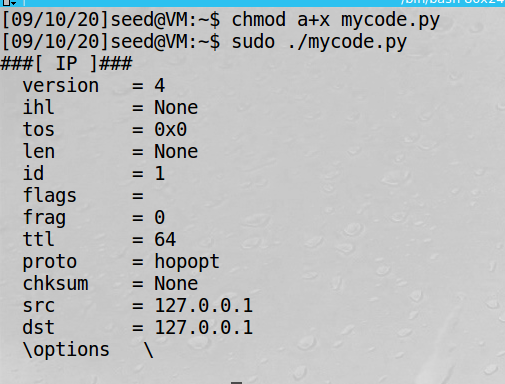
**Lab3**

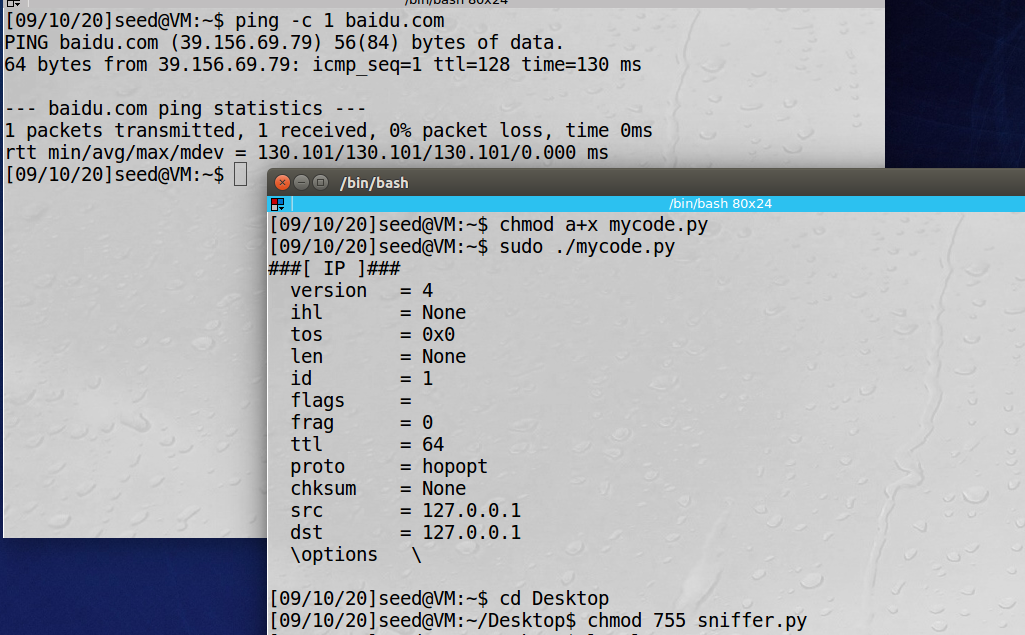
57117126张镇

Lab Task Set 1: Using Tools to Sniff and Spoof Packets

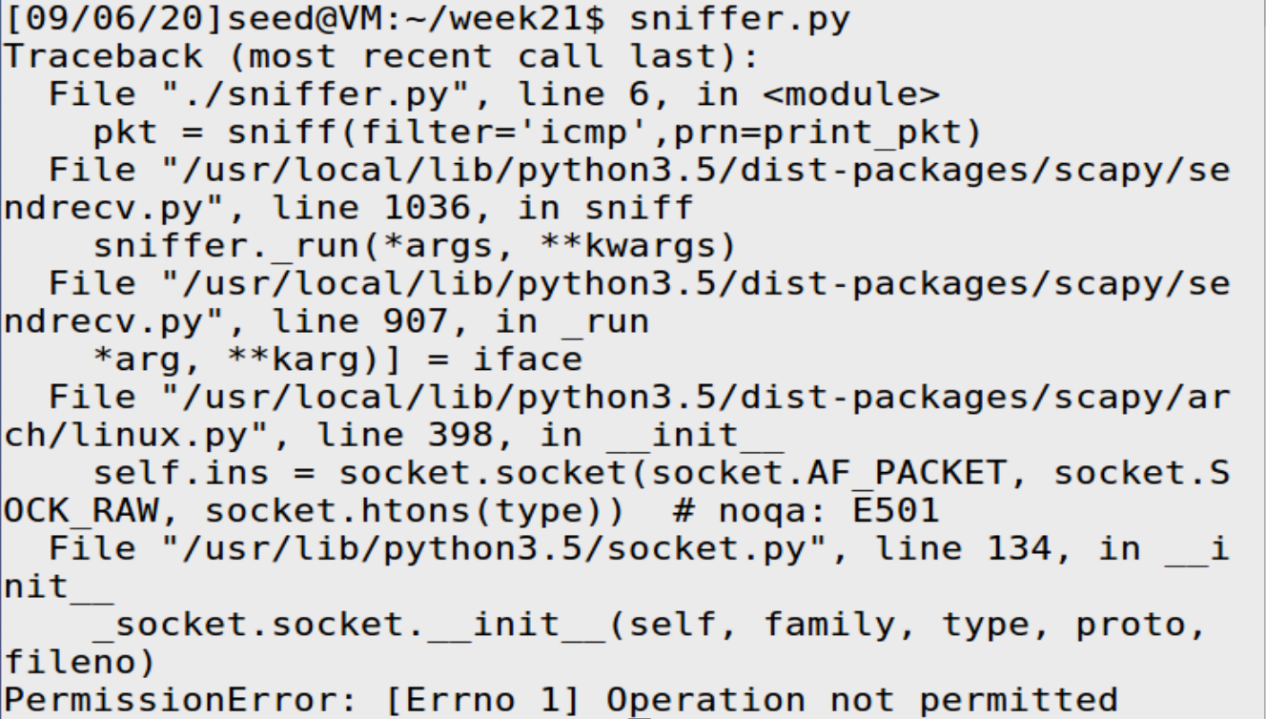


Task1.1A. SnifﬁngPackets

用管理员权限去运行sniffer.py，去ping百度。



当不以root权限运行时：



当以非root权限执行sniffer.py时，会报出“操作不被允许”的错误。

Task 1.1B

我们只需要将过滤器的值filter进行一定的改变即可

1.只捕获ICMP

**更改过滤器的值为filter =’ icmp’**，ping 0.0.0.0时，执行程序的结果为：

…………………………

###[ Ethernet ]###

dst = 00:00:00:00:00:00

src = 00:00:00:00:00:00

type = 0x800

###[ IP ]###

version = 4

ihl = 5

tos = 0x0

len = 84

id = 8068

flags =

frag = 0

ttl = 64

**proto = icmp**

chksum = 0x5d23

src = 127.0.0.1

dst = 127.0.0.1

\options \

###[ ICMP ]###

type = echo-reply

code = 0

chksum = 0x196c

id = 0x34cf

seq = 0x3c

###[ Raw ]###

load = '\xf4\x81T\_o\xa4\x0e\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'

2.监听所有来自指定 IP(另一个虚拟机)，目的端口为 23 的 TCP 连接。

首先将 sniffer.py 的 BPF 语句更改为 src host 10.0.2.5 && tcp dst port 23。 以 root 权限运行 sniffer.py，执行 ping 命令，观察是否能监听到数据包。

###[ Ethernet ]###

dst = 08:00:27:2f:a8:23

src = 08:00:27:36:0f:0c

type = 0x800

###[ IP ]###

version = 4

ihl = 5

tos = 0x10

len = 52

id = 28507

flags = DF

frag = 0

ttl = 64

proto = tcp

chksum = 0xb350

**src = 10.0.2.5**

dst = 10.0.2.4

\options \

###[ TCP ]###

sport = 49033

**dport = telnet**

seq = 3883306682L

ack = 89895857

dataofs = 8

reserved = 0

flags = A

window = 123

chksum = 0xd37

urgptr = 0

options = [('NOP', None), ('NOP', None), ('Timestamp', (387834, 423710))]

3. 监听源 ip 或目的 ip 为特定子网的数据包。选择的子网不能是虚拟机(192.168.52.133)所属的子网

**设置过滤器为filter = ‘net 58.192.0.0/16’**，使用ping指令，执行程序结果为：###[ Ethernet ]### dst = 08:00:27:2f:a8:23

src = 52:54:00:12:35:00

type = 0x800

###[ IP ]###

version = 4

ihl = 5

tos = 0x0

len = 84

id = 250

flags =

frag = 0

ttl = 248

proto = icmp

chksum = 0x45d

**src = 58.192.118.142**

dst = 10.0.2.4

\options \

###[ ICMP ]###

type = echo-reply

code = 0

chksum = 0xb1f0

id = 0xda5

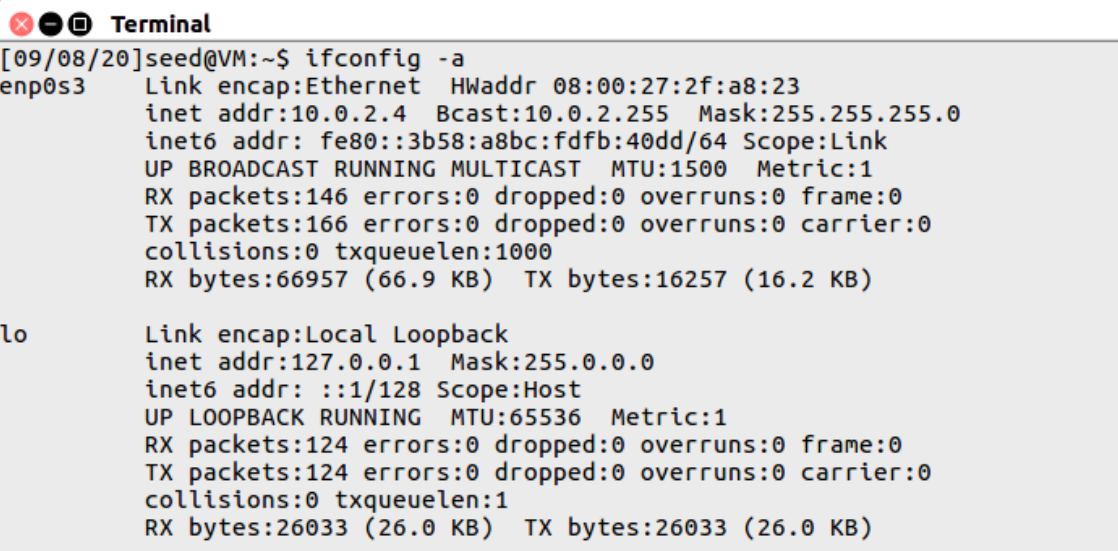
seq = 0x11

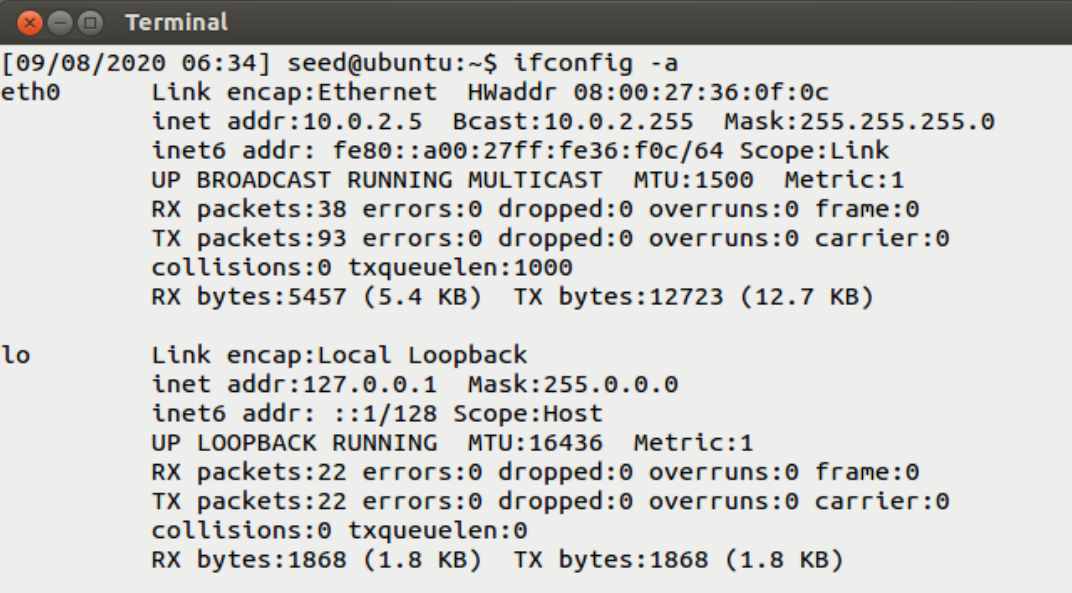
###[ Raw ]###

load = '\xb3\x8fW\_Dg\x06\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'

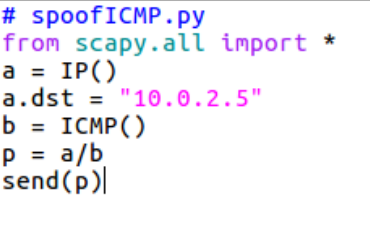
Task1.2

VM A的IP地址为10.0.2.4，VM B的IP地址为10.0.2.5。

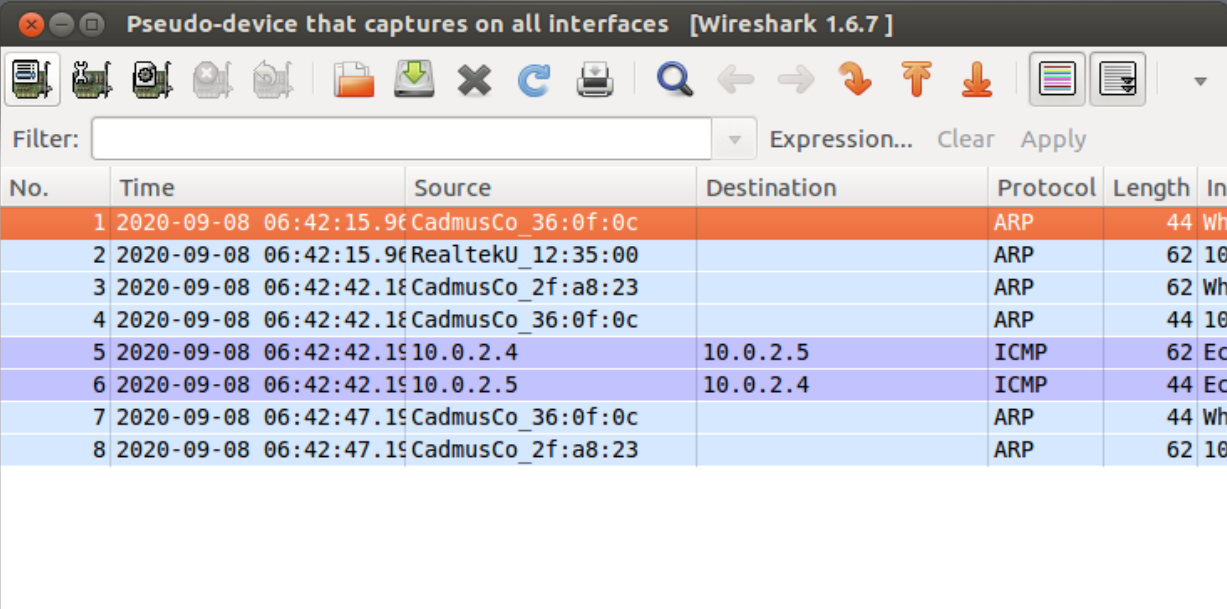




spoofICMP.py如下：



在VM B上打开wireshark准备抓包。程序执行后，向VM B发送一个报文，wireshark抓包结果如下图所示：



Task1.3

from scapy.all import \*

import sys

ip\_dst = sys.argv[1]

if len(sys.argv) < 2:

print ("[Usage]: python %s dstip" %(sys.argv[0]))

exit(1)

a = IP()

a.dst = ip\_dst

b = ICMP()

isGetDis = 0

mTTL = 1

i = 1

while isGetDis == False :

a.ttl = mTTL

ans, unans = sr(a/b)

if ans.res[0][1].type == 0:

isGetDis = True

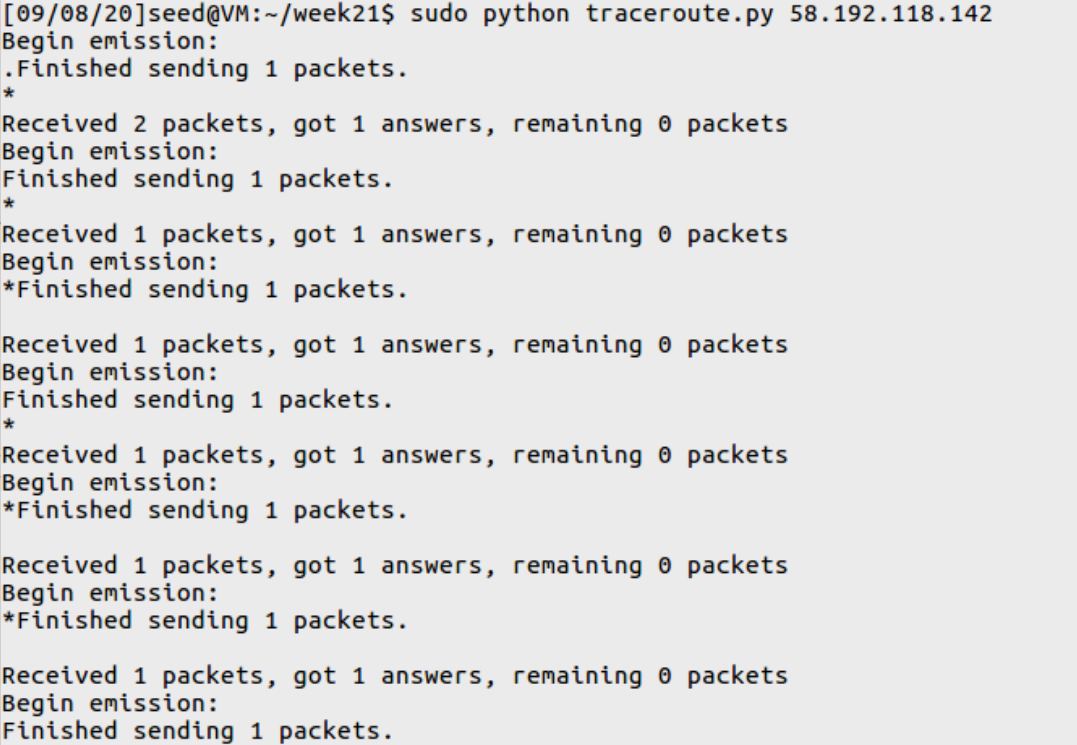
else:

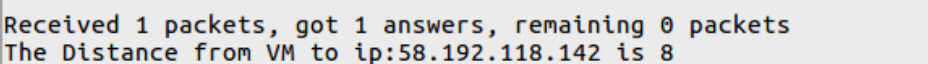
i += 1

mTTL += 1

print ('The Distance from VM to ip:%s is %d '%(ip\_dst, i))

如下图所示，当目的地址为58.192.118.142（www.seu.edu.cn的IP地址）时，需要经过8个路由器。





Task1.4

from scapy.all import \*

def print\_pkt(pkt):

a = IP()

a.src = pkt[IP].dst

a.dst = pkt[IP].src

b = ICMP()

b.type ="echo-reply"

b.code =0

b.id = pkt[ICMP].id

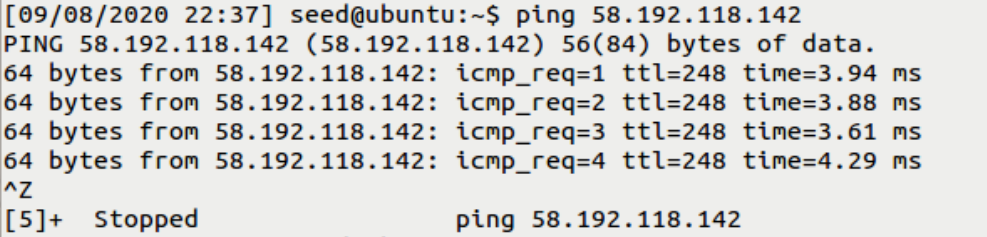
b.seq = pkt[ICMP].seq

p = a/b

send(p)

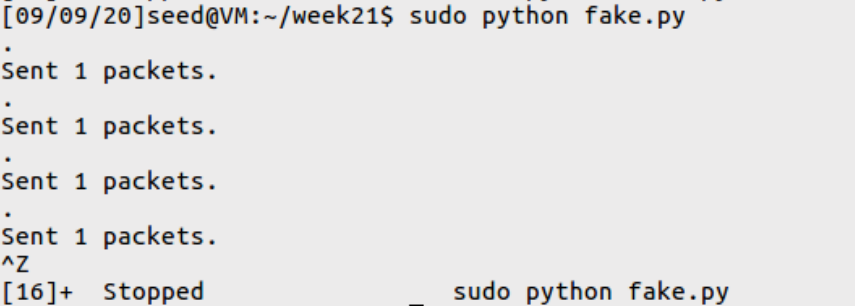
pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=print\_pkt)

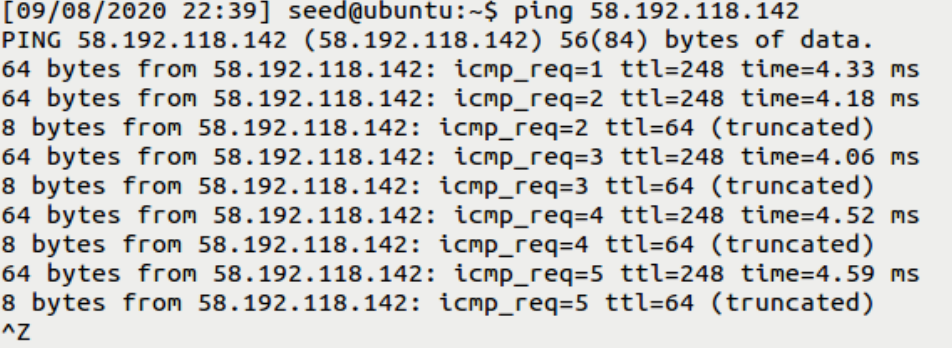
在VM A上不运行fake.py时，VM B ping 58.192.118.142（www.seu.edu.cn的IP地址）的结果如下：



可以看到，VM B成功收到了来自58.192.118.142（www.seu.edu.cn）的响应报文。

在VM A上运行fake.py后，VM B再次ping 58.192.118.142，VM A和VM B的响应如下：（第一张为VM A，第二张为VM B）





可以看到，除了真正来自58.192.118.142的ICMP响应报文外，VM B还额外收到了（截断的）4个8字节的ICMP响应报文，刚好对应于第一张图中VM A伪造的4个ICMP响应报文。

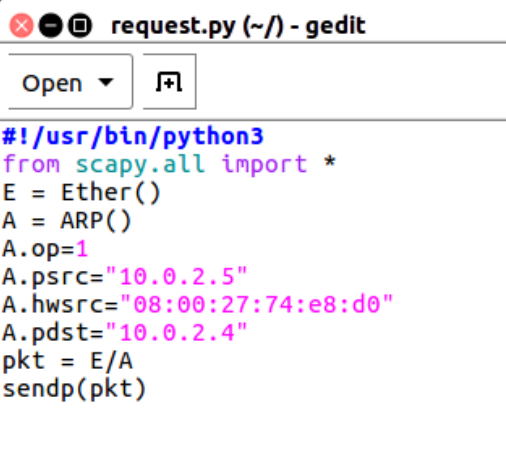
所以，利用Scapy，无论IP X是否在线，都能够收到伪造的源地址为X的ICMP响应报文。

Part2

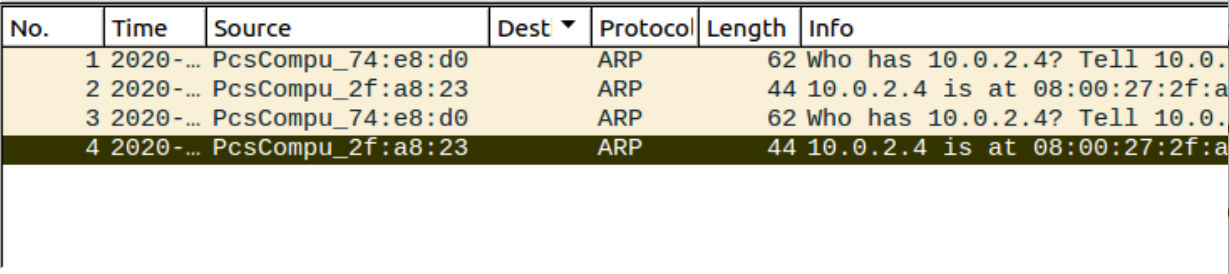
Task1A

在主机M上构造ARP请求包并发送到主机A。检查A的ARP缓存，看看M的MAC地址是否映射B的IP地址。

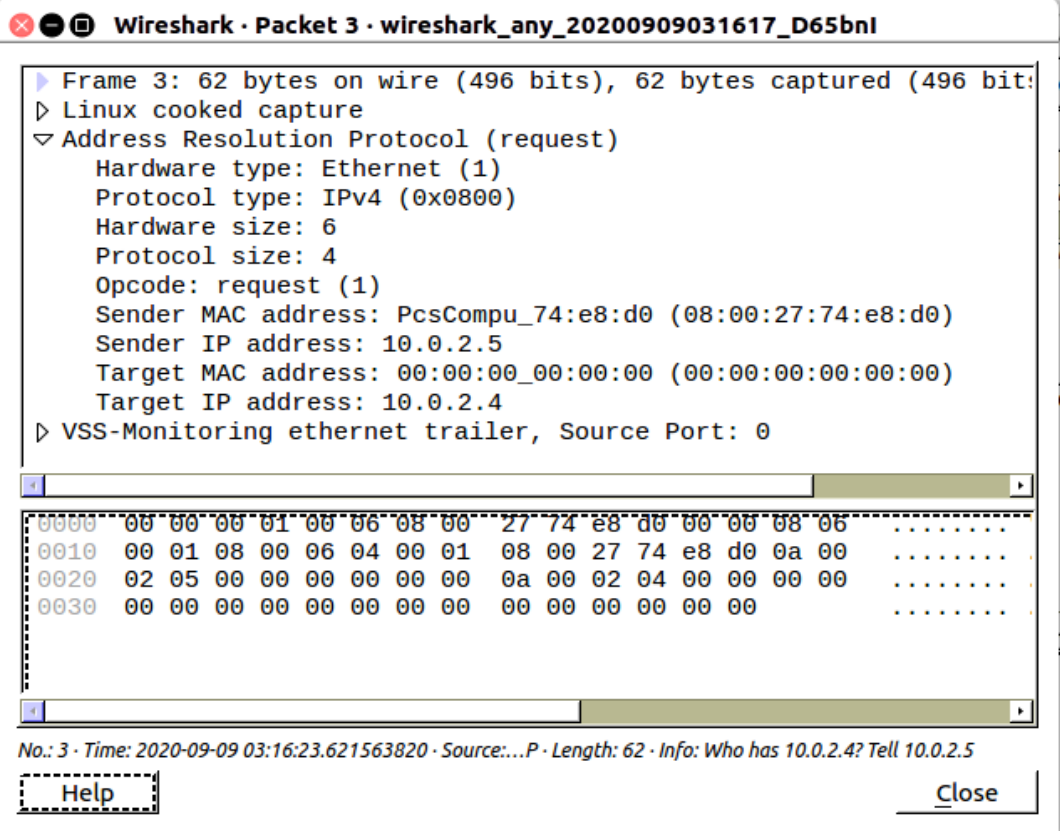
首先写一个py程序如下图所示，为了使VM A存入B的IP地址--> M的MAC地址的错误映射，应当构造一个源IP地址为B的IP（psrc=”10.0.2.5”），源MAC地址为M的MAC地址（hwsrc="08:00:27:74:e8:d0"），目的地址为A的IP地址（pdst="10.0.2.4"）的报文。报文的op选项设为1，表示为请求报文；



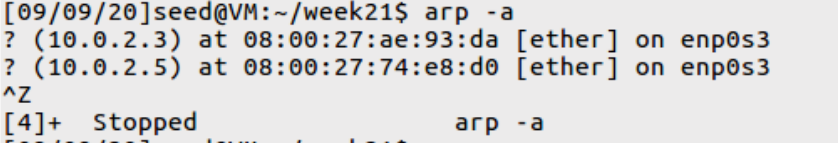
在虚拟机上打开WIRESHARK进行抓包



可以看到，VM A上可以抓到来自10.0.2.5且MAC地址08:00:27:74:e8:d0的ARP请求报文，具体信息如下图所示：



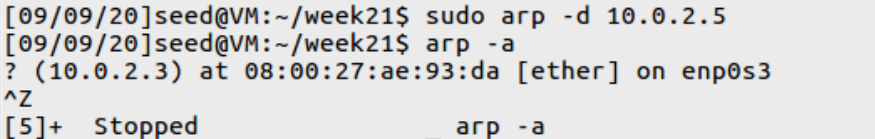
尽管10.0.2.5真实对应的MAC地址是B的MAC地址，但是这里的ARP请求报文的源MAC地址成功映射为M的MAC地址08:00:27:74:e8:d0。用ARP-A指令查看ARP缓存



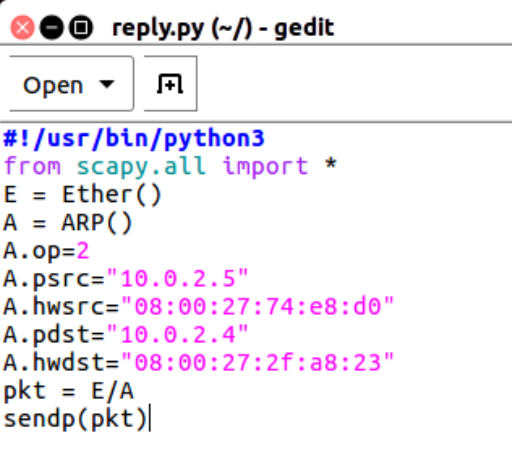
可以看到，A的ARP缓存中成功将B的IP地址与M的MAC地址映射，ARP缓存污染攻击成功。

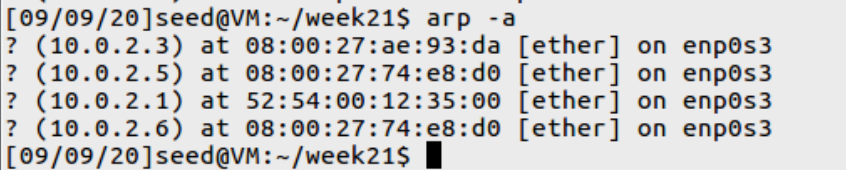
Task1B

在主机M上，构造ARP应答包并发送到主机A。检查A的ARP缓存中，M的MAC地址是否映射到B的IP地址。清除上面的残留缓存



构造一个ARP报文，响应报文的op选项设为2，表示为响应报文；源IP地址为B的IP地址，源MAC地址为M的MAC地址；目的IP地址为A的IP地址，目的MAC地址为A的MAC地址，**伪装为A的ARP请求报文的响应报文，发送给A。**

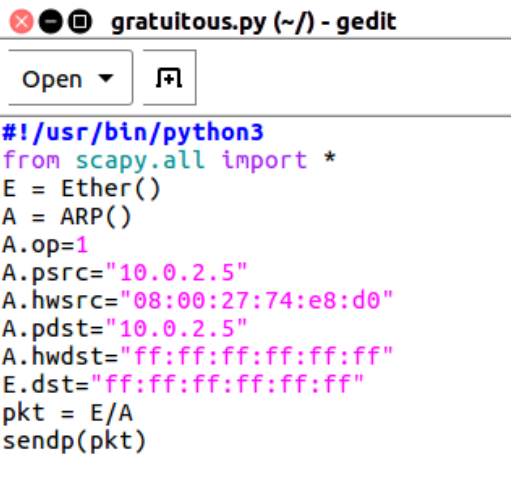


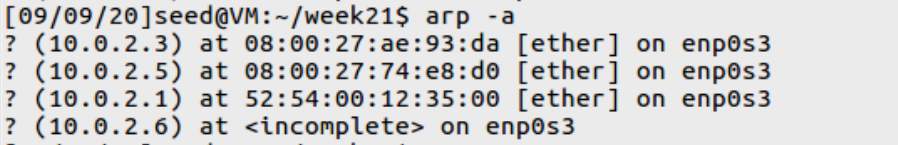


可以看到，A的ARP缓存中成功将B的IP地址与M的MAC地址映射，ARP缓存污染攻击成功。

Task1C

清楚VM A的ARP缓存后，在VM M上写程序，发送一个gratuitous请求报文。报文的op选项设为1，表示为请求报文；源IP地址为B的IP地址，源MAC地址为M的MAC地址；目的IP地址为B的IP地址，目的MAC地址为广播地址（ff:ff:ff:ff:ff:ff）。并且报文的以太网报头的目的MAC地址也设置为广播地址（ff:ff:ff:ff:ff:ff）。





可以看到，A的ARP缓存中成功将B的IP地址与M的MAC地址映射，ARP缓存污染攻击成功。

Prat3

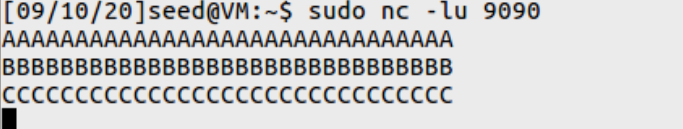
Task1A

编写UDP数据包构造程序，并编译运行

task.py内容如下：



在另一个虚拟机上开启UDP服务器监听UDP报文



如上图所示，VM M的UDP服务器可以收到来自VM A的IP报文，报文内容为31个A和一个“\n”，31个B和一个“\n”，31个C和一个“\n”，刚好96个字节，所以攻击成功。

Task1B

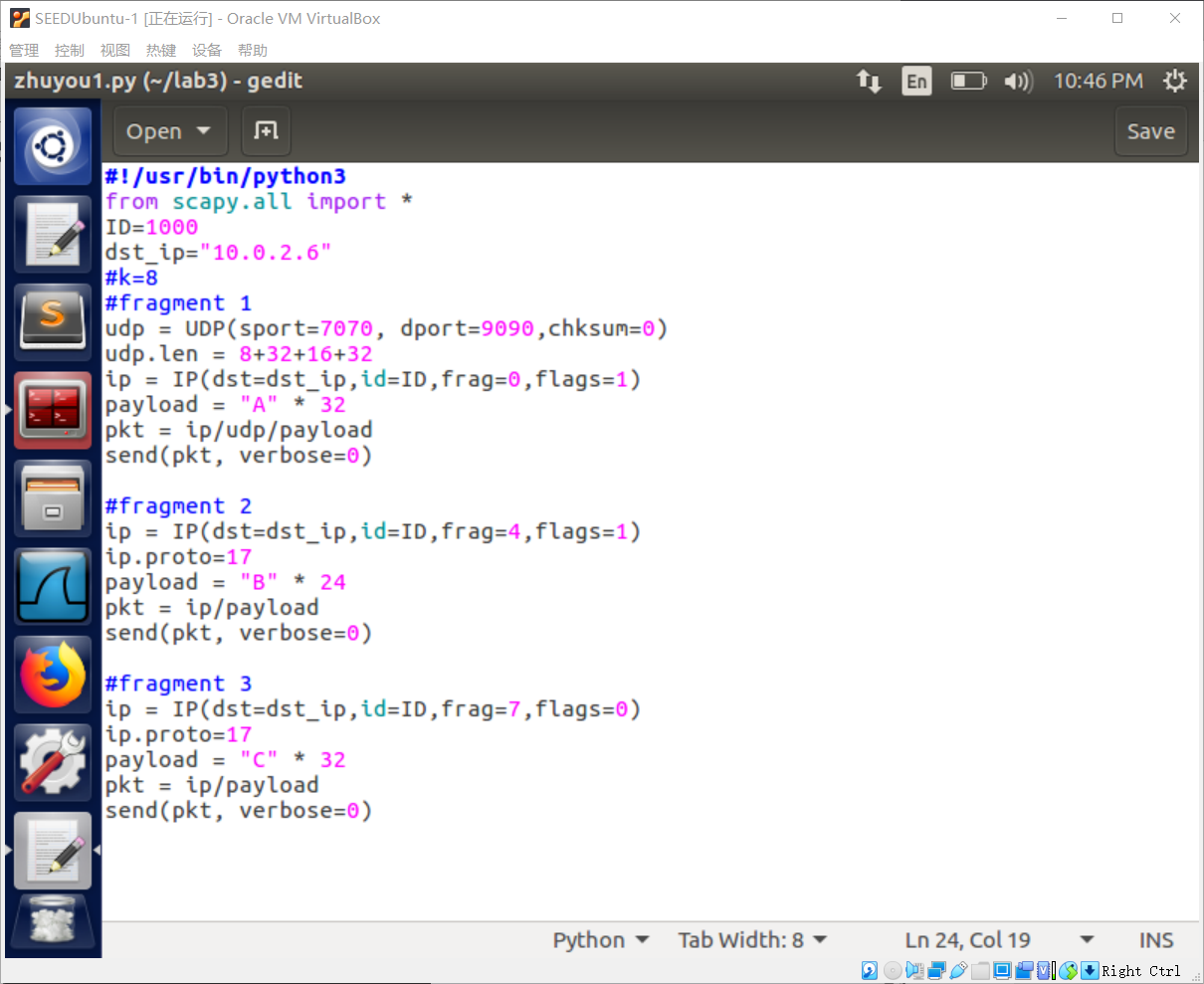
假设K=8，第一个和第三个分片数据部分为32字节，第二个分片的数据部分为24节，UDP头部为8字节，总共三个分片。

如果不修改长度



[H}E%2K3C9SJ~ZKWGC4BCPM**由于分片之间有重叠，所以在不修改UDP数据包不会交付给UDP服务器。**

**如果修改长度：**





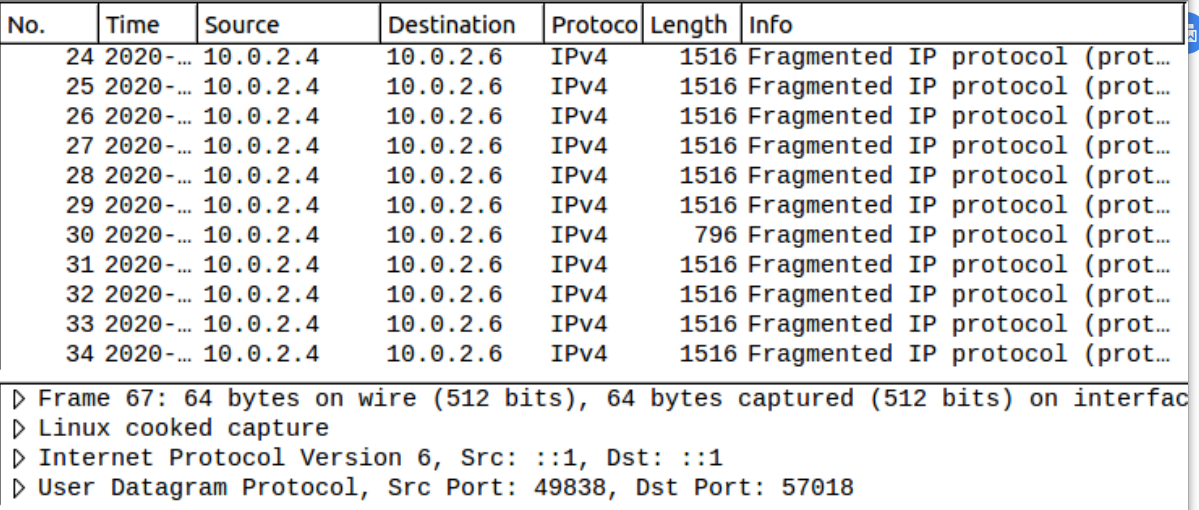
UDP服务器端丢弃分片2与分片1的重叠部分，把三个分片不重合的部分重组，所以打印结果为32个A和16个B和32个C。

Task1C

构造一个有三个分片的IP报文，三个分片的数据长度分别为20000，20000,25600字节，将报文发送给UDP服务器。



在VM M（UDP服务器端）利用wireshark抓包，可以看到，由于数据链路层所能承受的最大传输单元（MTU）为1500字节，所以对于长度超过1500字节的IP分片，会自动将其拆分为更小的分片。

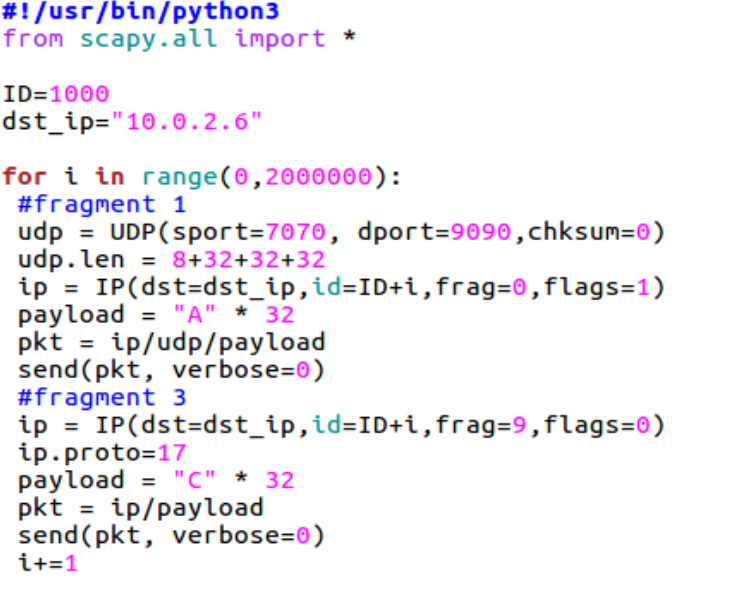


在传输层将所有分片整合后，发现整个UDP报文的长度已经大于UDP服务器端不会打印输出65535字节，超过了UDP的长度限制，所以UDP服务器端不会打印输出。

)AV8I`Z}]N)3IZKI`Y78]8W

Task1D

编写程序task4，如下图所示，VM A向VM M发送大量不完整的IP报文，并且每个IP报文都缺少一定分片。



用wireshark抓包，如下图所示，VM M上抓到了数量极大的不完整的IP数据包，可以观察到虚拟机M的运行速度降低，虽然没能直接体现为拒绝服务，但是已经可以体现不完整的IP报文对虚拟机内核内存的消耗。

