Sniffing and Spoofing Lab

Before:

配置环境，下载virtualbox，配置虚拟机，完成共享文件夹的搭建（自动挂载有bug，必须加一个开机自运行的脚本实现自动挂载，这个问题还困扰了一段时间，恼），至于实验中要用到的docker工具，scapy等发现虚拟机已经配好了。

容器配置：

最大问题就是docker的网站总是连接超时，被迫配置一个脚本使用镜像网站（但是网上的相当一部份都已经不能用了，不过最终还是解决了），接着用`dockps`和`docker network ls`查看了当前连接

```
[07/13/25]seed@VM:~/Desktop$ dockps
98bf2448fa0c  hostA-10.9.0.5
21ac405202a3  hostB-10.9.0.6
455e91fd9ed0  seed-attacker
```

```
[07/13/25]seed@VM:~/Desktop$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
f35e6cb4574a        bridge              bridge              local
b3581338a28d        host                host                local
28b83139cdb7        net-10.9.0.0        bridge              local
77acecccbe26        none                null                local
[07/13/25]seed@VM:~/Desktop$
```

用`ifconfig`查看网络接口，得到MAC地址`02:42:73:09:99:60`

```
[07/13/25]seed@VM:~/Desktop$ ifconfig
br-28b83139cdb7: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:73ff:fe09:9960  prefixlen 64  scopeid 0x20<link
>
        ether 02:42:73:09:99:60  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 53  bytes 6129 (6.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Task1 Scapy嗅探和伪造数据包

1.1A运行并观察sniffer.py(样例)

首先`chmod a+x sniffer.py`使程序可执行，再用root权限运行，此时无任何成果，这时`ping`一下hostA,可观察到相关packet被捕获了

```
[07/13/25]seed@VM:~/.../volumes$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:73:09:99:60
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 40140
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x89c5
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x4392
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = ':\xb7sh\x00\x00\x00\x00Ax\x06\x00\x00\x00\x00\
```

（仅仅取一个packet，可以看出是由虚拟机向容器发出的）

```
###[ Ethernet ]###
  dst       = 02:42:73:09:99:60
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 4642
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x5470
     src       = 10.9.0.5
     dst       = 10.9.0.1
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x4b92
        id        = 0x2
        seq       = 0x1
###[ Raw ]###
           load      = ':\xb7sh\x00\x00\x00\x00Ax\x06\x00\x00\x00\x00\
x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"
```

（这个的dst和src就反过来了，形成一个完整的request-reply)

再切换到seed账号，不使用root权限运行，会发现显示权限不足

```
[07/13/25]seed@VM:~/.../volumes$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    sniff(iface='br-28b83139cdb7',filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", lin
e 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", lin
e 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", l
ine 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket
.htons(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
```

这似乎是因为Linux内核为了保护系统安全，默认禁止普通程序直接访问数据链路层。

1.1B设置过滤器并再次演示

该实验的重点在于设置过滤器的BPF语法。基本用法如下

以下是一些常见的BPF语法示例：

1. 捕获特定源IP地址的数据包：

   `src host 10.9.0.1` ：捕获源IP地址为10.9.0.1的数据包。

2. 捕获特定目的IP地址的数据包：

   `dst host 10.9.0.6` ：捕获目的IP地址为10.9.0.6的数据包。

3. 捕获特定协议类型的数据包：

   `icmp` ：捕获ICMP协议的数据包。

   `tcp` ：捕获 TCP 协议的数据包。

   `udp` ：捕获 UDP 协议的数据包。

4. 捕获特定端口号的数据包：

   `tcp dst port 80` ：捕获目的端口号为80的TCP数据包。

   `udp src port 53` ：捕获源端口号为53的UDP数据包。

5. 使用逻辑运算符and和or组合多个条件：

   `src host 10.9.0.1 and tcp dst port 80` ：捕获源IP地址为10.9.0.1且目的端口号为80的数据包。

   `src host 10.9.0.1 or dst host 10.9.0.6` ：捕获源IP地址为10.9.0.1或目的IP地址为10.9.0.6的数据包。

所以对应的过滤器应分别为

1. `filter='icmp'`

2. `filter='tcp and dst port 23'`

3. `filter='net 128.230.0.0/16'`

演示部分略

1.2伪造ICMP包

本实验要用到wireshark，seed-ubuntu上已准备

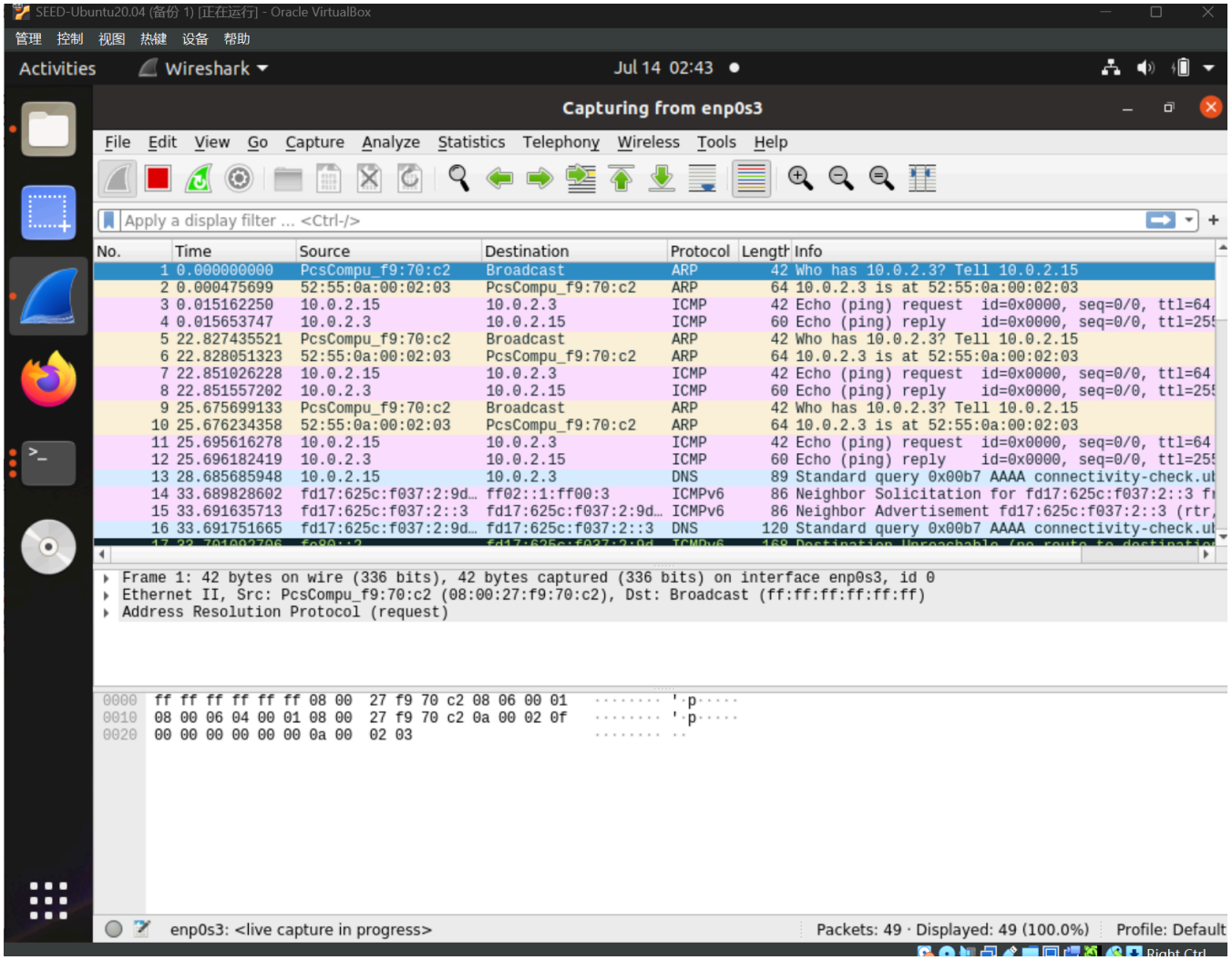
编写一个sendFake.py

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

ip=IP()
icmp=ICMP()

ip.dst ='10.0.2.3'
pkt=ip/icmp
send(pkt)
pkt.show()
```

并在终端中运行，查看发出的数据包的信息

```
[07/14/25]seed@VM:~/.../volumes$ sudo python3 sendFake.py
.
Sent 1 packets.
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.0.2.15
  dst       = 10.0.2.3
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0
```

并使用wireshark监测到icmp包



1.3Traceroute

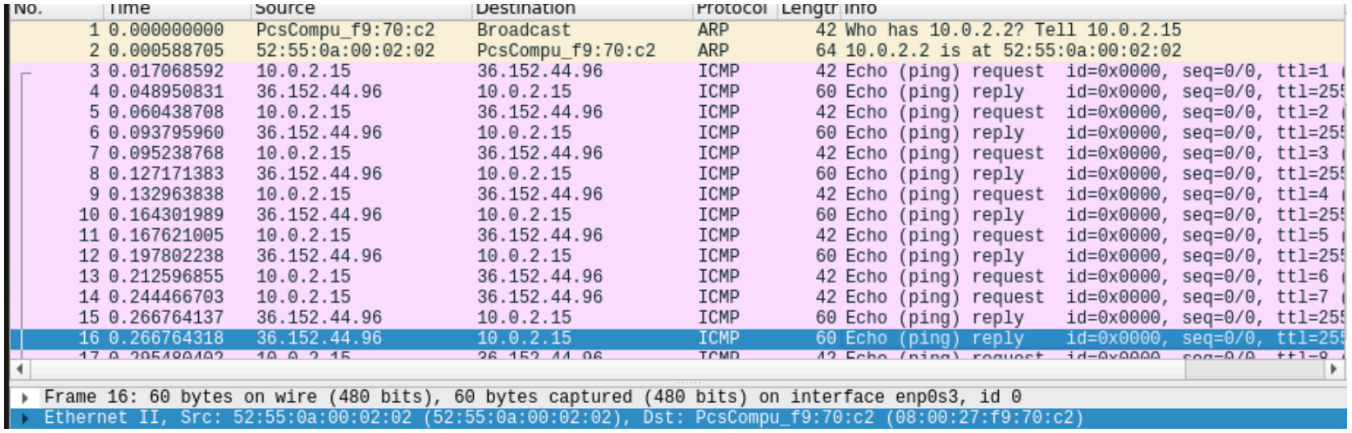
显然这个任务要使用循环进行不断地发送，直到得到最终结果或超过预计范围

先编写traceroute.py

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

ip=IP()
ip.dst='1.2.3.4'
icmp=ICMP()
pkt=ip/icmp
for i in range(1,20):
    pkt[IP].ttl=i
    send(pkt)
```

起初发向1.2.3.4，发现一直没有回应

在网上找到实验案例，发向36.152.44.96

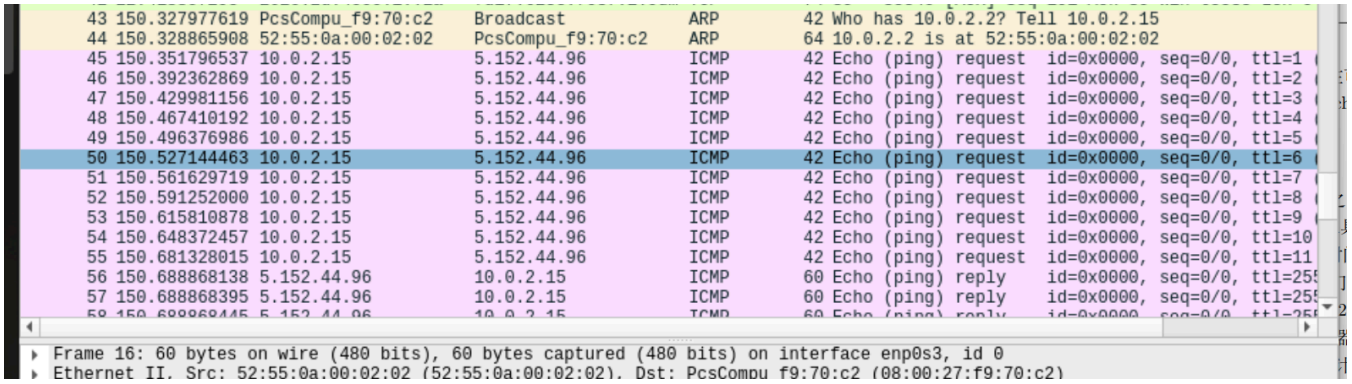


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_f9:70:c2	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
2	0.000588705	52:55:0a:00:02:02	PcsCompu_f9:70:c2	ARP	64	10.0.2.2 is at 52:55:0a:00:02:02
3	0.017068592	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1
4	0.048950831	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
5	0.060438708	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2
6	0.093795960	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
7	0.095238768	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3
8	0.127171383	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
9	0.132963838	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4
10	0.164301989	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
11	0.167621005	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5
12	0.197802238	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
13	0.212596855	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6
14	0.244466703	10.0.2.15	36.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7
15	0.266764137	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
16	0.266764318	36.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255

Frame 16: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface enp0s3, id 0
Ethernet II, Src: 52:55:0a:00:02:02 (52:55:0a:00:02:02), Dst: PcsCompu_f9:70:c2 (08:00:27:f9:70:c2)

但是第一个请求就回应了，未起到实验要求

接着尝试发向5.152.44.96



43	150.327977619	PcsCompu_f9:70:c2	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.15
44	150.328865908	52:55:0a:00:02:02	PcsCompu_f9:70:c2	ARP	64	10.0.2.2 is at 52:55:0a:00:02:02
45	150.351796537	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1
46	150.392362869	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2
47	150.429981156	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3
48	150.467410192	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4
49	150.496376986	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5
50	150.527144463	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6
51	150.561629719	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7
52	150.591252000	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8
53	150.615810878	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9
54	150.648372457	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10
55	150.681328015	10.0.2.15	5.152.44.96	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11
56	150.688868138	5.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255
57	150.688868395	5.152.44.96	10.0.2.15	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=255

Frame 16: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface enp0s3, id 0
Ethernet II, Src: 52:55:0a:00:02:02 (52:55:0a:00:02:02), Dst: PcsCompu_f9:70:c2 (08:00:27:f9:70:c2)

当ttl=11时回应了，说明中间有10个路由器，但很奇怪的是没有超时报错，无法找到路线(搞不明白，明明tcp异常还显示，怀疑是家里网络的问题)，啧，原理搞懂就行。

1.4窃听和伪造结合

创建以下程序，并且运行

```
#!/usr/bin/python3
from scapy.all import*
def sending(pkt):
    ip=IP()
    icmp=ICMP()
    ip.dst=pkt[IP].src
    ip.src=pkt[IP].dst
    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]==8',prn=sending)
```

1.首先ping一下互联网上不存在的主机

```
[07/14/25]seed@VM:~$ 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 ---
36 packets transmitted, 0 received, 100% packet loss, time 351
76ms
```

显然无回应

2.再ping一下局域网上的不存在主机

```
[07/14/25]seed@VM:~$ ping 10.9.0.7
PING 10.9.0.7 (10.9.0.7) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
^C
--- 10.9.0.7 ping statistics ---
11 packets transmitted, 0 received, +9 errors, 100% packet los
s, time 10220ms
pipe 3
```

这是因为监听的端口是主机端口，而icmp发文在其内部，所以未监听到

3.最后ping一下互联网上的一个存在的主机

```
[07/14/25]seed@VM:~$ ping 43.138.70.209
PING 43.138.70.209 (43.138.70.209) 56(84) bytes of data.
64 bytes from 43.138.70.209: icmp_seq=1 ttl=255 time=32.8 ms
8 bytes from 43.138.70.209: icmp_seq=2 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=2 ttl=255 time=34.7 ms (
DUP!)
8 bytes from 43.138.70.209: icmp_seq=3 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=3 ttl=255 time=32.3 ms (
DUP!)
8 bytes from 43.138.70.209: icmp_seq=4 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=4 ttl=255 time=33.1 ms (
DUP!)
8 bytes from 43.138.70.209: icmp_seq=5 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=5 ttl=255 time=32.1 ms (
DUP!)
8 bytes from 43.138.70.209: icmp_seq=6 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=6 ttl=255 time=38.2 ms (
DUP!)
8 bytes from 43.138.70.209: icmp_seq=7 ttl=64 (truncated)
64 bytes from 43.138.70.209: icmp_seq=7 ttl=255 time=33.7 ms (
DUP!)
```

ttl=64的即伪造的数据包，而`(DUP!)`表示数据包被重复接收了多次

Task2编写程序以窃听和欺骗数据包

2.1A理解窃听器的工作原理

代码如下（来自PDF）

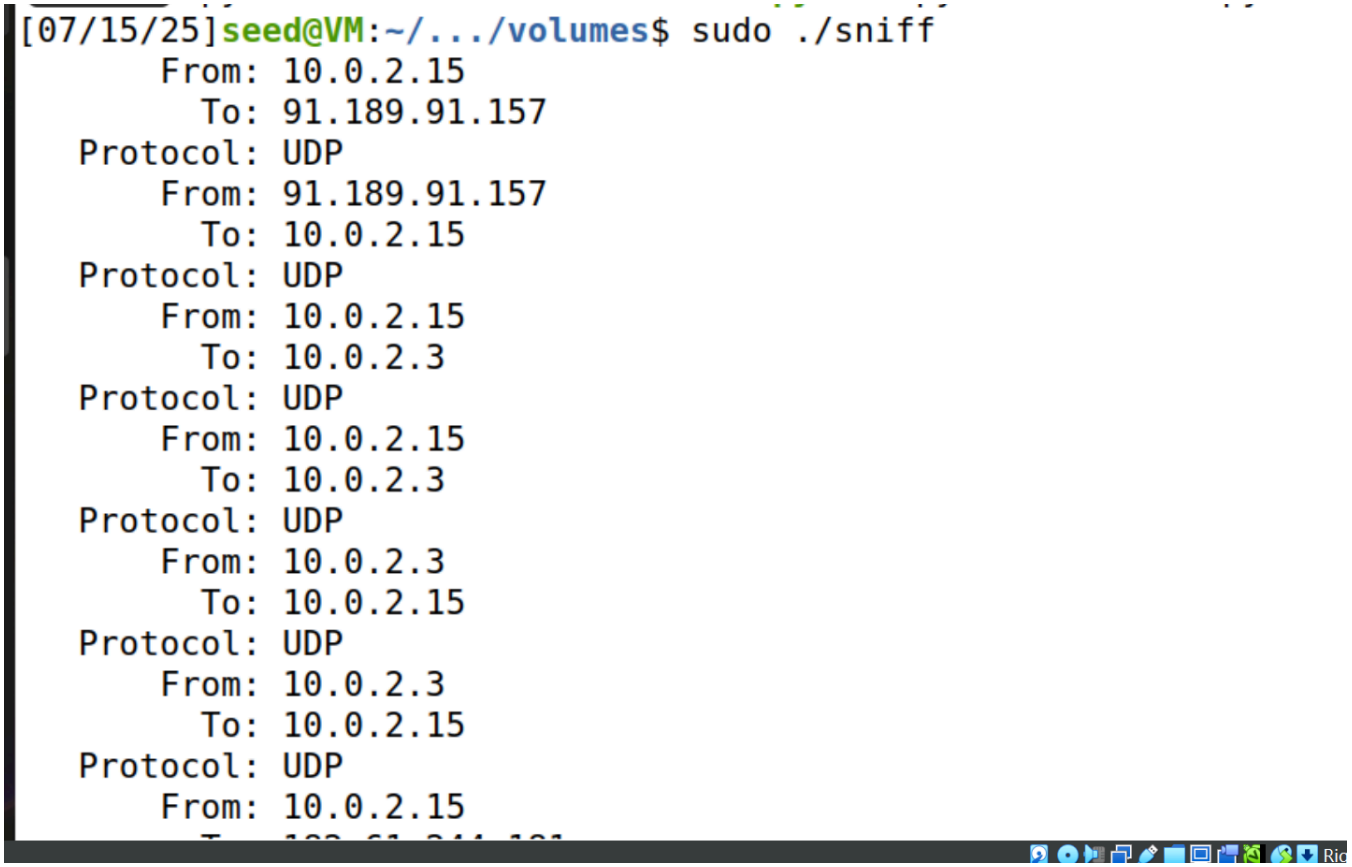
```c
#include <pcap.h>

#include <stdio.h>

#include <arpa/inet.h>



/* Ethernet header */

struct ethheader {

  u_char  ether_dhost[6]; /* destination host address */

  u_char  ether_shost[6]; /* source host address */

  u_short ether_type;     /* protocol type (IP, ARP, RARP, etc) */
```

```
};

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

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

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

```
    printf("       From: %s\n", inet_ntoa(ip->iph_sourceip));
    printf("         To: %s\n", inet_ntoa(ip->iph_destip));

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

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 enp0s3
  handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);

  // Step 2: Compile filter_exp into BPF psuedo-code
  pcap_compile(handle, &fp, filter_exp, 0, net);
  pcap_setfilter(handle, &fp);

  // Step 3: Capture packets
  pcap_loop(handle, -1, got_packet, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}
```

运行后ping了下百度，嗅探结果如下



A1:首先开始一个有效的pcap对话，同时将网络设置为混杂模式；然后通过pcap_compile和pcap_setfilter设置过滤器;接着pcap_loop循环，尝试不断捕获数据包。

A2：因为嗅探程序可能得到密码等重要信息，影响电脑、网络的安全，所以通过提高权限来绕过限制；没有root权限时，pcap_open_live中设置网络为混杂模式的部分会受影响。

A3：混杂模式可以监听所在网段下其他机器的数据包，关闭则只能监听与网卡信息一致的数据包。

2.1B编写筛选器

需要调整的只有`char filter_exp[] = "ip proto icmp";`

1、假设只捕捉主机10.9.0.15和10.0.2.3之间的，改为“icmp and host 10.9.0.15 and host 10.0.2.3”

2、改为”tcp and dst portrange 10-100“

至于结果就偷个懒，不截屏了

2.1C窃听密码

根据要求修改下程序，把`got_packet`函数修改为`packet_handler`，修改网络接口和过滤器，代码如下（AI居功甚伟）

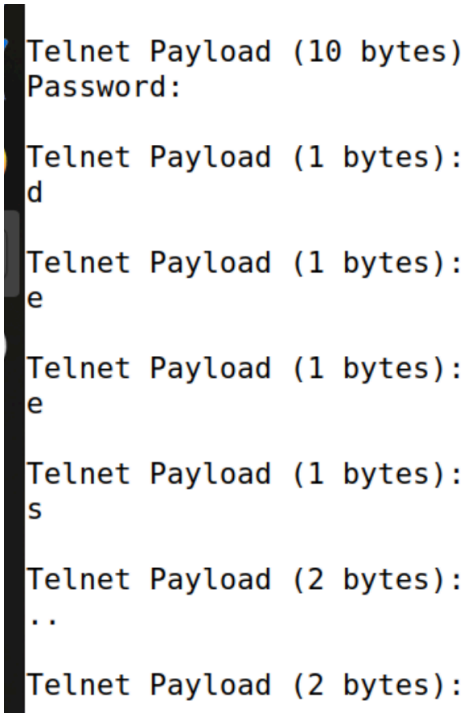
```
#include <pcap.h>
#include <stdio.h>
#include <arpa/inet.h>

/* Ethernet header */
struct ethheader {
```

```
  u_char  ether_dhost[6]; /* destination host address */
  u_char  ether_shost[6]; /* source host address */
  u_short ether_type;     /* protocol type (IP, ARP, RARP, etc) */
};

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

// TCP头部结构体（20字节 + 选项）
struct tcpheader {
    u_short th_sport;         // 源端口
    u_short th_dport;         // 目标端口
    u_int   th_seq;           // 序列号
    u_int   th_ack;           // 确认号
    u_char  th_offx2;         // 数据偏移(4位) + 保留(4位)
    #define TH_OFF(th)      (((th)->th_offx2 & 0xf0) >> 4)
    u_char  th_flags;         // 标志位
    #define TH_FIN  0x01
    #define TH_SYN  0x02
    #define TH_RST  0x04
    #define TH_PUSH 0x08
    #define TH_ACK  0x10
    #define TH_URG  0x20
    u_short th_win;           // 窗口大小
    u_short th_sum;           // 校验和
    u_short th_urp;           // 紧急指针
};

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

    // 仅处理IPv4数据包
    if (ntohs(eth->ether_type) != 0x0800)
        return;

    // 计算IP头部位置和长度
    struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
    int ip_header_len = ip->iph_ihl * 4;  // 转换为字节
```

```
    // 仅处理TCP数据包
    if (ip->iph_protocol != IPPROTO_TCP)
        return;

    // 计算TCP头部位置和长度
    struct tcpheader *tcp = (struct tcpheader *)((u_char*)ip + ip_header_len);
    int tcp_header_len = TH_OFF(tcp) * 4;  // 转换为字节

    // 计算TCP负载位置和长度
    const u_char *payload = (u_char*)tcp + tcp_header_len;
    int payload_len = ntohs(ip->iph_len) - (ip_header_len + tcp_header_len);

    // 打印负载内容
    if (payload_len > 0) {
        printf("Telnet Payload (%d bytes):\n", payload_len);
        for (int i = 0; i < payload_len; i++) {
            // 打印ASCII字符（可打印字符）
            if (payload[i] >= 32 && payload[i] <= 126)
                printf("%c", payload[i]);
            else
                printf(".");  // 不可打印字符用点号表示

            // 每行结尾换行
            if ((i + 1) % 80 == 0)
                printf("\n");
        }
        printf("\n\n");
    }
}

int main()
{
  pcap_t *handle;
  char errbuf[PCAP_ERRBUF_SIZE];
  struct bpf_program fp;
  char filter_exp[] = "tcp port 23";
  bpf_u_int32 net;

  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("br-28b83139cdb7", 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, packet_handler, NULL);

  pcap_close(handle);   //Close the handle
  return 0;
}
```

在容器里运行两台主机，并用其中一个telnet另一个

再观察已经运行了程序的攻击者，发现密码已经窃取成功



2.2伪造数据包

代码如下(请忽略中文注释变成的乱码)

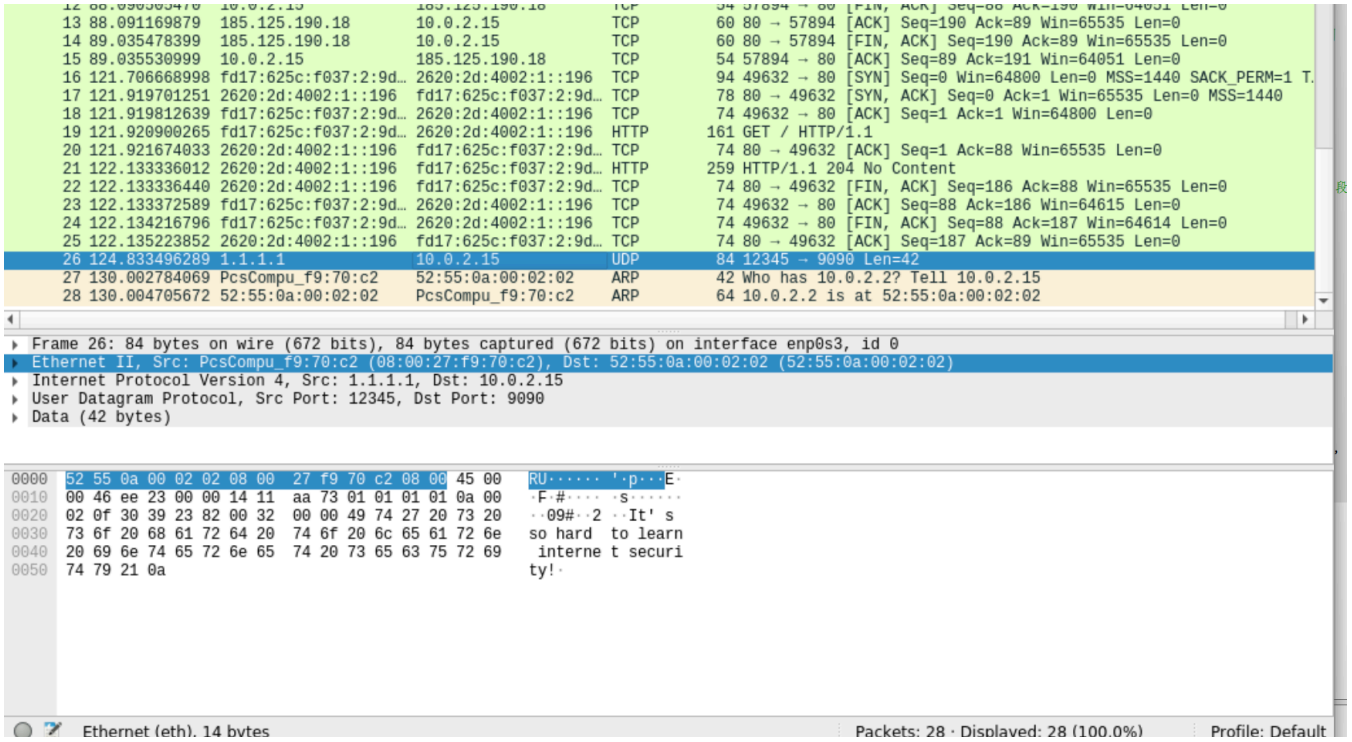
```
#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include<sys/socket.h>
```

```
#include<netinet/ip.h>

#include<arpa/inet.h>

#include<stdlib.h>

#include"headers.h"


// void send_raw_ip_packet(struct ipheader* 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->iph_destip;


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

//     close(sock);

// }



int main(){

    int sd;

    struct sockaddr_in sin;

    char buffer[1024]; // ¿ÉÒÔžüžÄ»º³åÇøŽóÐ¡

    /* ŽŽœšÒ»žöŸßÓÐ IP Ð•ÒéµÄ raw socket¡£ IPPROTO_RAW ²ÎÊý±íÃ÷
```

```
* IP Í·²¿ÒÑŸ•Ìá'©£¬ ÈÃ²Ù×÷ÏµÍ³²»ÒªÌíŒÓÐÂµÄ IP Í·²¿ */

sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

if(sd < 0) {

perror("socket() error"); exit(-1);

}

/* ÕâžöÊýŸÝœá'¹ÓÃÓÚ·¢ËÍÊýŸÝ°üÊ±Ê'ÓÃ¡£ Íš³£ÎÒÃÇÐèÒªÌî³ä¶àžö×Ö¶Î£¬

* µ«¶ÔÓÚ raw sockets£¬ ÎÒÃÇÖ»ÐèÒªÌî³äÕâžö×Ö¶Î */

sin.sin_family = AF_INET;

// ÔÚÕâÀï¿ÉÒÔÊ'ÓÃ buffer[] ''Ôì IP ÊýŸÝ°ü

// - ''œš IP Í·²¿ ...

// - ''œš TCP/UDP/ICMP Í·²¿ ...

// - Èç'ûÐèÒªµÄ»°£¬ ÌîÐŽÊýŸÝ²¿·Ö ...

// ÐèÒª×¢Òâ×ÖœÚË³Ðò¡£

memset(buffer,0,1024);

struct ipheader *ip = (struct ipheader*)buffer;

struct udpheader *udp = (struct udpheader*)(buffer+sizeof(struct ipheader));

char *data = buffer +sizeof(struct ipheader)+sizeof(struct udpheader);

const char *msg = "It' s so hard to learn internet security!\n";

strncpy(data,msg,strlen(msg));


udp->udp_sport = htons(12345);

udp->udp_dport = htons(9090);

udp->udp_ulen = htons(sizeof(struct udpheader) + strlen(msg));

udp->udp_sum =  0;


ip->iph_ver = 4;

ip->iph_ihl = 5;
```

```
    ip->iph_ttl = 20;

    ip->iph_sourceip.s_addr = inet_addr("1.1.1.1");

    ip->iph_destip.s_addr = inet_addr("10.0.2.15");

    ip->iph_protocol = IPPROTO_UDP; // The value is 17.

    ip->iph_len = htons(sizeof(struct ipheader) +sizeof(struct udpheader) + strlen(msg));



    /* ·¢ËÍ IP ÊýŸ°ü¡£ ip_len ÊÇÊýŸ°üµÄÊµŒÊŽóÐ¡ */

    if(sendto(sd, buffer, ntohs(ip->iph_len), 0, (struct sockaddr *)&sin, sizeof(sin)) < 0)
{

        perror("sendto() error"); exit(-1);

    }

}
```

代码中注释掉的函数就是最根本的发送伪造数据包的函数代码，为了按照实验要求的框架完成，故不用该函数，头文件`headers.h`来自网上的一个基本的对于各个结构体的定义，实验后结果如下



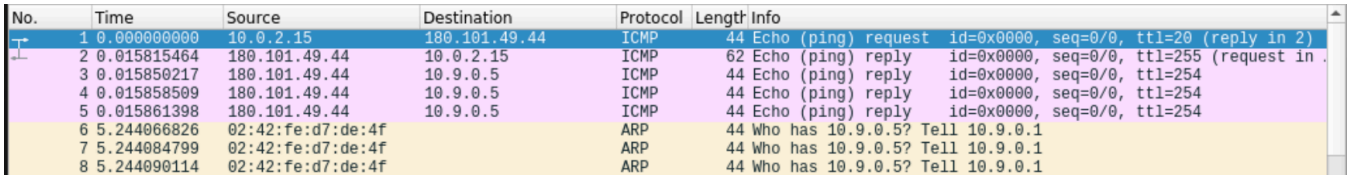
可见伪造数据包成功。

2.2B伪造ICMP echo 请求数据包

这一实验用到了校验和计算的知识，直接用的书上内容，和所要求的程序共同编译得到应用程序

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

void send_raw_ip_packet(struct ipheader* 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->iph_destip;

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

unsigned short in_cksum(unsigned short *buf,int length);

int main(){
    char buffer[1024];
    memset(buffer,0,1024);
    struct ipheader *ip = (struct ipheader*)buffer;
    struct icmpheader *icmp = (struct icmpheader*)(buffer+sizeof(struct ipheader));
    icmp->icmp_type = 8;
    icmp->icmp_chksum = in_cksum((unsigned short*)icmp,sizeof(struct icmpheader));

    ip->iph_ver = 4;
    ip->iph_ihl = 5;
    ip->iph_ttl = 20;
    ip->iph_sourceip.s_addr = inet_addr("10.9.0.5");
    ip->iph_destip.s_addr = inet_addr("180.101.49.44");
    ip->iph_protocol = IPPROTO_ICMP;
    ip->iph_len = htons(sizeof(struct ipheader) +sizeof(struct icmpheader));

    send_raw_ip_packet(ip);
    return 0;
}
```

监控any端口，结果如下,可见目标IP向虚假的地址回应。



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.15	180.101.49.44	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 2)
2	0.015815464	180.101.49.44	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=255 (request in
3	0.015850217	180.101.49.44	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=254
4	0.015858509	180.101.49.44	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=254
5	0.015861398	180.101.49.44	10.9.0.5	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=254
6	5.244066826	02:42:fe:d7:de:4f		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
7	5.244084799	02:42:fe:d7:de:4f		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
8	5.244090114	02:42:fe:d7:de:4f		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1

A4：不可以，调小长度一定会破坏数据包，调大长度也可能造成缓存溢出，导致程序异常

A5：必须计算，否则默认缺省为0，在终点处通不过检验会被直接丢弃

A6：系统为了安全不允许无权限用户执行该操作

Task2.3嗅探和伪造结合

直接上代码

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

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

    while (nleft > 1) {
        sum += *w++;
        nleft -= 2;
    }

    if (nleft == 1) {
        *(u_char *)(&temp) = *(u_char *)w;
        sum += temp;
    }

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

void send_raw_ip_packet(struct ipheader* 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));
```

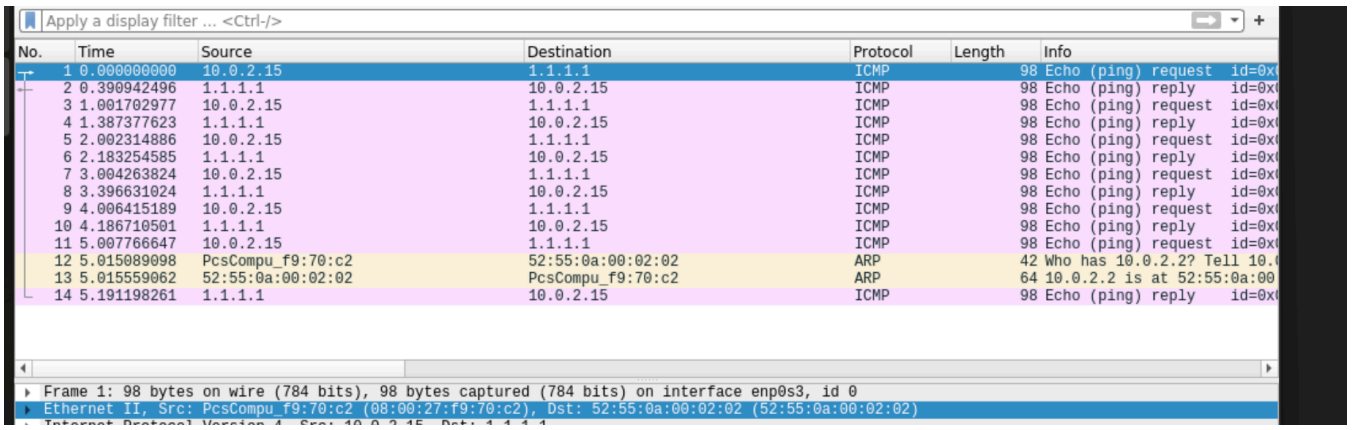
```
    dest_info.sin_family = AF_INET;
    dest_info.sin_addr = ip->iph_destip;

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

void send_icmp_reply(struct ipheader* orig_ip, struct icmpheader* orig_icmp) {

    int ip_header_len = orig_ip->iph_ihl * 4;
    int icmp_data_len = ntohs(orig_ip->iph_len) - ip_header_len - sizeof(struct icmpheader);

    char buffer[1500];
    memset(buffer, 0, 1500);

    struct ipheader *ip = (struct ipheader*)buffer;
    ip->iph_ver = 4;
    ip->iph_ihl = 5;
    ip->iph_ttl = 64;
    ip->iph_sourceip = orig_ip->iph_destip;     // 交换源IP和目标IP
    ip->iph_destip = orig_ip->iph_sourceip;
    ip->iph_protocol = IPPROTO_ICMP;
    ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader) +
icmp_data_len);

    struct icmpheader *icmp = (struct icmpheader*)(buffer + sizeof(struct ipheader));
    icmp->icmp_type = 0;
    icmp->icmp_code = 0;
    icmp->icmp_id = orig_icmp->icmp_id;
    icmp->icmp_seq = orig_icmp->icmp_seq;

    char *data = (char *)(icmp + 1);
    char *orig_data = (char *)(orig_icmp + 1);
    memcpy(data, orig_data, icmp_data_len);

    icmp->icmp_chksum = 0;
    icmp->icmp_chksum = in_cksum((unsigned short *)icmp,
                                 sizeof(struct icmpheader) + icmp_data_len);

    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;
  if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
    struct ipheader * ip = (struct ipheader *)
                           (packet + sizeof(struct ethheader));
    if(ip->iph_protocol == IPPROTO_ICMP) {
        struct icmpheader *icmp = (struct icmpheader *)(packet+sizeof(struct
ethheader)+sizeof(struct ipheader));
        if(icmp->icmp_type == 8){
```

```
            printf("Detected ICMP request from: %s to %s\n",
                       inet_ntoa(ip->iph_sourceip), inet_ntoa(ip->iph_destip));
            send_icmp_reply(ip, icmp);
            printf("Sent forged ICMP reply to: %s\n", inet_ntoa(ip->iph_sourceip));
        }
    }
  }
}

int main(){
    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp[0]==8";
    bpf_u_int32 net;
    // Step 1: Open live pcap session on NIC with name enp0s3
    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
    // Step 2: Compile filter_exp into BPF psuedo-code
    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);
    // Step 3: Capture packets
    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);   //Close the handle
    return 0;
}
```

将嗅探和伪造结合起来，运行程序

```
[07/20/25]seed@VM:~/.../volumes$ docker exec -it 98bf2448fa0c sh
# sudo ./tool
sh: 1: sudo: not found
# ./tool
sh: 2: ./tool: not found
# ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=64 time=280 ms
64 bytes from 1.1.1.1: icmp_seq=1 ttl=254 time=391 ms (DUP!)
64 bytes from 1.1.1.1: icmp_seq=2 ttl=64 time=302 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=254 time=386 ms (DUP!)
64 bytes from 1.1.1.1: icmp_seq=3 ttl=254 time=181 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=64 time=329 ms (DUP!)
64 bytes from 1.1.1.1: icmp_seq=4 ttl=64 time=348 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=254 time=392 ms (DUP!)
64 bytes from 1.1.1.1: icmp_seq=5 ttl=254 time=180 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=64 time=368 ms (DUP!)
64 bytes from 1.1.1.1: icmp_seq=6 ttl=254 time=184 ms
64 bytes from 1.1.1.1: icmp_seq=6 ttl=64 time=405 ms (DUP!)
^C
--- 1.1.1.1 ping statistics ---
6 packets transmitted, 6 received, +6 duplicates, 0% packet loss, time 5008m
rtt min/avg/max/mdev = 180.427/312.175/405.466/83.516 ms
# 
```

得到结果如下


Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000000 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
2 0.390942496 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
3 1.001702977 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
4 1.387377623 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
5 2.002314886 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
6 2.183254585 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
7 3.004263824 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
8 3.396631024 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
9 4.006415189 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
10 4.186710501 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
11 5.007766647 10.0.2.15 1.1.1.1 ICMP 98 Echo (ping) request id=0x
12 5.015089098 PcsCompu_f9:70:c2 52:55:0a:00:02:02 ARP 42 Who has 10.0.2.2? Tell 10.
13 5.015559062 52:55:0a:00:02:02 PcsCompu_f9:70:c2 ARP 64 10.0.2.2 is at 52:55:0a:00
14 5.191198261 1.1.1.1 10.0.2.15 ICMP 98 Echo (ping) reply id=0x
Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
Ethernet II, Src: PcsCompu_f9:70:c2 (08:00:27:f9:70:c2), Dst: 52:55:0a:00:02:02 (52:55:0a:00:02:02)

OVER!!!