**Lab3实验报告**

57117138 吴伊杰

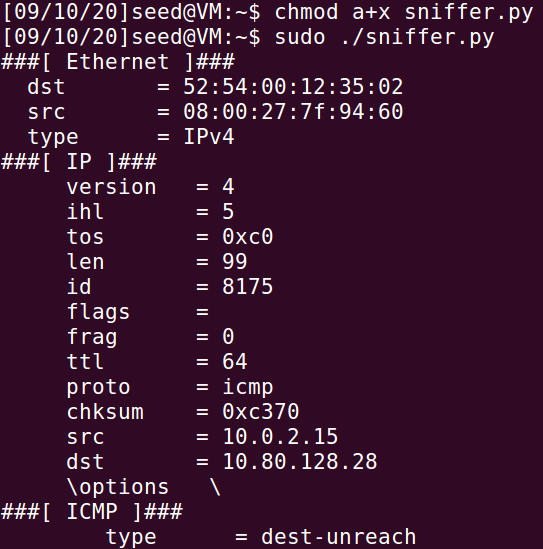
**（一）Packet Sniffing and Spoofing Lab**

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

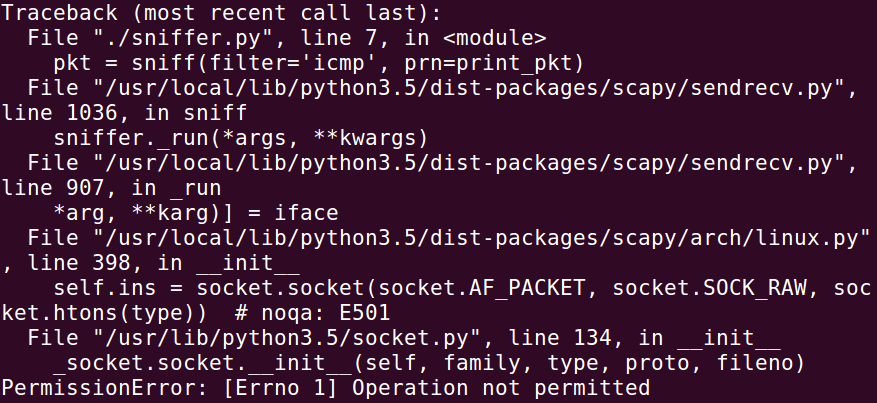
* **Task 1.1 Sniffing Packets**

**Task 1.1A**

将所给程序保存为sniffer.py，执行chmod a+x sniffer.py命令，然后先用sudo命令执行，执行成功（部分结果如图）：



若以普通用户权限运行，则报错：



报错的原因在于socket操作没有得到许可。

**Task 1.1B**

按照不同的要求进行不同的设置：

* **Capture only the ICMP packet**

filter=’icmp’

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

filter = ‘ip src 10.208.66.81 and tcp and dst port 23’

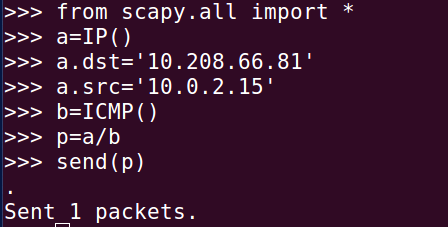
（10.208.66.81为我的宿主机ip地址）

* **Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.**

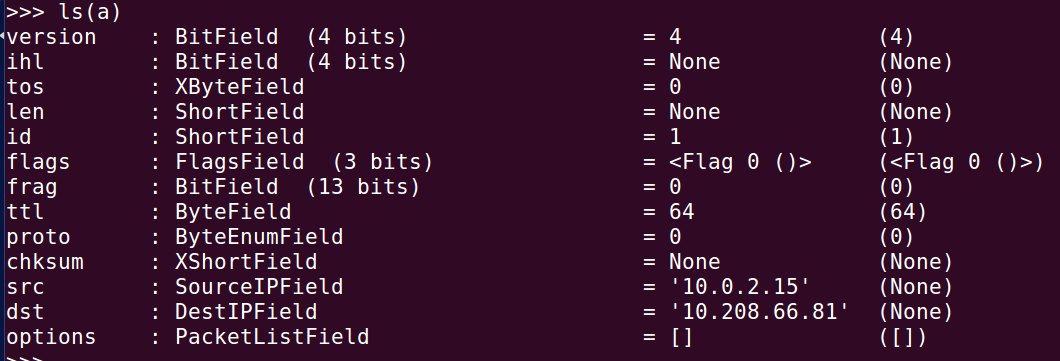
filter = ‘net 128.230.0.0/16’

* **Task 1.2 Spoofing ICMP Packets**

修改a.src，修改后的程序如下：



使用ls(a)命令查看属性：

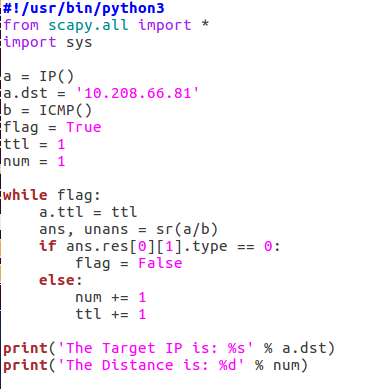


使用Wireshark查看，伪造成功：

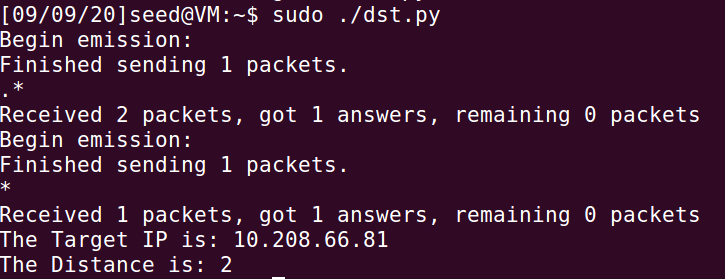


* **Task 1.3 Traceroute**

根据所给代码提示，写Python代码以计算虚拟机到宿主机的距离，代码如下：

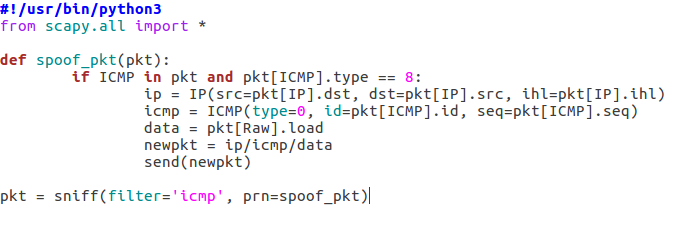


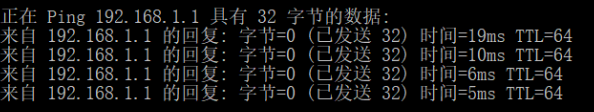
运行结果：



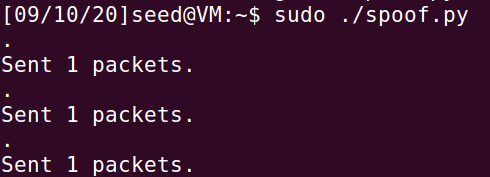
* **Task 1.4 Sniffing and-then Spoofing**

任务的目的是将ICMP报文的源地址和宿地址互换，然后发送形成伪造攻击。代码如下：



运行此代码，然后在宿主机的命令行内ping一个伪造的地址如192.168.1.1，本应无法ping通，但此时却ping通了：

虚拟机上的程序也有了结果：

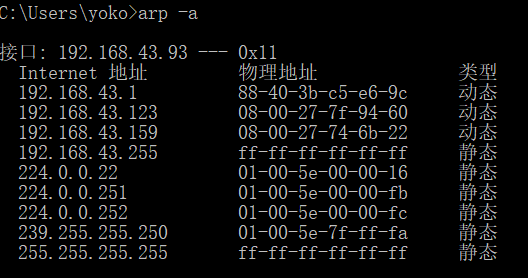


**（二）ARP Cache Poisoning Attack Lab**

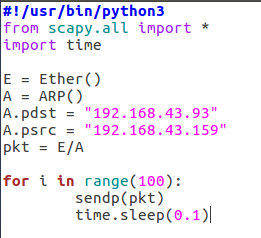
# Task 1: ARP Cache Poisoning

* **Task 1A (using ARP request)**

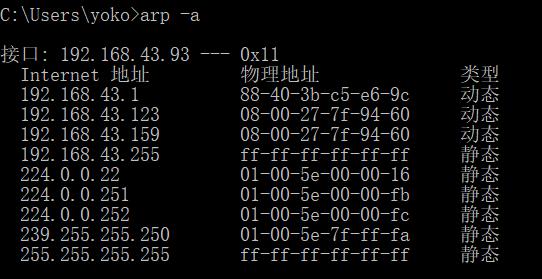
使用ARP请求污染。首先在宿主机（下面称为A）中使用arp -a命令查看ARP缓存表（主机和虚拟机中应当事先通信过后才会有记录）：



这里我们攻击的目标IP地址是（下面称为B），其MAC地址为，写发送ARP请求报文的代码，pdst和psrc的地址分别为A和B的IP地址，然后执行，就能让A不断向B发送ARP请求报文，这样就能将其污染。



运行程序，发送了大量包后，用arp -a命令查看：



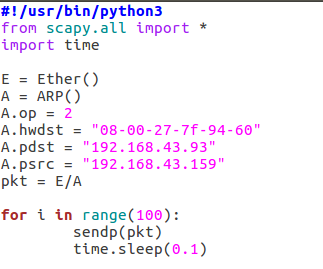
此时发现主机B的MAC地址变成了0，污染成功。

此时再让B向A发起几次ping通信，可以把ARP缓存表刷新。

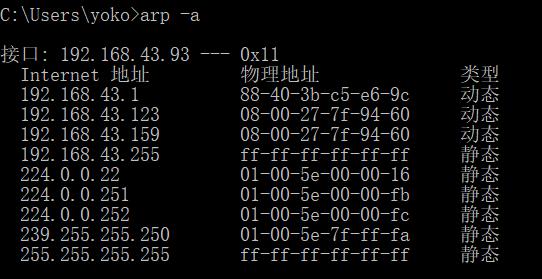
* **Task 1B (using ARP reply)：**

使用ARP应答污染。

代码根据Task 1A的代码修改，如下：



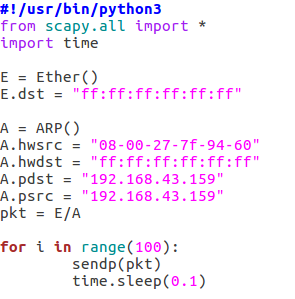
和1A的步骤一样，执行程序后同样会看到B的MAC地址被修改，表明污染成功：



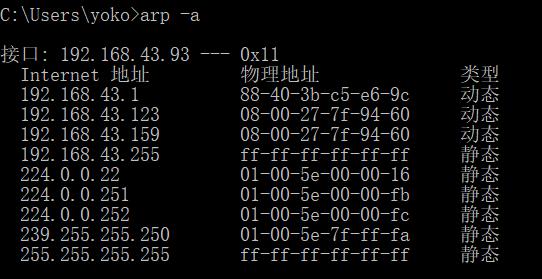
同样进行ping刷新。

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

免费ARP攻击。通过不断地广播ARP报文达到污染的效果。代码在原有基础上进行修改：



将源、宿IP地址均设置为污染目标IP地址，宿MAC地址为全f。同样执行上述程序后，用arp -a命令查看，发现污染成功。



**（三）IP/ICMP Attacks Lab**

# Task 1: IP Fragment

* **Task 1.a: Conducting IP Fragmentation**

参照手册，将代码补全，保存为frag.py，然后打开Wireshark进行监测：



在攻击端运行frag.py，通过Wireshark监测发包结果：



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

修改frag.py文件，将第二片报文的前8个字节与第一片报文的后8个字节重合，然后将第二片报文的载荷中的A全部改为B：



再用相同的方式执行并查看Wireshark：

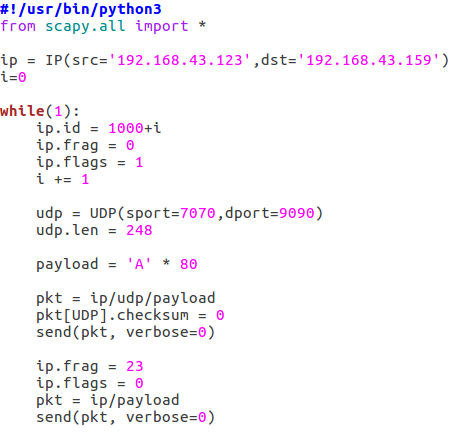


* **Task 1.c: Sending a Super-Large Packet**

将IP头中的字段设置为，然后不断发送为1的报文，也就是一直继续分片。当分片总长超过后，设置其为0。此时UDP服务器崩溃了。

* **Task 1.d: Sending Incomplete IP Packet**

在此任务中，依旧改写frag.py，改成只发送部分分片，并且不断改变id的程序：



可以发现，作为UDP服务器的虚拟机（ip为192.168.43.159）内存占用会不断升高。