信息安全实验报告

Lab 13　Packet Sniffing and Spoofing Lab

孙铁

SA20225414

Task 1

安装 Scapy：

```
[07/04/21]seed@VM:~/Lab13$ sudo pip3 install scapy
```

创建文件 mycode.py：

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

a = IP()
a.show()
```

代码定义了一个 IP 报文，并用 show 函数输出了报文相关信息。

运行 mycode.py：

```
[07/04/21]seed@VM:~/Lab13$ sudo ./mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

Task 1.1

创建文件 sniffer.py：

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

def print_pkt(pkt):
        pkt.show()

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

代码的作用是嗅探 ICMP 报文并打印出一些报文的信息。

Task 1.1A

为了产生 ICMP 报文，向 www.baidu.com 使用 PING 指令：

```
[07/04/21]seed@VM:~/Lab13$ ping www.baidu.com
PING www.a.shifen.com (112.80.248.76) 56(84) bytes of data.
64 bytes from 112.80.248.76: icmp_seq=1 ttl=55 time=8.74 ms
64 bytes from 112.80.248.76: icmp_seq=2 ttl=55 time=8.13 ms
64 bytes from 112.80.248.76: icmp_seq=3 ttl=55 time=8.33 ms
```

在 root 权限下运行 sniffer.py，捕获到的 ICMP 报文:

```
[07/04/21]seed@VM:~/Lab13$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:32:83:13
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 13066
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x92fb
     src       = 10.0.2.7
     dst       = 112.80.248.76
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
```

在普通用户权限下运行 sniffer.py：

```
[07/04/21]seed@VM:~/Lab13$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt = sniff(filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1263, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1128, in
 _run
    **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 487, i
n __init__
    socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python3.5/socket.py", line 134, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

返回错误信息，提示权限不足以支持 socket.__init__函数的调用。

Task 1.1B

1） 只捕获 ICMP 报文：

将 sniffer.py 文件中 sniff 函数的 filter 参数设为‘icmp’即可，如 Task 1.1A 所示。

2） 捕获特定 IP 地址（本机 IP 地址 10.0.2.7）23 端口发出的 TCP 报文：

修改 sniffer.py 文件如下：

```
pkt = sniff(filter='tcp and src 10.0.2.7 and dst port 23',prn=print_pkt)
```

运行 sniffer.py 文件。

创建 TCP 报文发送脚本：

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

ip=IP()
ip.src='10.0.2.7'
ip.dst='192.168.31.219'
tcp=TCP()
tcp.dport=23
send(ip/tcp)
```

运行脚本之后 sniffer.py 捕获到指定的 TCP 报文：

```
[07/04/21]seed@VM:~/Lab13$ sudo ./sniffer1.1b.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:32:83:13
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x8e45
     src       = 10.0.2.7
     dst       = 192.168.31.219
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
```

3) 捕获发往子网 128.230.0.0/16 的报文：

修改 sniffer.py 文件如下：

```
pkt = sniff(filter='dst net 128.230.0.0/16',prn=print_pkt)
```

运行 sniffer.py 文件。

创建报文发送脚本：

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

ip=IP()
ip.src='10.0.2.7'
ip.dst='128.230.0.0/16'
send(ip)
```

运行脚本之后 sniffer.py 捕获到发往子网 128.230.0.0/16 报文：

```
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:32:83:13
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0xea30
     src       = 10.0.2.7
     dst       = 128.230.3.204
     \options   \
```

Task 1.2

在这个 Task 需要用到两个虚拟机：

- VMA：10.0.2.7；
- VMB：10.0.2.8；

在 VMA 创建报文发送脚本：

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

ip = IP()
ip.dst = '10.0.2.8'
icmp=ICMP()
send(ip/icmp)
```

运行脚本，从 VMA 向 VMB 发送一个 ICMP 报文。

在 VMB 上用 WireShark 捕获到 ICMP 报文：

```
4 2021-07-05 09:56:32.9409477… 10.0.2.7    10.0.2.8    ICMP    62 Echo (ping) request
5 2021-07-05 09:56:32.9409654… 10.0.2.8    10.0.2.7    ICMP    44 Echo (ping) reply
```

可以看到 VMB 收到了 VMA 的 ICMP 报文并返回了一条 ICMP 应答报文

修改 VMA 上的报文发送脚本，添加一任意 IP 地址作为伪造源地址（以主机地址 192.168.31.219 为例）。：

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

ip = IP()
ip.src = '192.168.31.219'
ip.dst = '10.0.2.8'
icmp=ICMP()
send(ip/icmp)
```

运行脚本发送报文。

在 VMB 上用 WireShark 捕获到 ICMP 报文：

```
4 2021-07-05 10:11:32.4064238… 192.168.31.219   10.0.2.8          ICMP   62 Echo (ping) request
5 2021-07-05 10:11:32.4064483… 10.0.2.8         192.168.31.219    ICMP   44 Echo (ping) reply
```

可以看到 VMB 捕获到的 ICMP 报文源地址为伪造地址。

此时 VMA 上用 WireShark 也成功捕获到 VMB 返回的 ICMP 应答报文：

```
4 2021-07-05 10:13:29.0877521… 192.168.31.219   10.0.2.8          ICMP   44 Echo (ping) request
5 2021-07-05 10:13:29.0880280… 10.0.2.8         192.168.31.219    ICMP   62 Echo (ping) reply
```

Task 1.3

在 VMA（10.0.2.7）创建报文发送脚本：

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

ip = IP()
ip.ttl = 1
ip.dst = '192.168.31.219'
icmp=ICMP()
send(ip/icmp)
```

运行脚本发送一个 TTL=1 的 ICMP 报文。

在 VMA 上用 WireShark 捕获到 ICMP 报文：

```
3 2021-07-06 06:54:58.1471891… 10.0.2.7         192.168.31.219    ICMP   44 Echo (ping) request
4 2021-07-06 06:54:58.1474075… 10.0.2.1         10.0.2.7          ICMP   72 Time-to-live exceeded
```

报文在发送过程中 TTL 变为 0，被路由器丢弃。

将脚本中 ttl 修改为 2，再次运行报文发送脚本，在 VMA 上用 WireShark 捕获到 ICMP 报文：

```
3 2021-07-06 07:02:08.4512604… 10.0.2.7         192.168.31.219    ICMP   44 Echo (ping) request
4 2021-07-06 07:02:08.4518628… 192.168.31.219   10.0.2.7          ICMP   62 Echo (ping) reply
```

成功收到 192.168.31.219 返回的 ICMP 应答报文。

由此可以得到 VMA 与 192.168.31.219 之间的路由数量为 2。

Task 1.4

在这个 Task 需要用到两个虚拟机：

- VMA：10.0.2.7；
- VMB：10.0.2.8；

在 VMA（10.0.2.7）创建文件 sniffer.py：

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

def print_pkt(pkt):
        ip = IP()
        ip.src = pkt[IP].dst
        ip.dst = pkt[IP].src
        icmp = ICMP()
        icmp.type = "echo-reply"
        icmp.code = 0
        icmp.id = pkt[ICMP].id
        icmp.seq = pkt[ICMP].seq
        send(ip/icmp)

pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print_pkt)
```

代码作用是嗅探 ICMP 回显请求报文，如果成功捕获回显请求报文，则调用 print_pkt 函数伪造并发送一个 ICMP 回显应答报文。

正常在 VMB（10.0.2.8）向 www.baidu.com 使用 PING 指令：

```
[07/06/21]seed@VM:~$ ping www.baidu.com
PING www.a.shifen.com (112.80.248.76) 56(84) bytes of data.
64 bytes from 112.80.248.76: icmp_seq=1 ttl=55 time=11.3 ms
64 bytes from 112.80.248.76: icmp_seq=2 ttl=55 time=12.3 ms
64 bytes from 112.80.248.76: icmp_seq=3 ttl=55 time=12.4 ms
```

运行 sniffer.py，在 VMB 向 www.baidu.com 使用 PING 指令：

```
[07/06/21]seed@VM:~$ ping www.baidu.com
PING www.a.shifen.com (112.80.248.76) 56(84) bytes of data.
64 bytes from 112.80.248.76: icmp_seq=1 ttl=55 time=12.2 ms
8 bytes from 112.80.248.76: icmp_seq=1 ttl=64 (truncated)
64 bytes from 112.80.248.76: icmp_seq=2 ttl=55 time=11.1 ms
8 bytes from 112.80.248.76: icmp_seq=2 ttl=64 (truncated)
64 bytes from 112.80.248.76: icmp_seq=3 ttl=55 time=12.9 ms
8 bytes from 112.80.248.76: icmp_seq=3 ttl=64 (truncated)
```

此时 VMA 上 sniffer.py 程序会不断发出伪造的回显应答报文：

```
[07/06/21]seed@VM:~/Lab13$ sudo ./sniffer1.4.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

与正常的 PING 结果相比较可以发现 sniffer.py 运行之后 VMB 会收到两次 ICMP 回显应答，其中一次会被截断（truncated）。

Task 2

Task 2.1

在 VMA（10.0.2.7）中创建文件 sniffer2.1.c：

```c
#include <pcap.h>
#include <stdio.h>

void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{
    printf("Got a packet\n");
}

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

// Step 1: Open live pcap session on NIC with name eth3
    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);
    return 0;
}
```

代码的作用是嗅探所有收到的 ICMP 报文，如果成功获取报文，则会输出“Got a packet”。

编译生成可执行文件 sniffer，由于代码使用了 pcap 库，需要在编译命令后面增加 -lcap 参数：

```
[07/06/21]seed@VM:~/Lab13$ gcc -o sniffer sniffer2.1.c -lpcap
```

运行 sniffer，并在 VMB（10.0.2.8）向 VMA（10.0.2.7）使用 PING 指令：

```
[07/06/21]seed@VM:~$ ping 10.0.2.7
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
64 bytes from 10.0.2.7: icmp_seq=1 ttl=64 time=0.310 ms
64 bytes from 10.0.2.7: icmp_seq=2 ttl=64 time=0.344 ms
64 bytes from 10.0.2.7: icmp_seq=3 ttl=64 time=0.407 ms
```

此时 VMA 上的 sniffer 成功嗅探到了 ICMP 报文：

```
[07/06/21]seed@VM:~/Lab13$ sudo ./sniffer
Got a packet
Got a packet
Got a packet
```

Task 2.1A

创建文件 header.h：

```
#include <arpa/inet.h>
#ifndef _HEADER_H_
#define _HEADER_H_

/* ethernet headers are always exactly 14 bytes */
#define SIZE_ETHERNET 14

/* Ethernet addresses are 6 bytes */
#define ETHER_ADDR_LEN  6

/* Ethernet header */
struct sniff_ethernet {
        u_char ether_dhost[ETHER_ADDR_LEN]; /* Destination host address */
        u_char ether_shost[ETHER_ADDR_LEN]; /* Source host address */
        u_short ether_type; /* IP? ARP? RARP? etc */
};

/* IP header */
struct sniff_ip {
        u_char ip_vhl;          /* version << 4 | header length >> 2 */
        u_char ip_tos;          /* type of service */
        u_short ip_len;         /* total length */
        u_short ip_id;          /* identification */
        u_short ip_off;         /* fragment offset field */
#define IP_RF 0x8000            /* reserved fragment flag */
#define IP_DF 0x4000            /* dont fragment flag */
#define IP_MF 0x2000            /* more fragments flag */
#define IP_OFFMASK 0x1fff       /* mask for fragmenting bits */
        u_char ip_ttl;          /* time to live */
        u_char ip_p;            /* protocol */
        u_short ip_sum;         /* checksum */
        struct in_addr ip_src,ip_dst; /* source and dest address */
};
#endif
```

代码内容为 IP 和 Ethernet 数据包的头部结构体的定义。

为了输出嗅探到的报文的源地址和目的地址，对 sniffer2.1.c 进行修改：

```
#include <pcap.h>
#include <stdio.h>
#include "header.h"

void got_packet(u_char *args, const struct pcap_pkthdr *header,const u_char *packet)
{
    printf("Got a packet\n");
    struct sniff_ethernet *eth = (struct sniff_ethernet *)packet;

    if(ntohs(eth->ether_type) == 0x0800){
        struct sniff_ip *ip = (struct sniff_ip *)(packet + SIZE_ETHERNET);
        printf("IP src:%s\n",inet_ntoa(ip->ip_src));
        printf("IP dst:%s\n",inet_ntoa(ip->ip_dst));
    }
}
```

将 header.h 引入 sniffer2.1.c，然后在 got_packet 函数中增加打印 IP 数据包的源地址和目的地址的语句。

重新编译 sniffer2.1.c，运行 sniffer 并在 VMB（10.0.2.8）向 VMA（10.0.2.7）使用 PING 指令：

```
[07/06/21]seed@VM:~/Lab13$ sudo ./sniffer
Got a packet
IP src:10.0.2.8
IP dst:10.0.2.7
Got a packet
IP src:10.0.2.7
IP dst:10.0.2.8
Got a packet
IP src:10.0.2.8
IP dst:10.0.2.7
```

VMA（10.0.2.7）上 sniffer 成功打印出嗅探到的 ICMP 报文的源地址和目的地址。

Question 1

嗅探时依次调用了 5 种函数：

1) pcap_open_live 函数打开指定网卡设备准备嗅探；
2) pcap_compile 函数将过滤规则编译为 BPF 格式；
3) pcap_setfilter 函数将 BPF 格式的过滤规则传递给内核；
4) pcap_loop 函数进入循环，开始捕获数据包，捕获到即会调用设置好的回调函数，对数据包进行处理；
5) pcap_close 函数释放网卡，停止嗅探。

Question 2

使用非 root 权限运行 sniffer 时会直接返回段错误：

```
[07/06/21]seed@VM:~/Lab13$ ./sniffer
Segmentation fault
```

主要原因是因为访问网卡设备时权限不足。

Question 3

在 VMA（10.0.2.7）上运行 sniffer 运行，并从 VMB 向 www.baidu.com 使用 PING 指令：

```
[07/06/21]seed@VM:~$ ping www.baidu.com
PING www.a.shifen.com (112.80.248.75) 56(84) bytes of data.
64 bytes from 112.80.248.75: icmp_seq=1 ttl=55 time=12.3 ms
64 bytes from 112.80.248.75: icmp_seq=2 ttl=55 time=8.21 ms
64 bytes from 112.80.248.75: icmp_seq=3 ttl=55 time=8.61 ms
```

VMA 上的 sniffer 成功嗅探到数据包：

```
Got a packet
IP src:112.80.248.76
IP dst:10.0.2.8
```

将 sniffer2.1.c 中 pcap_open_live 函数的第三个参数从 1 改为 0 来关闭混杂模式：

```
    handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);
```

重新编译，在 VMA 上运行 sniffer，并从 VMB 向 www.baidu.com 使用 PING 指令：

```
[07/06/21]seed@VM:~/Lab13$ sudo ./sniffer
```

和混杂模式开启时不同，此时 VMA 上的 sniffer 不会嗅探到任何数据包。由此可以看出当混杂模式关闭时，VMA 只能嗅探到发送给自己的数据包：

Task 2.1B

- 嗅探两个特定主机之间的 ICMP 数据包：

  修改文件 sniffer2.1.c 中的 filter_exp 字符串：

  ```
      char filter_exp[] = "icmp && host 10.0.2.7 && host 10.0.2.8";
  ```

  编译运行 sniffer，从 VMB 向 VMA 或是从 VMA 向 VMB 使用 PING 指令：

  ```
  [07/07/21]seed@VM:~/Lab13$ sudo ./sniffer
  Got a packet
  IP src:10.0.2.8
  IP dst:10.0.2.7
  ```

  成功嗅探到数据包。

  从 VMA 或是 VMB 向 www.baidu.com 使用 PING 指令：

  ```
  [07/06/21]seed@VM:~/Lab13$ sudo ./sniffer
  ```

  无法嗅探到数据包。

- 嗅探目的端口号为 10-100 的 TCP 数据包：

  修改文件 sniffer2.1.c 中的 filter_exp 字符串：

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

  编译运行 sniffer，在 VMB 运行如下报文发送脚本向 VMA 发送端口为 23 的 TCP 报文：

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

  ip=IP()
  ip.dst='10.0.2.7'
  tcp=TCP()
  tcp.dport=23
  send(ip/tcp)
  ```

VMA 上的 sniffer 成功嗅探到 VMB 发来的 TCP 数据包：

```
[07/07/21]seed@VM:~/Lab13$ sudo ./sniffer
Got a packet
IP src:10.0.2.8
IP dst:10.0.2.7
Got a packet
IP src:10.0.2.7
IP dst:10.0.2.8
```

修改 VMB 上的报文发送脚本，将端口号改为 9：

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

ip=IP()
ip.dst='10.0.2.7'
tcp=TCP()
tcp.dport=9
send(ip/tcp)
```

VMA 上的 sniffer 成功没有嗅探到从 VMB 上发来的 TCP 数据包，而是首先嗅探到了 VMA 返回给 VMB 的 TCP 数据包：

```
[07/07/21]seed@VM:~/Lab13$ sudo ./sniffer
Got a packet
IP src:10.0.2.7
IP dst:10.0.2.8
```

Task 2.1C

在这个 Task 需要用到三个虚拟机：

- VMA：10.0.2.7；
- VMB：10.0.2.8；
- VMC：10.0.2.9；

在 header.h 中加入 TCP 数据包的头部结构体的定义：

```
/* TCP header */
typedef u_int tcp_seq;

struct sniff_tcp {
        u_short th_sport;       /* source port */
        u_short th_dport;       /* destination port */
        tcp_seq th_seq;         /* sequence number */
        tcp_seq th_ack;         /* acknowledgement number */
        u_char th_offx2;        /* data offset, rsvd */
#define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
        u_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)
        u_short th_win;         /* window */
        u_short th_sum;         /* checksum */
        u_short th_urp;         /* urgent pointer */
};
```

因为 telnet 使用 TCP 协议且固定端口号为 23，所以为了针对地嗅探 telnet 的数据包，修改文件 sniffer2.1.c 中 filter_exp 字符串：

```
    char filter_exp[] = "tcp port 23";
```

修改 got_packet 函数：

```
        u_int size_ip = IP_HL(ip)*4;

        const struct sniff_tcp *tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);

        u_int size_tcp = TH_OFF(tcp)*4;

        const char *playload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp );

        if(strlen(playload) != 0) {
                printf("%s",playload);
        }
```

代码的作用是将数据包层层分解，找到 TCP 报文的位置并输出其数据部分。

编译运行 sniffer，在 VMB（10.0.2.8）向 VMC（10.0.2.9）使用 telnet 指令：

```
[07/07/21]seed@VM:~$ telnet 10.0.2.9
Trying 10.0.2.9...
Connected to 10.0.2.9.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Wed Jul  7 04:19:31 EDT 2021 from 10.0.2.8 on pts/3
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-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.

[07/07/21]seed@VM:~$
```

此时 VMA（10.0.2.7）上的 sniffer 成功嗅探到 telnet 发出的 TCP 报文：

```
[07/07/21]seed@VM:~/Lab13$ sudo ./sniffer
                                                                                
                                                                                
                                                                                
```

ICMP 请求报文，为了节省篇幅，我选择将两者结合，直接伪造一个 ICMP 请求报文并发送。

在 header.h 中加入 ICMP 数据包的头部结构体的定义：

```
/* ICMP header*/
struct sniff_icmp {
        u_char  icmp_type; // ICMP message type
        u_char  icmp_code; // Error code
        u_short  icmp_chksum; //Checksum for ICMP Header and
        u_short  icmp_id;     //Used for identifying request
        u_short  icmp_seq;    //Sequence number
};
```

创建文件 sniffer2.2.c：

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

    int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

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

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

    sendto(sock, ip, ntohs(ip->ip_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));

    close(sock);
}

int main()
{
    char buffer[1500];

    memset(buffer, 0, 1500);

    struct sniff_icmp *icmp = (struct sniff_icmp *)(buffer + sizeof(struct sniff_ip));

    icmp->icmp_type = 8;
    icmp->icmp_chksum = 0;
    icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct sniff_icmp));

    struct sniff_ip *ip = (struct sniff_ip *) buffer;
    ip->ip_vhl = 4 << 4 | 5;
    ip->ip_tos = 0;
    ip->ip_off = 0;
    ip->ip_ttl = 64;
    ip->ip_src.s_addr = inet_addr("1.2.3.4");
    ip->ip_dst.s_addr = inet_addr("10.0.2.7");

    ip->ip_p = IPPROTO_ICMP;
    ip->ip_len = htons(sizeof(struct sniff_ip) + sizeof(struct sniff_icmp));

    send_raw_ip_packet(ip);

    return 0;
}
```

代码的作用是伪造一个 ICMP 报文并使用 raw socket 发送。

要伪造数据包首先要创建一个缓冲区，然后依次填入 ICMP 头部信息和 IP 头部信息，其中 ICMP 请求报文头部只需要设置类型和校验和字段（校验和字段通过 in_cksum 函数生成）；完成数据包的伪造之后，需要将缓冲区的指针传递给 send_raw_ip_packet 函数，send_raw_ip_packet 函数的作用是创建并设置一

个 raw socket 然后以此将数据包发送出去。

编译并运行 sniffer2， WireShark 成功捕获到 ICMP 请求和响应报文：

```
1 2021-07-07 08:38:22.6740960… 1.2.3.4          10.0.2.7          ICMP      44 Echo (ping) request
2 2021-07-07 08:38:22.6741072… 10.0.2.7         1.2.3.4           ICMP      44 Echo (ping) reply
```

将目的地址修改为百度的 IP 地址 112.80.248.75，重新编译运行 sniffer2：

```
2 2021-07-07 08:57:26.0676438… 10.0.2.7         112.80.248.75     ICMP      44 Echo (ping) request
3 2021-07-07 08:57:26.0763599… 112.80.248.75    10.0.2.7          ICMP      62 Echo (ping) reply
```

成功发出 ICMP 请求报文并收到了回应报文。

Question 4

1） 修改 sniffer2.1.c 中的 IP 数据包长度：

```
    ip->ip_len = htons(17);
```

编译并运行 sniffer2，WireShark 没有捕获到任何报文。

2） 再次修改 sniffer2.1.c 中的 IP 数据包长度：

```
    ip->ip_len = htons(100);
```

编译并运行 sniffer2，成功捕获到 ICMP 报文，此时报文长度为 116。

```
3 2021-07-07 10:20:49.5635812… 10.0.2.7         112.80.248.75     ICMP     116 Echo (ping) request
4 2021-07-07 10:20:49.5710714… 112.80.248.75    10.0.2.7          ICMP     116 Echo (ping) reply
```

3） 再次修改 sniffer2.1.c 中的 IP 数据包长度：

```
    ip->ip_len = htons(80000);
```

编译并允许 sniffer2，WireShark 没有捕获到任何报文。

由上面实验可知，IP 数据包长度不能过大或者是过小，在合法的 IP 数据包长度内，可以任意设置 IP 数据包长度。

Question 5

使用 raw socket 不需要计算 IP 报文的校验和，因为在数据包的发送过程中操作系统会设置该字段。

Question 6

只有 root 权限的进程才可以创建 raw socket。使用非 root 权限运行程序会在调用套接字初始化函数 socket 函数时出错。

Task 2.3

在这个 Task 需要用到两个虚拟机：

- VMA：10.0.2.7；
- VMB：10.0.2.8；

在 VMA（10.0.2.7）创建文件 sniffer2.3.c，核心代码如下：

```
int spoof(int sock, struct in_addr ip_src, struct in_addr ip_dst, struct sniff_icmp * r_icmp)
{
        char buffer[1024];
        struct sockaddr_in sin;
        int len = 0;

        sin.sin_family = AF_INET;
        struct sniff_ip *ip = (struct sniff_ip *) buffer;
        struct sniff_icmp *icmp = (struct sniff_icmp *)(buffer + sizeof(struct sniff_ip));

        len += sizeof(struct sniff_icmp);
        icmp->icmp_type = 0;
        icmp->icmp_code = 0;
        icmp->icmp_chksum = 0;
        icmp->icmp_id = r_icmp->icmp_id;
        icmp->icmp_seq = r_icmp->icmp_seq;
        icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct sniff_icmp));

        for(int i = 0; i < ICMP_DATA_LENGTH; i++){
                icmp->icmp_data[i] = r_icmp->icmp_data[i];
        }

        ip->ip_vhl = 4<<4 | 5;
        ip->ip_tos = 0;
        ip->ip_len = htons(sizeof(struct sniff_ip) + sizeof(struct sniff_icmp));
        ip->ip_off = 0;
        ip->ip_ttl = 64;
        ip->ip_p = IPPROTO_ICMP;
        ip->ip_src = ip_dst;
        ip->ip_dst = ip_src;

        send_raw_ip_packet(ip);
}
void got_packet(u_char * args, const struct pcap_pkthdr *header, const u_char *packet)
{
        struct ethheader *eth = (struct ethheader *)packet;
        struct sniff_ip *ip = (struct sniff_ip *)(packet + SIZE_ETHERNET);

        printf("IP src:%s\n",inet_ntoa(ip->ip_src));
        printf("IP dst:%s\n",inet_ntoa(ip->ip_dst));

        u_int size_ip = IP_HL(ip)*4;

        struct sniff_icmp *icmp = (struct sniff_icmp *)(packet + SIZE_ETHERNET + size_ip);

        int sock = (int)args;

        spoof(sock, ip->ip_src, ip->ip_dst, icmp);
}
```

代码的作用是实现了报文的嗅探与伪造。首先不断尝试嗅探 ICMP 回显请求报文，如果成功则调用 send_raw_ip_packet 函数以及 spoof 函数伪造并发送一个 ICMP 回显应答报文。

sniffer2.3.c 整合了 sniffer1.4.py、sniffer2.1.c 与 sniffer2.2.c，主要思路为：

1) 将伪造 ICMP 数据包的部分提取出来抽象整合为 spoof 函数；
2) 依旧使用 got_packet 函数作为回调函数，输出嗅探到的数据包的源地址与

目的地址，然后调用 spoof 函数；

3) 在 proof 函数中进行数据包的伪造，然后调用 send_raw_ip_packet 函数；
4) 依旧使用 send_raw_ip_packet 函数来进行 raw socket 的创建以及伪造报文的发送；
5) 依旧使用 in_cksum 函数来得到 ICMP 报文的检验和字段；
6) 修改文件 sniffer2.1.c 中 main 函数中的 filter_exp 字符串，从而实现针对 ICMP 请求报文进行嗅探：

```
        char filter_exp[] = "icmp[icmptype] == icmp-echo";
```

正常在 VMB（10.0.2.8）向 1.2.3.4 使用 PING 指令：

```
[07/07/21]seed@VM:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 58.240.200.197 icmp_seq=1207 Destination Net Unreachable
```

不会有任何数据包回复。

编译并运行 sniffer3，在 VMB 向 1.2.3.4 使用 PING 指令：

```
[07/07/21]seed@VM:~$ 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=569 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=595 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=619 ms
```

此时 VMA 上 sniffer.py 程序嗅探到 VMB 发出的 ICMP 回显请求报文，打印出报文的源地址和目的地址：

```
[07/07/21]seed@VM:~/Lab13$ sudo ./sniffer3
IP src:10.0.2.8
IP dst:1.2.3.4
IP src:10.0.2.8
IP dst:1.2.3.4
IP src:10.0.2.8
IP dst:1.2.3.4
```

与正常的 PING 结果相比较，可以发现 sniffer3 运行之后 VMB 会收到 VMA 伪造的 ICMP 回显应答。

VMB 的 WireShark 也能够证明 VMA 成功嗅探并伪造了 ICMP 回显应答报文：

1	2021-07-07 12:13:11.2775952…	10.0.2.8	1.2.3.4	ICMP	100 Echo (ping) request
2	2021-07-07 12:13:11.4899373…	1.2.3.4	10.0.2.8	ICMP	100 Echo (ping) reply