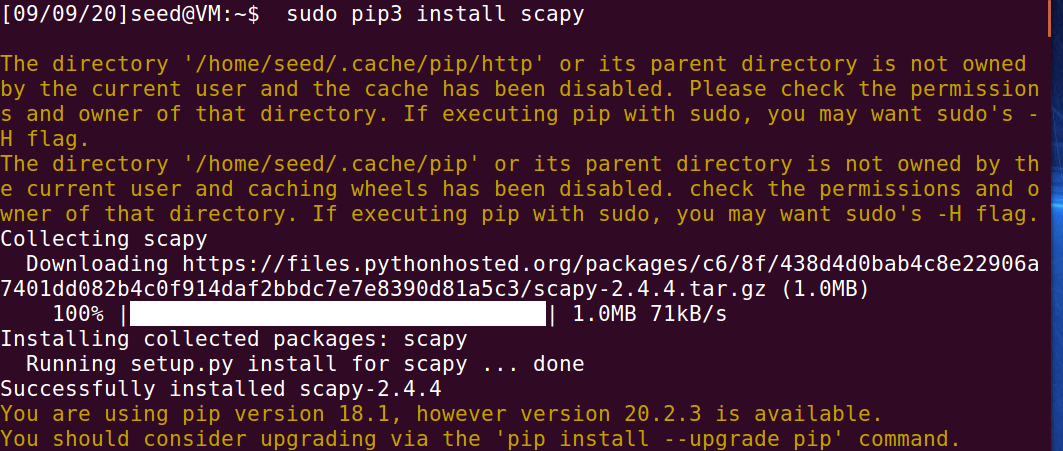
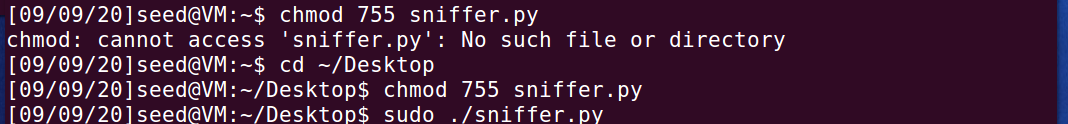
**Network Security**

**Task 1.1: Sniffing Packets**

安装scapy

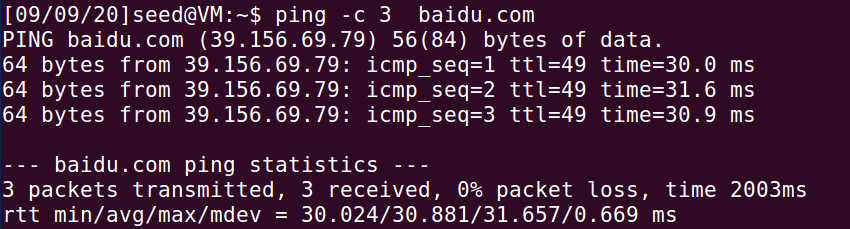


编写程序，监听icmp协议包并设置为可执行程序

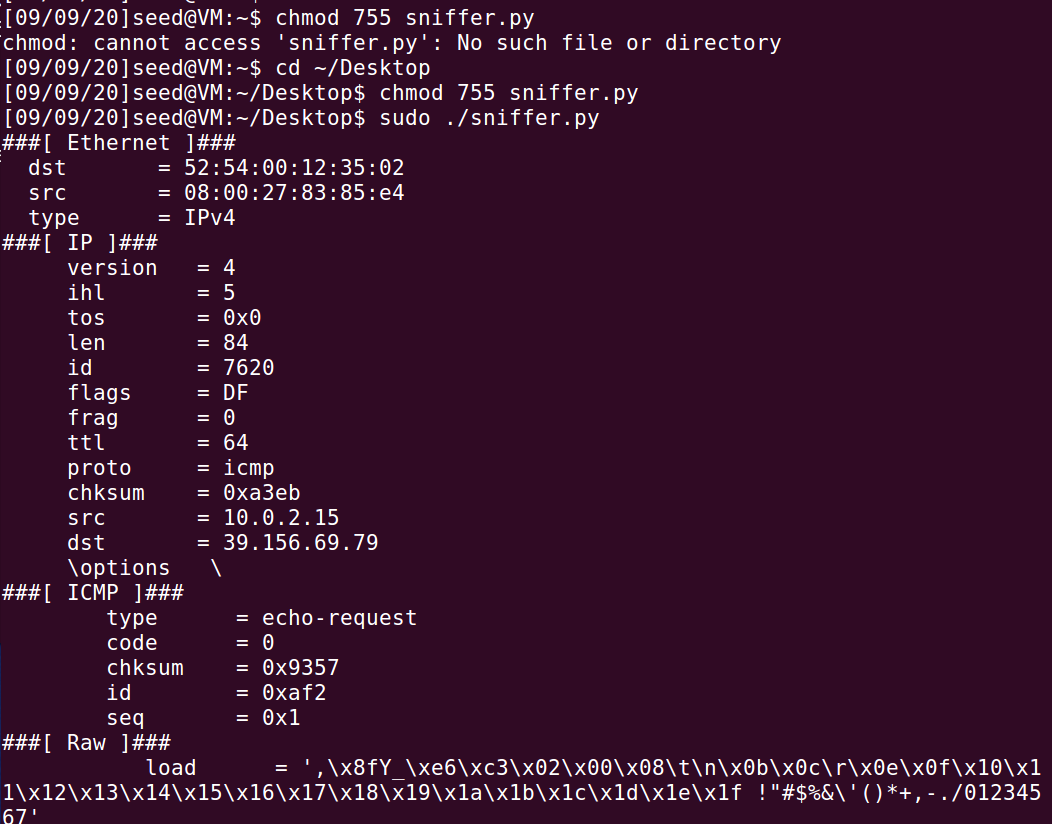


**TASK 1.1A**

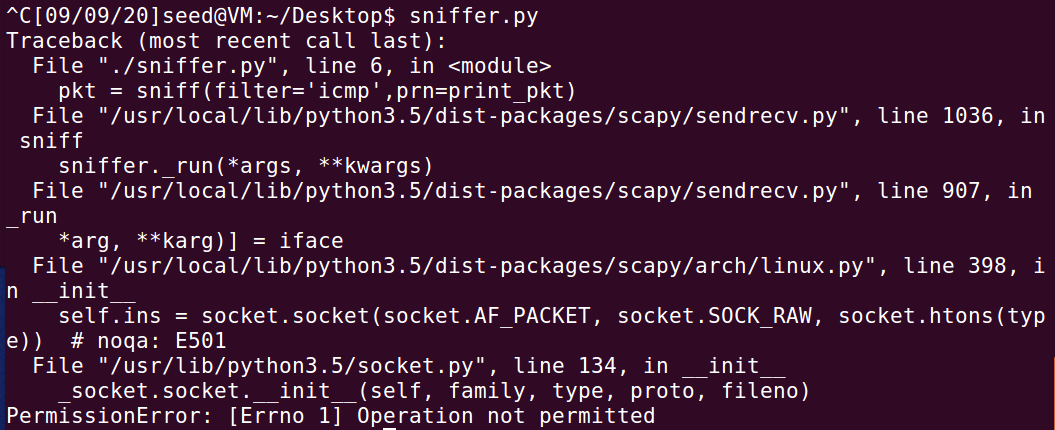
用管理员权限运用sniffer.py，并在另一个终端中发起ping命令。



监听两个数据报：请求报文和响应报文，报文从上至下依次为：以太协议、IP协议、ICMP协议

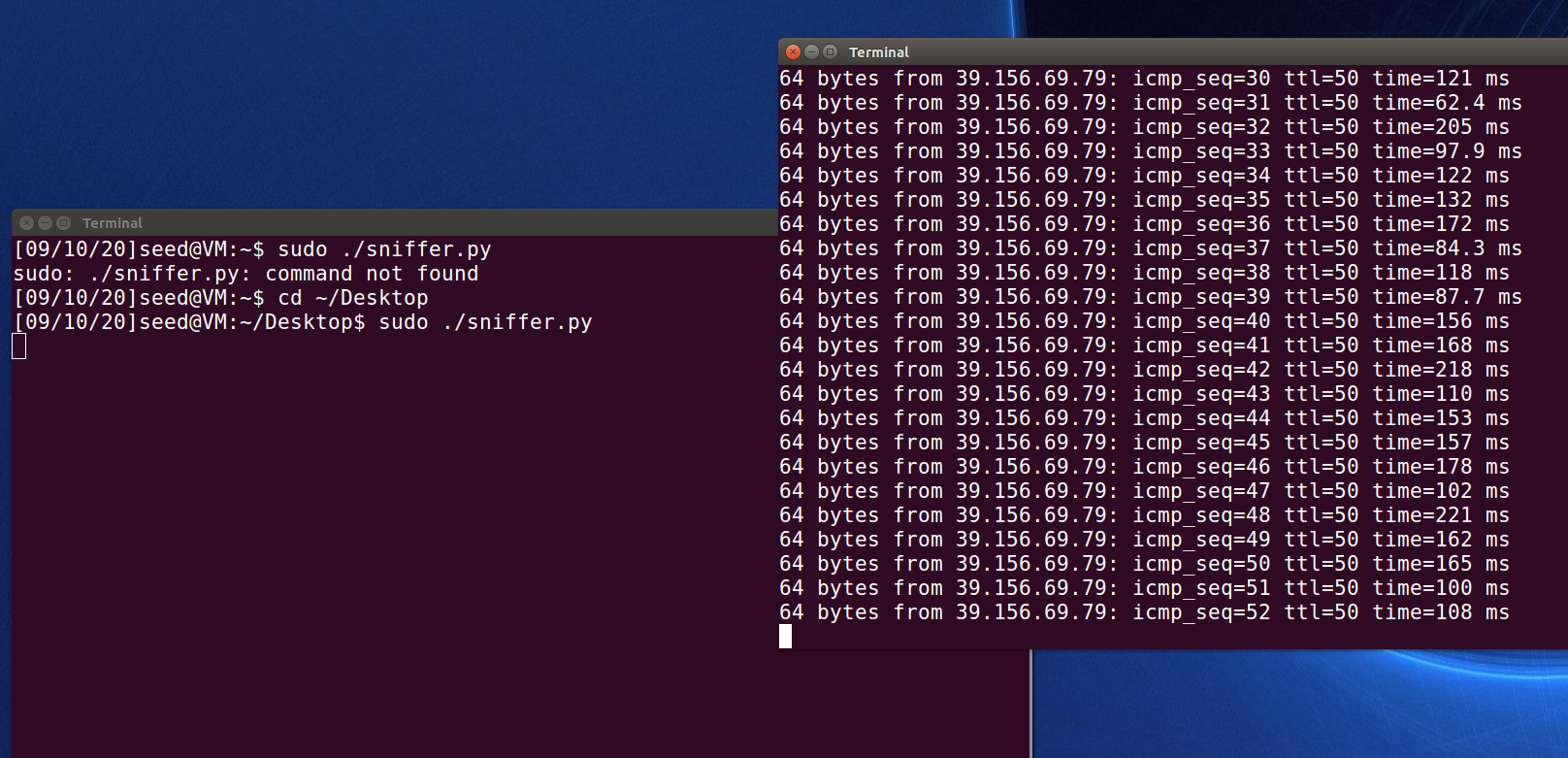
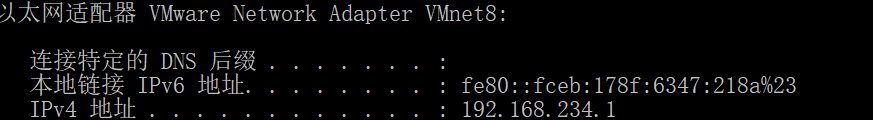


用普通权限运行sniffer.py，提示Operation not permitted，说明权限不足。无法启动监听功能

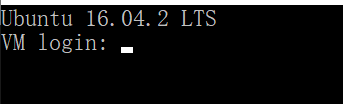


**Task 1.1B**

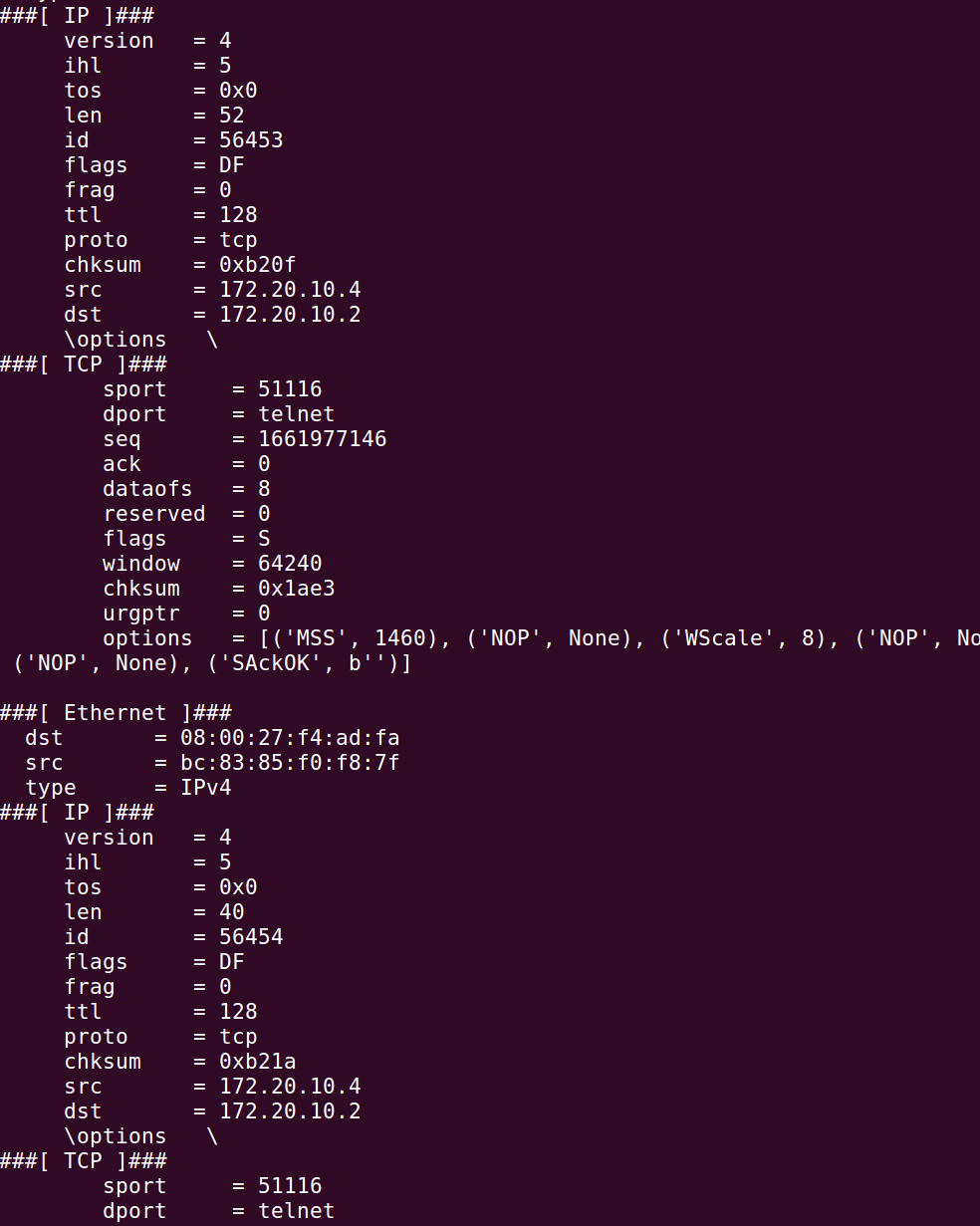
1. 仅监听 icmp 报文，与上面的实验一致，不再重复
2. 查看物理机ip，监听所有来自该ip，目的端口为23的tcp连接。首先将sniffer.py的BPF语句更改为如下图所示。以root权限运行sniffer.py，执行ping命令。没有监听到数据包。

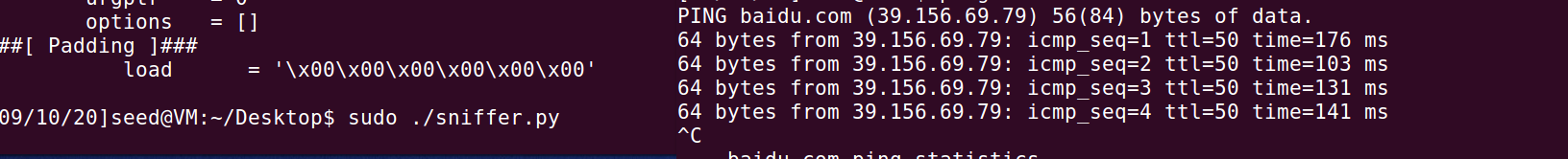


在物理机打开telnet功能，对虚拟机发起telnet连接请求。默认端口为23

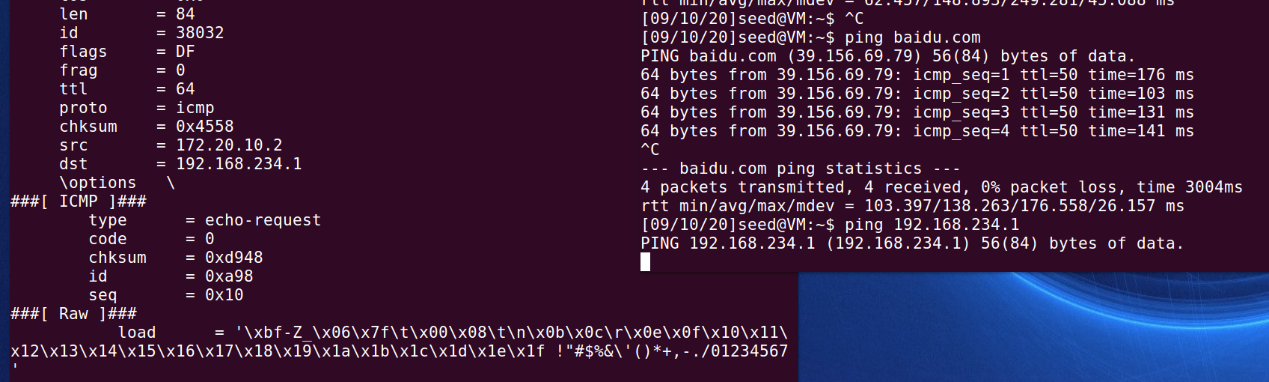


Sniffer.py监听到了多个telnet数据包。

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所属的子网。对百度发起ping请求，监听端没有监听到数据包。



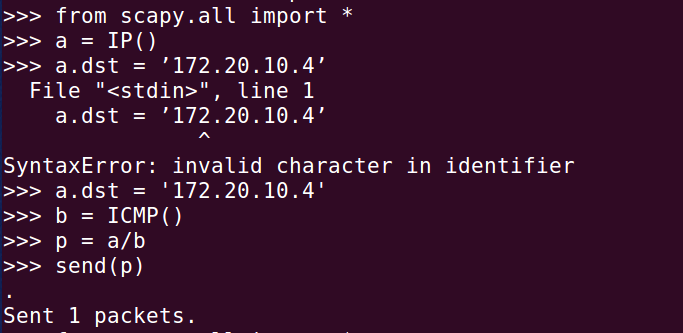
对选定的子网内的地址发起ping请求。

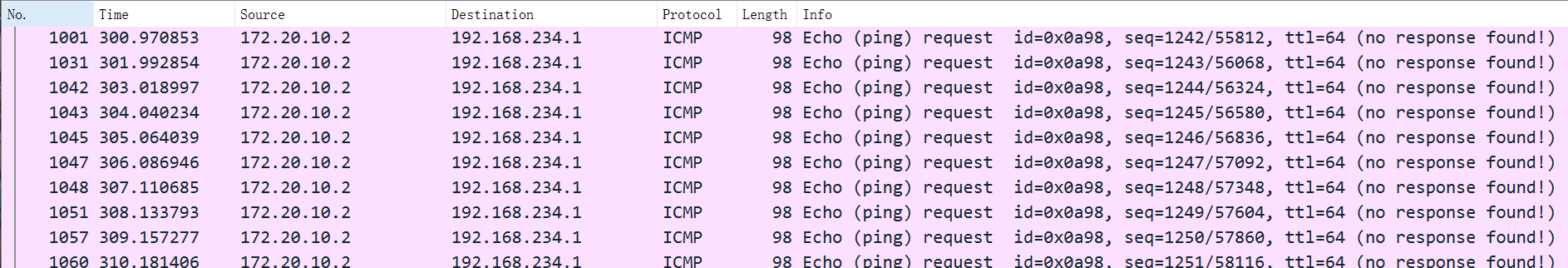


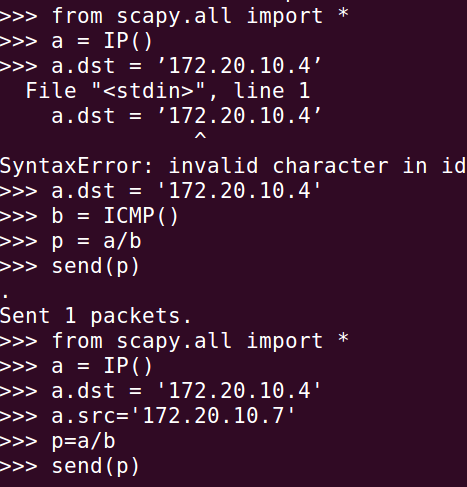
成功监听到。

**Task 1.2: Spoofing ICMP Packets**

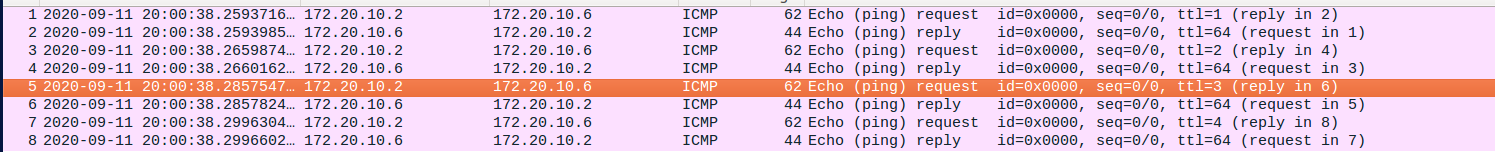
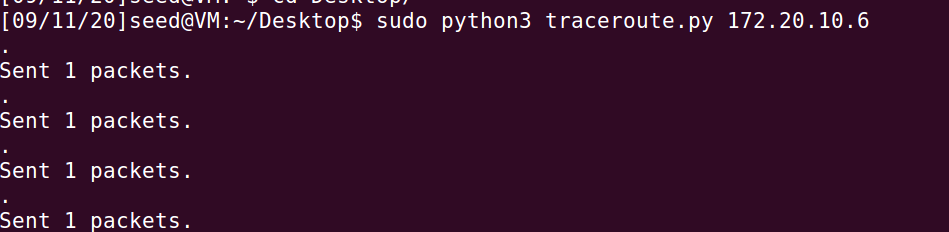
本实验利用scapy构造一个伪造的数据包：从虚拟机向物理机发送icmp报文

****

****

****

****

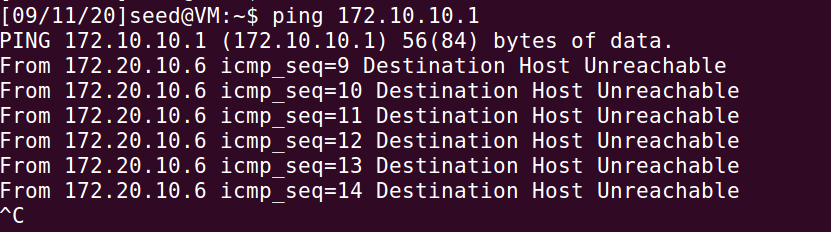
****

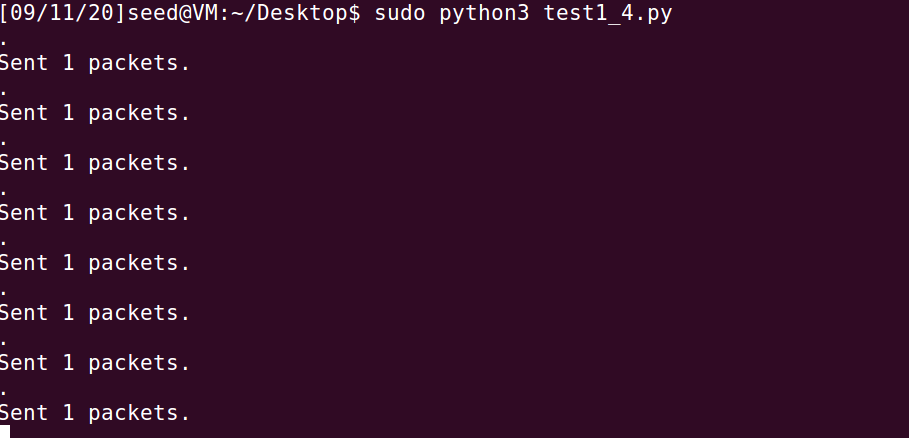
伪造ip成功。真实地址为172.20.10.2 伪造后的地址为172.20.10.7

**Task 1.4: Sniffing and-then Spoofing**

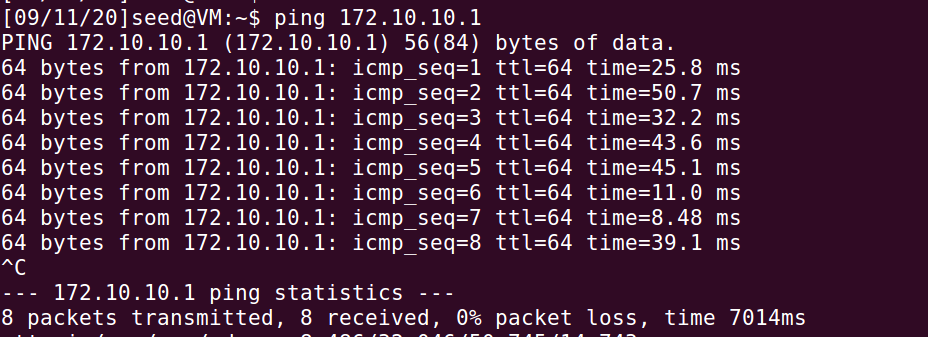
结合监听和伪造技术来实施攻击。

开始时，无法ping这个编造的地址。

****

****

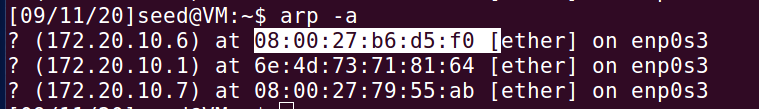
在另一个虚拟机中运行代码后，发现可以ping通。这说明伪造回复报文成功。

****

ARP Cache Poisoning Attack Lab

**Task1: ARP Cache Poisoning**

**查看宿主机中的arp列表**

****

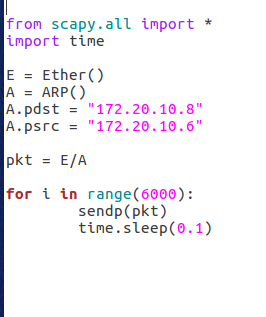
**Task 1A (using ARP request)：**

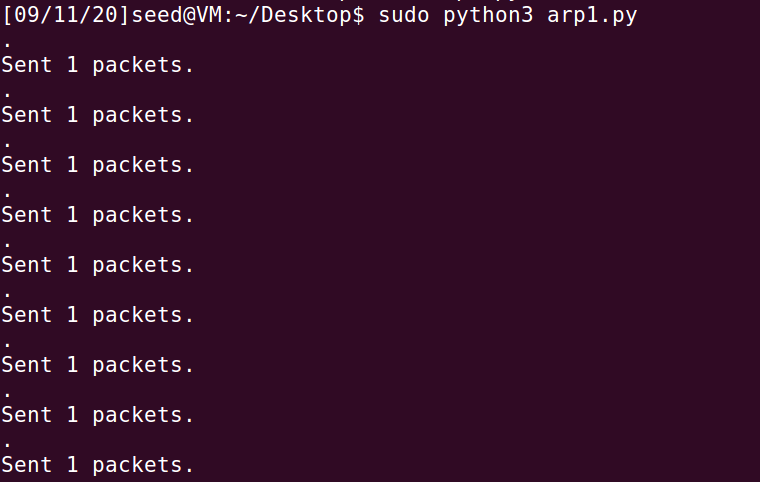
目标是把爱172.20.10.6对应的mac地址污染。

准备三台主机：目标主机A（172.20.10.8）主机B（172.20.10.6 08:00:27:b6:d5:f0）主机c（172.20.10.7 08:00:27:79:55:ab）

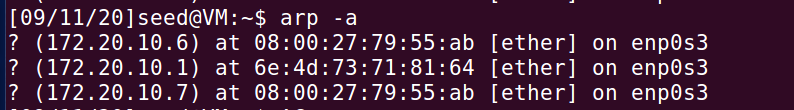
污染目标是把主机A中的主机B的ip地址对应主机c的mac地址

Step1：用主机c构造主机b的报文，发送给主机A

****

****

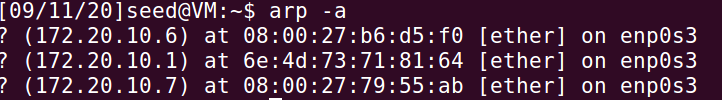
此时查看主机A的arp表，发现172.20.10.6已经被172.20.10.7主机污染

****

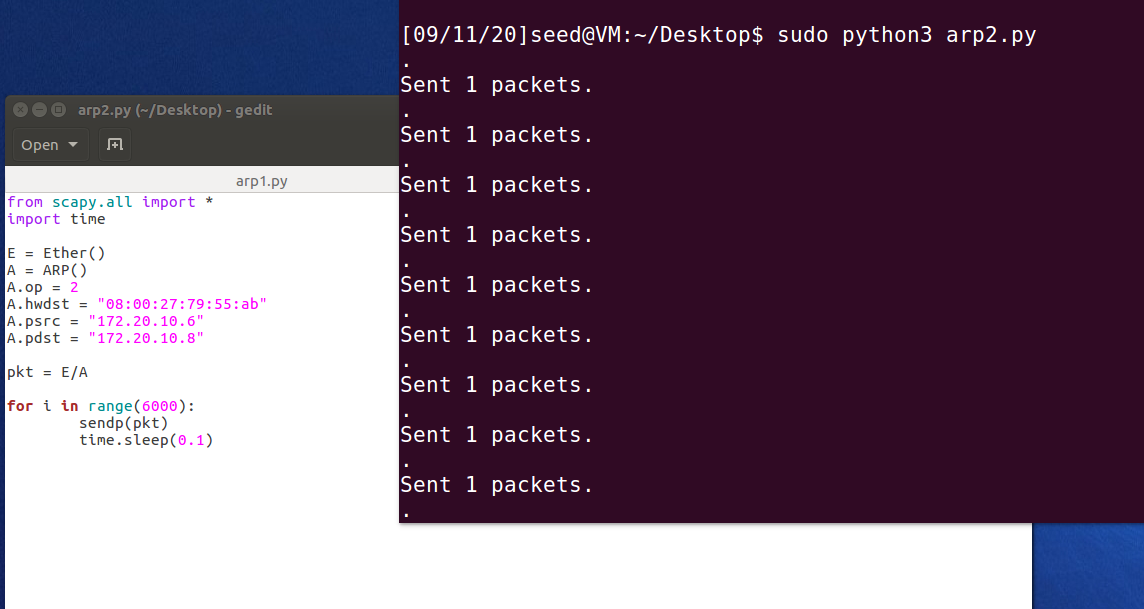
Step2：用172.20.10.6主机再多发送几次报文，刷新arp，为下次实验做准备

**Task 1B (using ARP reply)：**

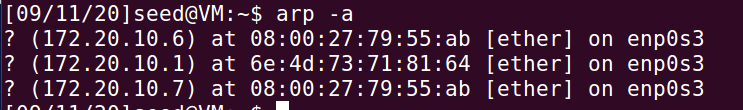
**刷新后的arp**

****

**构造arp响应包并不断向目标主机发送**

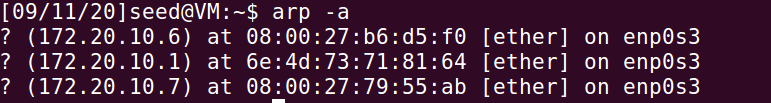
****

**查看目标主机的arp列表发现再次被污染**

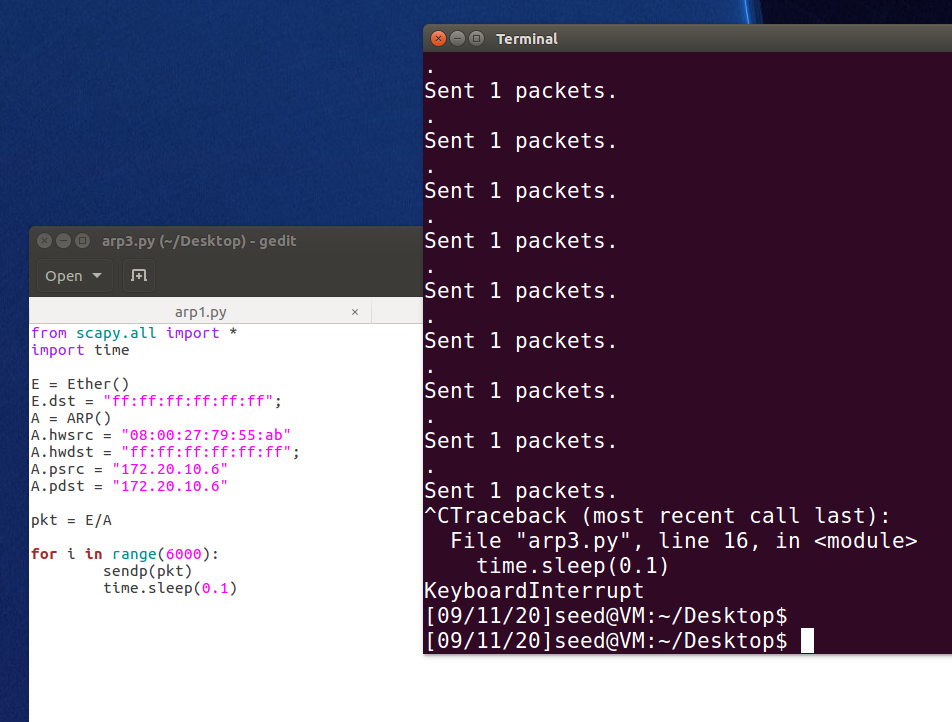
****

**Task 1C (using ARP gratuitous message)：**

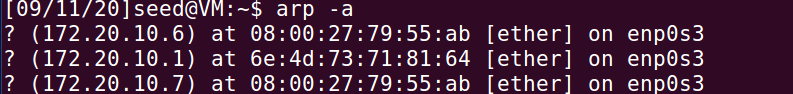
刷新arp列表

****

**持续广播ARP gratuitous报文**

****

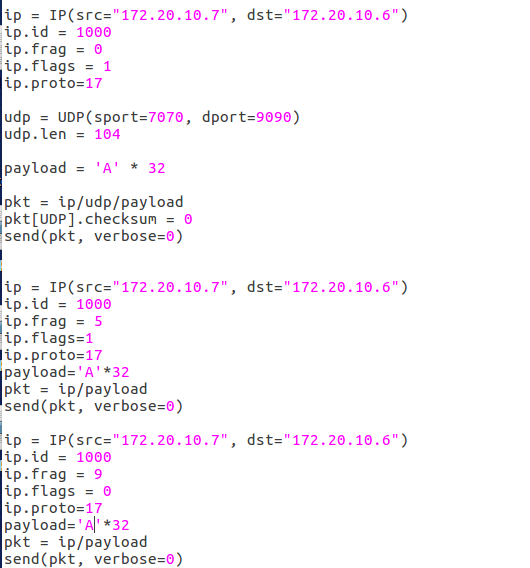
**再次查看目标主机arp列表，发现已经被感染**

****

IP/ICMP Attacks Lab

**Tasks1: IP Fragmentation**

**Task1.a: Conducting IP Fragmentation**

****

**将UDP报文分片，构造3个连续的数据包。**

**第一个包，frag=0，flags=1，说明后面还有其他分片。**

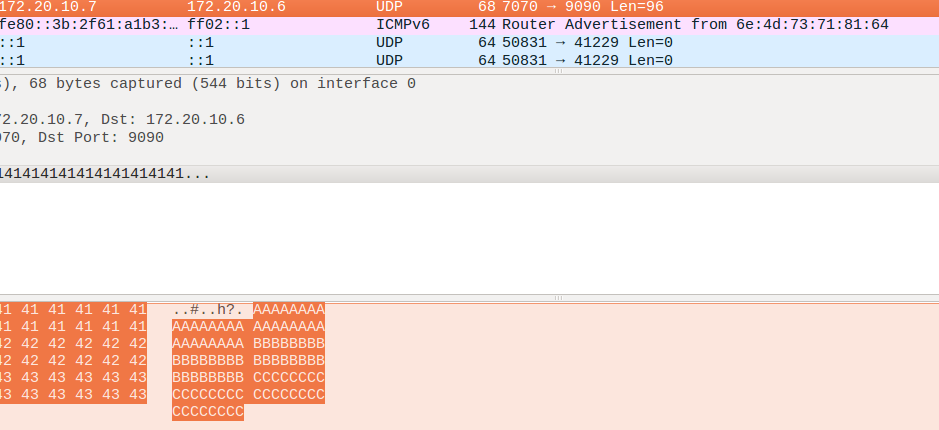
**第二个包，frag=5，flags=1**

**第三个包，frag=9，flags=0，表明不再有分片**

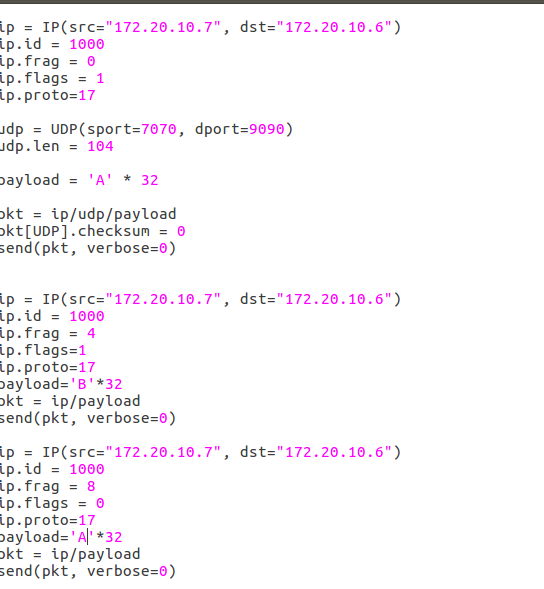
**在172.20.10.7中运行该脚本**

****

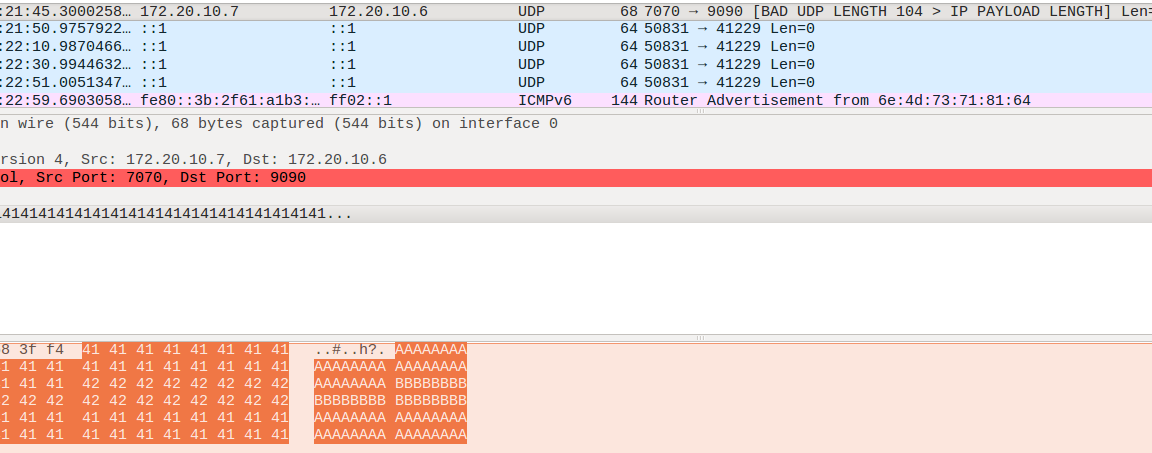
**在172.20.10.6中监听9090端口。我们发现它将收到的3个包成功能重组。**

****

**Task 1.b: IP Fragments with Overlapping Contents**

****

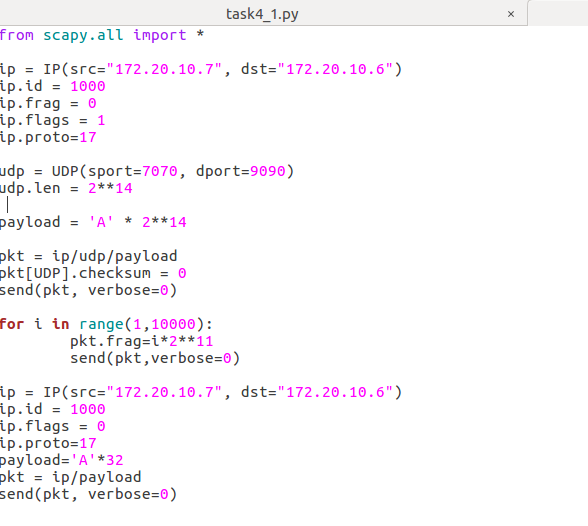
在172.20.10.6端监听，收到32个A，24个B，32个A。说明第二个包的前8个字节被盖住了。



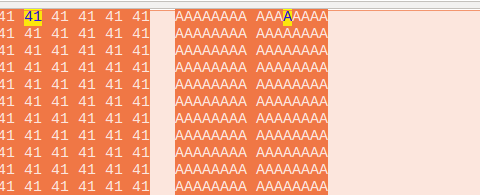
交换第二片IP报文与第一片IP报文的发送顺序，得到的结果与前面一致。这是因为，重组报文是在接收到所有报文后，才开始进行。

**Task 1.c Sending a Super-Large Packet**

编写脚本，不断的向UDP服务器发送长度为2\*14的分片。设置其len字段为0xFFFF。当分片长度超过0xFFFF后，设置其flag=0。

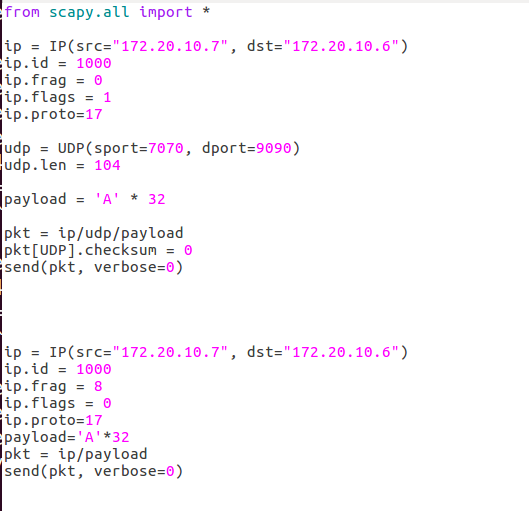


编写脚本，不断向UDP服务器发送长度为2^14的分片。由于报文过长，这里没有等到最后一个报文结束，猜测UDP报文接收端崩溃。



**1.d Sending Incomplete IP Packet**

改写脚本，不再发送第二片分片，而是只发送第一片、第三片，并不断改变id：



通过这种方案，尽管wireshark抓包没有异常，但是服务器的内存占用急剧升高。