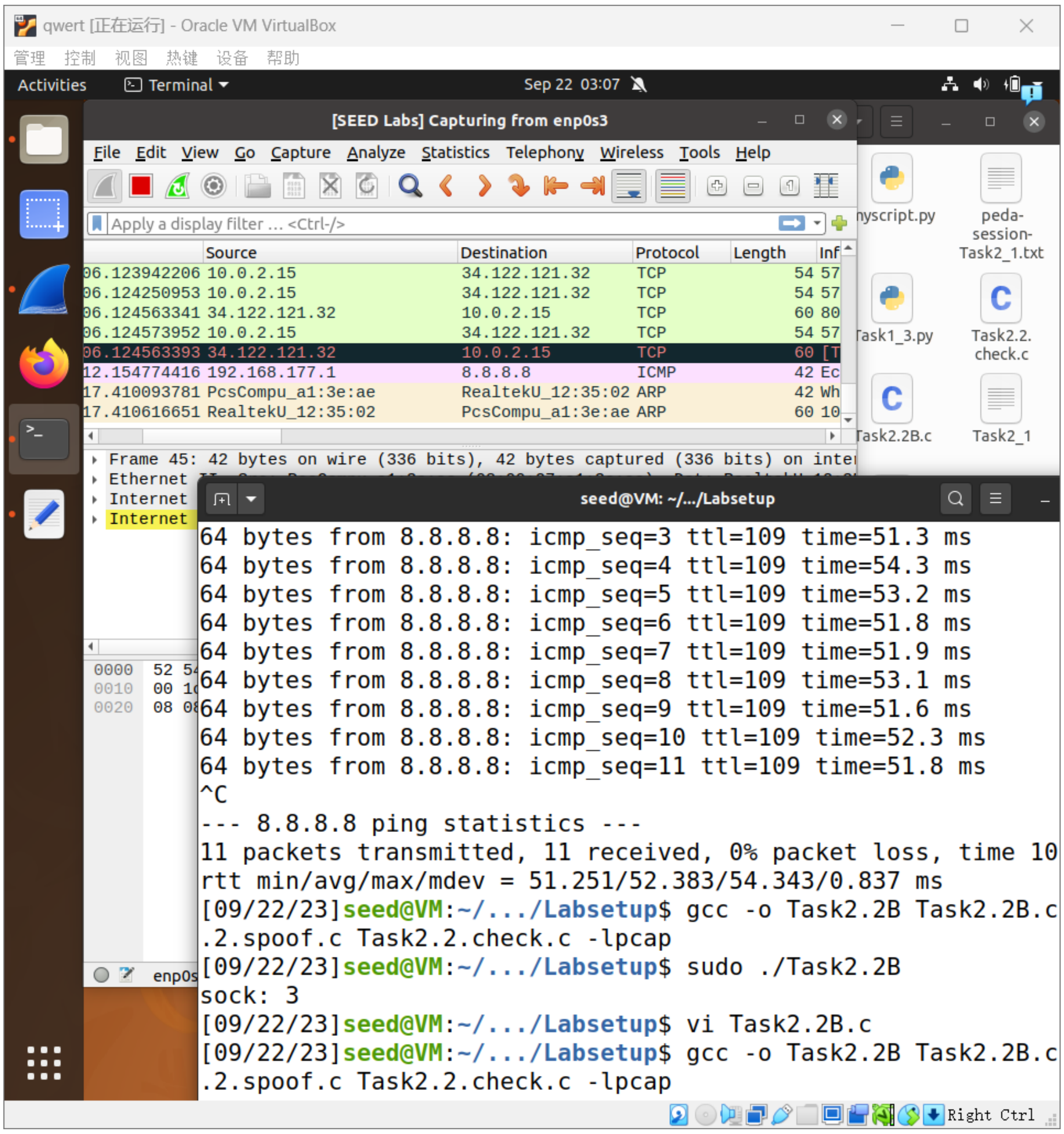
Answer: 将原先代码中的设置长度部分改为设置并打印长度，我们可以得到长度为 28B，我们可以将其 设置为 10B，wireshark 没有抓到，修改为 100B，显示可以抓到并且收到了相应，说明可以调大 length，但是不能调小 length。

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

Answer: 使用 raw socket 编程时，不用计算 IP 头部的 checksum，但是需要计算 ICMP 头部的 checksum。

• 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?

Answer: 使用 raw socket 编程时必须要 root 权限，是因为读取发送包的系统设定所需权限较高。如果没有 root 会报错，如下图所示。


qwert [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities
Terminal
Sep 22 03:07
[SEED Labs] Capturing from enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
myscript.py
peda-session-Task2_1.txt
Task1_3.py
Task2.2.check.c
Task2.2B.c
Task2_1
seed@VM: ~/…/Labsetup
Right Ctrl

```
                   Source                    Destination         Protocol  Length  Inf
06.123942206 10.0.2.15                   34.122.121.32       TCP          54 57
06.124250953 10.0.2.15                   34.122.121.32       TCP          54 57
06.124563341 34.122.121.32               10.0.2.15           TCP          60 80
06.124573952 10.0.2.15                   34.122.121.32       TCP          54 57
06.124563393 34.122.121.32               10.0.2.15           TCP          60 [T
12.154774416 192.168.177.1               8.8.8.8             ICMP         42 Ec
17.410093781 PcsCompu_a1:3e:ae           RealtekU_12:35:02   ARP          42 Wh
17.410616651 RealtekU_12:35:02           PcsCompu_a1:3e:ae   ARP          60 10

▸ Frame 45: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on inte
▸ Ethernet
▸ Internet
▸ Internet

0000   52 54
0010   00 1
0020   08 0
```

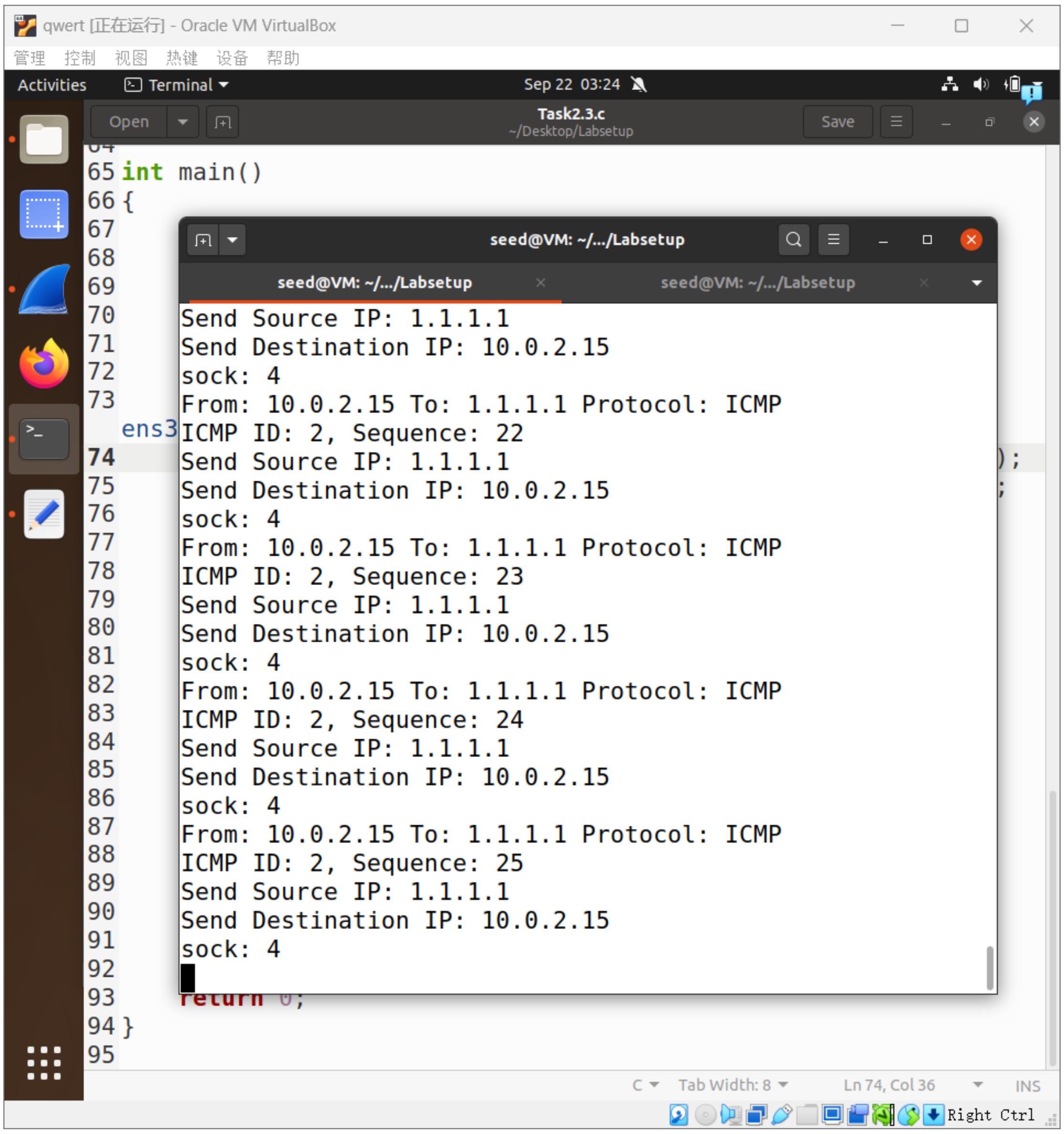
```
64 bytes from 8.8.8.8: icmp_seq=3 ttl=109 time=51.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=109 time=54.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=109 time=53.2 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=109 time=51.8 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=109 time=51.9 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=109 time=53.1 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=109 time=51.6 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=109 time=52.3 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=109 time=51.8 ms
^C
--- 8.8.8.8 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10
rtt min/avg/max/mdev = 51.251/52.383/54.343/0.837 ms
[09/22/23]seed@VM:~/.../Labsetup$ gcc -o Task2.2B Task2.2B.c
.2.spoof.c Task2.2.check.c -lpcap
[09/22/23]seed@VM:~/.../Labsetup$ sudo ./Task2.2B
sock: 3
[09/22/23]seed@VM:~/.../Labsetup$ vi Task2.2B.c
[09/22/23]seed@VM:~/.../Labsetup$ gcc -o Task2.2B Task2.2B.c
.2.spoof.c Task2.2.check.c -lpcap
```

Task 2.3: Sniff and then Spoof

编写程序完成嗅探并伪造数据包，首先监听局域网上的 ICMP 回波请求数据包，截获并用数据包欺骗数据伪造一个回波回复。

代码较长见附录。

程序的运行结果如下。



```
int main()
{
ens3
    return 0;
}
```

```
Send Source IP: 1.1.1.1
Send Destination IP: 10.0.2.15
sock: 4
From: 10.0.2.15 To: 1.1.1.1 Protocol: ICMP
ICMP ID: 2, Sequence: 22
Send Source IP: 1.1.1.1
Send Destination IP: 10.0.2.15
sock: 4
From: 10.0.2.15 To: 1.1.1.1 Protocol: ICMP
ICMP ID: 2, Sequence: 23
Send Source IP: 1.1.1.1
Send Destination IP: 10.0.2.15
sock: 4
From: 10.0.2.15 To: 1.1.1.1 Protocol: ICMP
ICMP ID: 2, Sequence: 24
Send Source IP: 1.1.1.1
Send Destination IP: 10.0.2.15
sock: 4
From: 10.0.2.15 To: 1.1.1.1 Protocol: ICMP
ICMP ID: 2, Sequence: 25
Send Source IP: 1.1.1.1
Send Destination IP: 10.0.2.15
sock: 4
```

附录

Task2.1A

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

struct ip_hdr {
    unsigned char ver_ihl;
    unsigned char tos;
    unsigned short len;
    unsigned short ident;
    unsigned short flags_offset;
    unsigned char ttl;
    unsigned char protocol;
    unsigned short checksum;
    struct in_addr src_ip;
    struct in_addr dest_ip;
};

struct eth_hdr {
    u_char dhost[6];
    u_char shost[6];
    u_short type;
};

void packet_handler(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
    struct eth_hdr *eth = (struct eth_hdr *)packet;
    if (ntohs(eth->type) == 0x0800) {
        struct ip_hdr *ip = (struct ip_hdr *)(packet + sizeof(struct eth_hdr));
        printf("From: %s\n", inet_ntoa(ip->src_ip));
        printf("To: %s\n", inet_ntoa(ip->dest_ip));
        switch(ip->protocol) {
            case 1:
                printf("Protocol: ICMP\n");
                break;
            case 6:
                printf("Protocol: TCP\n");
                break;
            case 17:
                printf("Protocol: UDP\n");
                break;
            default:
                printf("Protocol: Unknown\n");
                break;
        }
        printf("\n");
    }
}

int main() {
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    bpf_u_int32 net;

    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    if (!handle) {
        fprintf(stderr, "Could not open device: %s\n", errbuf);
        return 2;
    }

    pcap_loop(handle, -1, packet_handler, NULL);

    pcap_close(handle);

    return 0;
}
```

Task2.2spoof.c

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

#include "Task2.2.header.h"
void send_raw_ip_packet(struct ipheader* ip) {
    struct sockaddr_in dest_info;
    int enable = 1;

    int raw_socket = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
    printf("raw_socket: %d\n", raw_socket);

    setsockopt(raw_socket, IPPROTO_IP, IP_HDRINCL,
        &enable, sizeof(enable));

    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

    sendto(raw_socket, ip, ntohs(ip->iph_len), 0,
        (struct sockaddr*)&dest_info, sizeof(dest_info));
    close(raw_socket);
}

```

Task2.2header.h

```
/* Ethernet header */
struct ethheader {
    u_char  dhost[6];
    u_char  shost[6];
    u_short type;
};

/* IP Header */
struct ipheader {
    unsigned char      ihl : 4, ver : 4;
    unsigned char      tos;
    unsigned short int len;
    unsigned short int ident;
    unsigned short int flag : 3, offset : 13;
    unsigned char      ttl;
    unsigned char      protocol;
    unsigned short int chksum;
    struct  in_addr    sourceip;
    struct  in_addr    destip;
};

/* ICMP Header */
struct icmpheader {
    unsigned char type;
    unsigned char code;
    unsigned short int chksum;
    unsigned short int id;
    unsigned short int seq;
};
```

```c

/* UDP Header */
struct udpheader {
    u_int16_t sport;
    u_int16_t dport;
    u_int16_t ulen;
    u_int16_t sum;
};

/* TCP Header */
struct tcpheader {
    u_short sport;
    u_short dport;
    u_int   seq;
    u_int   ack;
    u_char  offx2;
#define OFFSET(th)      (((th)->offx2 & 0xf0) >> 4)
    u_char  flags;
#define FIN  0x01
#define SYN  0x02
#define RST  0x04
#define PUSH 0x08
#define ACK  0x10
#define URG  0x20
#define ECE  0x40
#define CWR  0x80
#define FLAGS        (FIN|SYN|RST|ACK|URG|ECE|CWR)
    u_short win;
    u_short sum;
    u_short urp;
};

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

Task2.2A

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

#include "Task2.2.header.h"

// 发送原始 IP 数据包的函数
void send_raw_ip_packet(struct ipheader* ip);

int main() {
    char buffer[1500];

    // 初始化缓冲区
    memset(buffer, 0, 1500);

    // 设置 IP 头部和 UDP 头部的指针
    struct ipheader* ip = (struct ipheader*)buffer;
    struct udpheader* udp = (struct udpheader*)(buffer + sizeof(struct ipheader));

    // 设置数据部分的指针和消息内容
    char* data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
    const char* msg = "Hello Server!\n";
    int data_len = strlen(msg);
    strncpy(data, msg, data_len);

    // 配置 UDP 头部字段
    udp->udp_sport = htons(12345); // 源端口
    udp->udp_dport = htons(9090);  // 目标端口
    udp->udp_ulen = htons(sizeof(struct udpheader) + data_len); // UDP 长度
    udp->udp_sum = 0; // UDP 校验和（0 表示不校验）

    // 配置 IP 头部字段
    ip->iph_ver = 4; // IP 版本
    ip->iph_ihl = 5; // IP 头部长度（以 32 位字为单位）
    ip->iph_ttl = 20; // 存活时间
    ip->iph_sourceip.s_addr = inet_addr("1.1.1.1"); // 源 IP 地址
    ip->iph_destip.s_addr = inet_addr("8.8.8.8");   // 目标 IP 地址
    ip->iph_protocol = IPPROTO_UDP; // 协议类型为 UDP
    ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct udpheader) + data_len); // IP 数据包长度

    // 调用发送原始 IP 数据包的函数
    send_raw_ip_packet(ip);

    return 0;
}

```

Task2.2check

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

#include "Task2.2.header.h"
struct ipheader {
    unsigned char ver_ihl;
    unsigned char tos;
    unsigned short len;
    unsigned short ident;
    unsigned short flags_offset;
    unsigned char ttl;
    unsigned char protocol;
    unsigned short checksum;
    struct in_addr src_ip;
    struct in_addr dest_ip;
};

unsigned short calc_checksum(unsigned short* buf, int length) {
    unsigned int sum = 0;
    while (length > 1) {
        sum += *buf++;
        length -= 2;
    }
    if (length == 1) {
        sum += *(unsigned char*)buf;
```

```c
    }
    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    return (unsigned short)(~sum);
}

unsigned short calc_tcp_checksum(struct ipheader* ip) {
    struct tcpheader* tcp = (struct tcpheader*)((u_char*)ip + sizeof(struct ipheader));
    int tcp_len = ntohs(ip->len) - sizeof(struct ipheader);
    struct pseudo_tcp p_tcp;
    memset(&p_tcp, 0x0, sizeof(struct pseudo_tcp));
    p_tcp.saddr = ip->src_ip.s_addr;
    p_tcp.daddr = ip->dest_ip.s_addr;
    p_tcp.ptcl = ip->protocol;
    p_tcp.tcpl = htons(tcp_len);
    memcpy(&p_tcp.tcp, tcp, tcp_len);
    return (unsigned short)calc_checksum((unsigned short*)&p_tcp, tcp_len + 12);
}
```

Task2.2B

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include "Task2.2.header.h"

unsigned short calc_checksum(unsigned short* buf, int length);
void send_raw_ip_packet(struct ipheader* ip);

int main() {
    char buffer[1500];
    memset(buffer, 0, 1500);

    // 构造 ICMP 头部
    struct icmpheader* icmp = (struct icmpheader*)(buffer + sizeof(struct ipheader));
    icmp->icmp_type = 8;  // ICMP 类型: 8 表示请求, 0 表示回复
    icmp->icmp_chksum = 0;
    icmp->icmp_chksum = calc_checksum((unsigned short*)icmp, sizeof(struct icmpheader));

    // 构造 IP 头部
    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("192.168.177.1");
    ip->iph_destip.s_addr = inet_addr("8.8.8.8");
    ip->iph_protocol = IPPROTO_ICMP;
    ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));

    // 发送伪造的 IP 数据包
    send_raw_ip_packet(ip);

    return 0;
}
```

Task2.3

```c
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <arpa/inet.h>
#include <pcap.h>
#include "Task2.2.header.h"
#include <stdlib.h>

void send_raw_ip_packet(struct ipheader* ip);
unsigned short calc_checksum(unsigned short* buf, int length);

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

    // 检查以太网帧类型是否为 IP
    if (ntohs(eth->ether_type) == 0x0800) {
        struct ipheader* ip = (struct ipheader*)(packet + sizeof(struct ethheader));
        printf("源地址: %s ", inet_ntoa(ip->iph_sourceip));
        printf("目标地址: %s ", inet_ntoa(ip->iph_destip));

        // 检查协议类型是否为 ICMP
        if (ip->iph_protocol == IPPROTO_ICMP)
            printf("协议类型: ICMP\n");
        else
            printf("协议类型: 其他\n");

        // 提取 ICMP 数据包
        struct icmpheader* icmp_pkt = (struct icmpheader*)(packet + sizeof(struct ethheader) + sizeof(struct ipheader));

        if (ip->iph_protocol == IPPROTO_ICMP) {
            char buffer[1500];
            memset(buffer, 0, 1500);

            // 构造 ICMP 请求数据包
            struct icmpheader* icmp = (struct icmpheader*)(buffer + sizeof(struct ipheader));
            icmp->icmp_type = 8;
            icmp->icmp_code = 0;
            icmp->icmp_id = icmp_pkt->icmp_id;
            icmp->icmp_seq = icmp_pkt->icmp_seq;
            printf("ICMP ID: %d, Sequence: %d\n", ntohs(icmp_pkt->icmp_id), ntohs(icmp_pkt->icmp_seq));

            // 计算校验和
            icmp->icmp_chksum = 0;
            icmp->icmp_chksum = calc_checksum((unsigned short*)icmp, sizeof(struct icmpheader));

            // 构造 IP 数据包
            struct ipheader* ipp = (struct ipheader*)buffer;
            ipp->iph_ver = 4;
            ipp->iph_ihl = 5;
            ipp->iph_ttl = 64;
            ipp->iph_sourceip.s_addr = ip->iph_destip.s_addr;
            ipp->iph_destip.s_addr = ip->iph_sourceip.s_addr;
            ipp->iph_protocol = IPPROTO_ICMP;
            ipp->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));

            printf("发送源 IP 地址：%s\n", inet_ntoa(ipp->iph_sourceip));
            printf("发送目标 IP 地址：%s\n", inet_ntoa(ipp->iph_destip));

            // 发送伪造的 IP 数据包
            send_raw_ip_packet(ipp);
        }
    }
}

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

    // 打开指定网卡的 pcap 会话
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    printf("监听网络接口，返回值：%p...\n", handle);

    printf("尝试编译过滤规则...\n");
    pcap_compile(handle, &fp, filter_exp, 0, net);

    if (pcap_setfilter(handle, &fp) != 0) {
        pcap_perror(handle, "错误:");
        exit(EXIT_FAILURE);
    }

    printf("尝试设置过滤规则...\n");
    pcap_setfilter(handle, &fp);

    printf("开始嗅探...\n");
    pcap_loop(handle, -1, process_packet, NULL);
    pcap_close(handle);
    return 0;
}
```