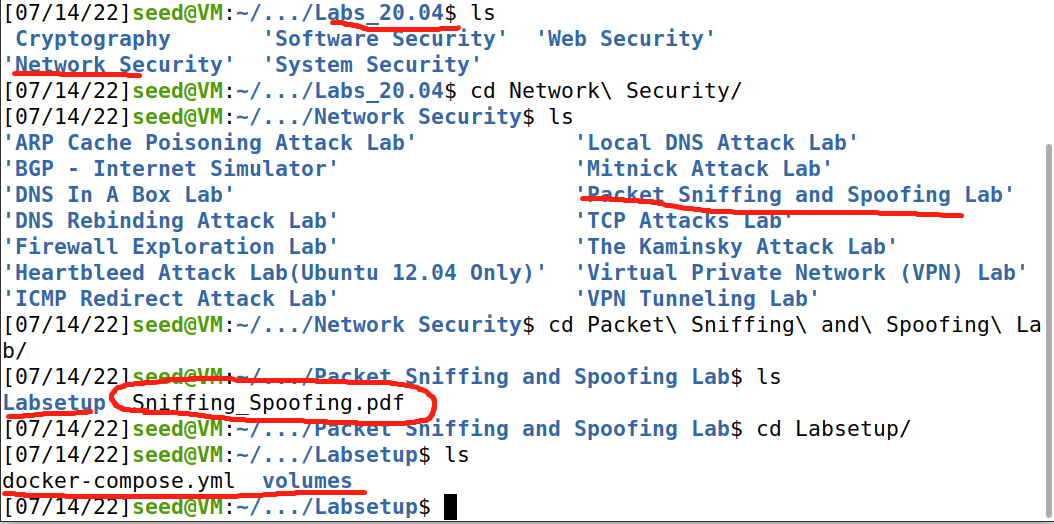
**Lab-1报文嗅探和伪造**

1. 环境配置
2. **进入实验目录: 用户名seed，密码dees**

/home/seed/Desktop/**Labs\_20.04**/**Network Security**/**Packet Sniffing and Spoofing Lab**/**Labsetup**

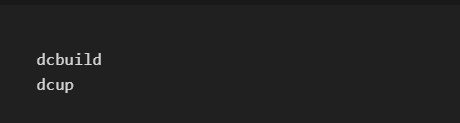


可以看到本实验的task 任务书**sniffing\_spoofing.pdf**，以及Labsetup下的**docker-compose.yml**文件和用于映射到容器内的文件夹

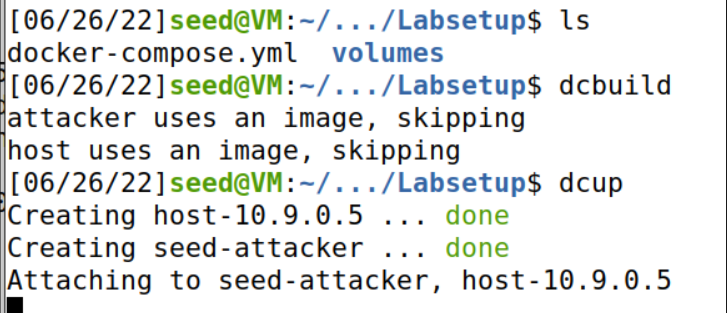
(**docker-compose.yml和文件夹用于创建docker**)

1. **创建容器镜像和启动容器**

在一个窗口中运行以下命令:



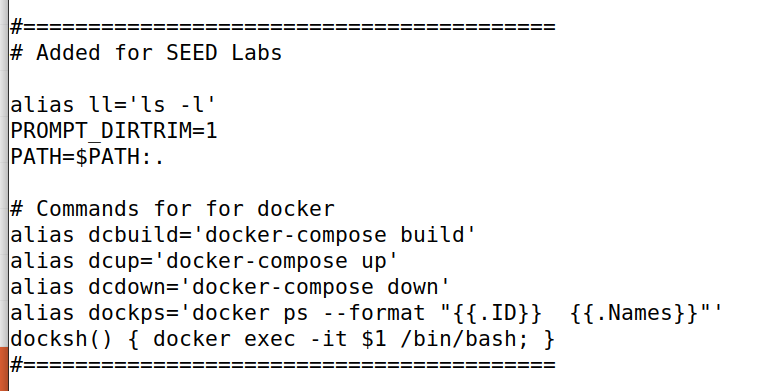
之后结果如下所示：



**注意**

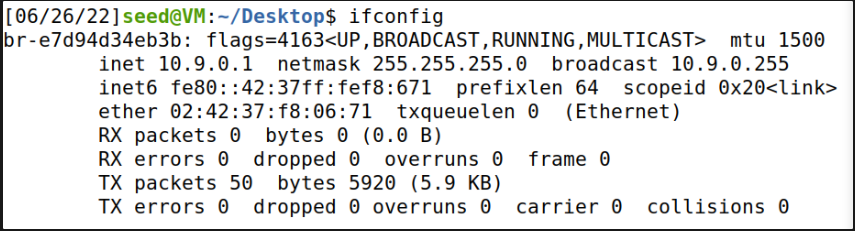
Attaching进入容器可能会卡住，我们需要另外开两个终端窗口用docker exec进入容器。

另外，dcbuild，dcup，dockersh等命令是为了方便给docker命令起了别名，具体的定义在**~/.bashrc**文件中（**~/表示用户的home目录，.表示隐藏文件**）。



1. **查看网桥信息**

在容器创建后，**docker会为容器创建网桥用于容器间通信**，我们还要在宿主机中查看该网桥的名称，使用ifconfig命令即可查看:

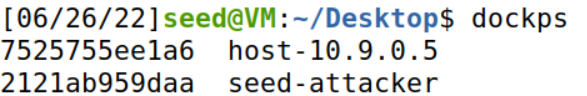


可以发现，这里创建的网桥名称是**br-e7d94d34eb3b**(各人实验名称不同),后面会在嗅探时用到该信息。

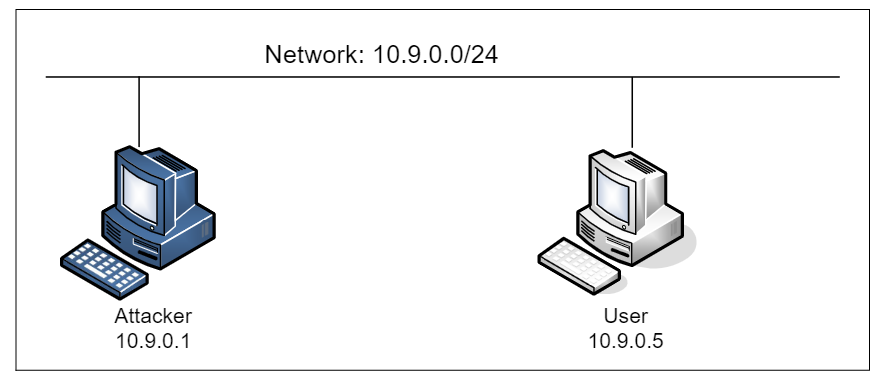
1. 进入两个容器进行操作

此时两个容器已经启动完毕，使用**docksh**命令可以连接到容器进行接下来的操作。

但是首先要看一下两个容器的ID是多少: 使用`**dockps**`命令查看



可以看到被攻击主机容器ID以75开头，攻击容器ID以21开头（自己的机器上不一定是这两个id），我们连接到容器的时候使用ID前两位即可标识一个容器。



1. 分别在另外两个窗口连接到被攻击主机和攻击主机。

命令如下（实际上是docker exec。。。）:



结果如下所示:





1. Task1 - Using Scapy to Sniff and Spoof Packets

**1.1 Sniffing packets**

**1A Simple Sniffing**

1. 进入攻击主机，在/volumes文件夹下创建sniffer.py文件，并编写以下代码:
2. #!/usr/bin/env python3
3. from scapy.all import \*
5. def print\_pkt(pkt):
6. pkt.show()
8. pkt = sniff(iface='br-e7d94d34eb3b', filter='icmp', prn=print\_pkt)

因为volumes文件夹已经映射，所以可以直接在虚拟机上写该文件，写代码会更方便一些。

**注意：**

这里的网卡名称br-e7d94d34eb3b就是上边ifconfig获取的名称。

一般容器都是交换机互联，是看不到其他容器的报文的。这里搭建攻击者容器指定了host模式，可以看到。

**解释：**

Python的Scapy 是一个屏幕、报文、web内容抓取工具。

其中的sniff函数及其参数如下：

sniff(count=0,store=1,offline=None,prn=None,filter=None,L2socket=None,timeout=None,opened\_socket=None,stop\_filter=None,iface=None)

各参数的含义为：

Count：指定最多嗅探多少个符合要求的报⽂，设置为0时则⼀直捕获

store:指定保存抓取的数据包或者丢弃，1为保存，0为丢弃

offline:从pcap⽂件中读取数据包，⽽不进⾏嗅探，默认为None

prn:为每个数据包定义一个回调函数，回调函数会在捕获到符合filter 的报⽂时被调⽤。

filter:⽤来筛选抓取的信息，其⽤法与常见抓包软件WireShark 等相同

L2socket:使⽤给定的L2socket

timeout:在给定的事件后停⽌嗅探，默认为None

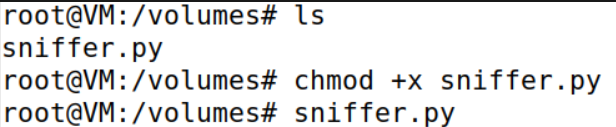
opened\_socket:对指定的对象使⽤.recv进⾏读取

stop\_filter:定义⼀个函数，决定在抓到指定的数据之后停⽌

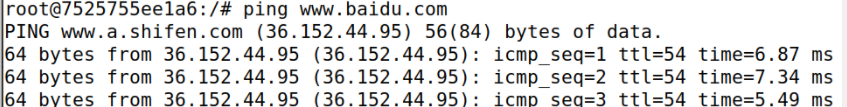
iface:指定抓包的⽹卡,不指定则代表所有⽹卡。

1. **攻击主机开始监听网卡**

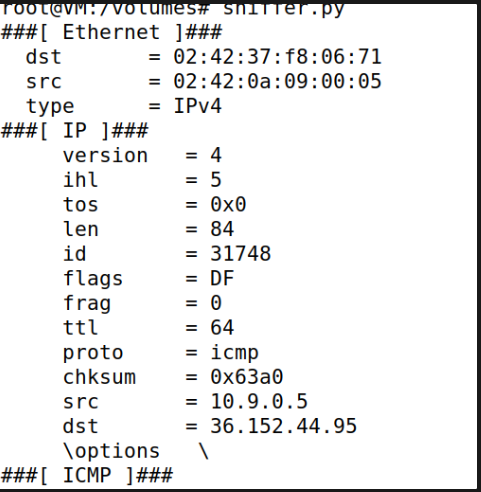
先给文件赋予可执行权限，然后开始监听。



1. **被攻击主机容器中发送ICMP包 (ping 即可)**



1. **攻击主机容器可以看到ICMP包**



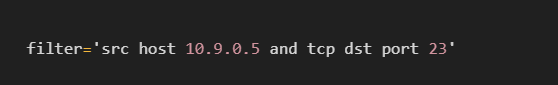
**1B: 切换规则实现监听不同的报文**

(1). 仅监听ICMP报文

和1A过程相同。

(2). 监听来自特定IP地址的TCP协议，目的端口为23的报文

主机的IP地址位10.9.0.5，我们监听从本机发出的报文，attacker上sniffer.py的filter修改如下:



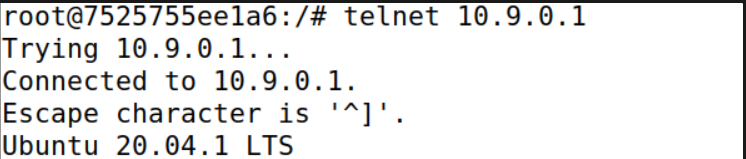
重复上述监听过程:

1. attacker主机启动sniffer.py

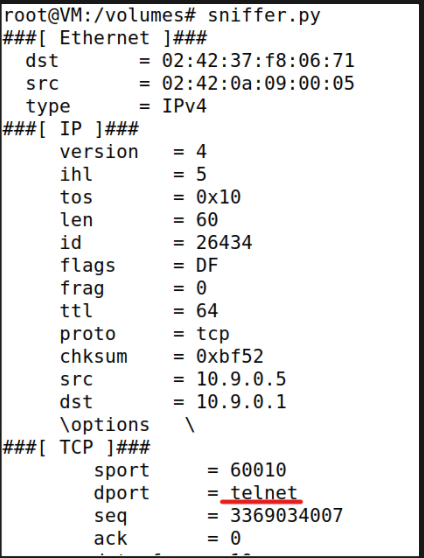


1. 被攻击主机发出数据包

因为**23端口是telnet端口**，所以使用telnet命令发出数据包，这里直接向10.9.0.1网关发送请求。



1. 在attacker主机可看到数据包



**(3). 监听某子网的报文**

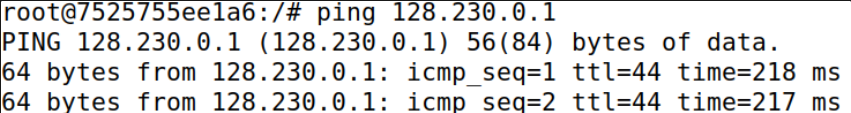
修改sniffer.py为



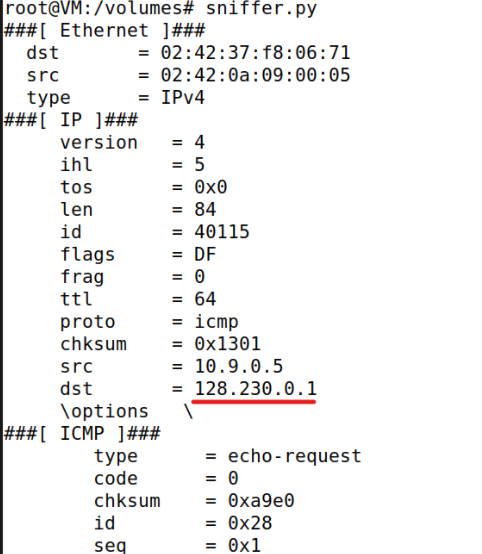
1. attacker主机启动sniffer.py



1. 在被监听主机容器中发送到128.230.0.1的数据包



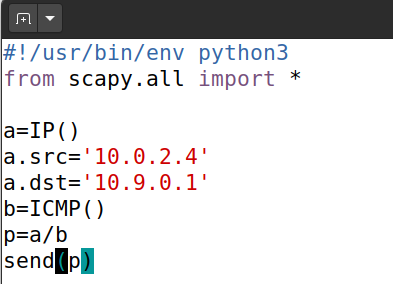
1. attacker主机监听结果如下:



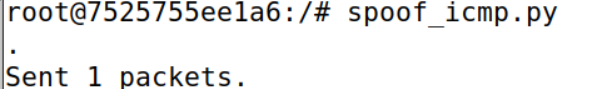
**1.2 Spoofing ICMP Packets**

**IP地址欺骗**

1. 修改attacker上的sniffer.py，设置filter=’icmp’
2. 在attacker主机上运行sniffer.py
3. 在被监听主机上创建spoof\_icmp.py文件，内容如下:



chmod +x spoof\_icmp.py 之后运行结果如下，发送成功。



**注释**：

1、scapy库的IP()、ICMP()和send()函数，分别提供构造报文和发送功能。

2、scapy库中重载了/除法运算符，起到报文连接运算符的作用。

3、在没有挂载volume的docker里面编写代码不太方便，可以安装一个vim或者通过拷贝文件传输。

**安装vim的命令是**

apt-get update

apt-get install vim

vim /home/seed/spoof.py

进入新建的文件

按 i 键切换为输入模式，录入如下代码

from scapy.all import \*

a=IP()

a.src='10.0.2.4'

a.dst='10.9.0.1'

b=ICMP()

p=a/b

send(p)

完成录入后，先按Esc键回到命令模式，再输入 :wq 保存并退出vim

可以用cat /home/seed/spoof.py检查一下文件

然后用chmod +x /home/seed/spoof.py 修改可执行

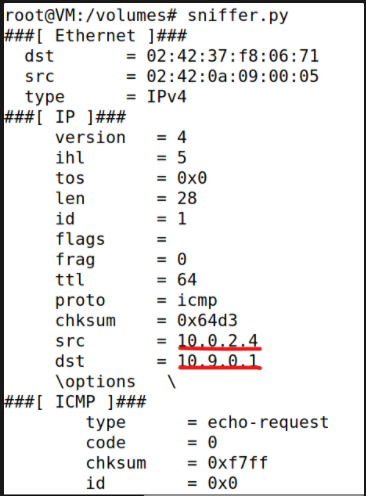
**# 从宿主机拷贝文件到容器内**

# docker cp 文件路径 容器id:容器内路径

例如，docker cp test.txt eaac94ef6926:/home/test.txt

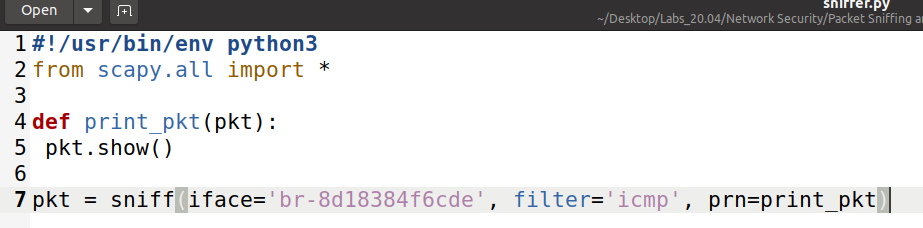
# 反过来可以从容器cp到主机

1. attacker主机中收到的包如下所示:



**注意：**

**可能需要把sniffer的filter改回icmp。**

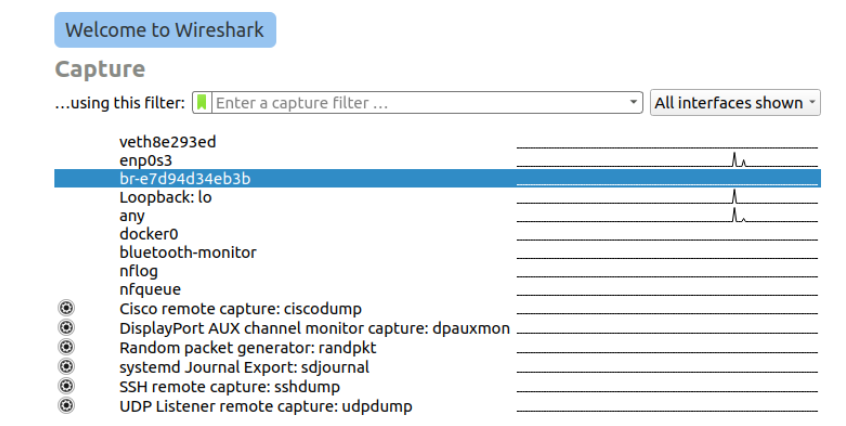


**可能需要安装scapy：apt-get install scapy**

**1.3 Traceroute**

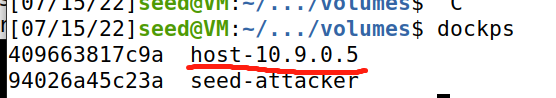
**注意：这个实验需要编写python文件不断尝试TTL的大小，直到有回复为止。**

1. 在被监听主机上创建try\_ttl.py文件，内容如下：
2. #!/usr/bin/env python3
3. from scapy.all import \*
5. **for** i in range(1,30):  # 一般30跳足矣
6. a=IP()
7. a.dst='202.108.2.5'
8. a.ttl=i
9. b=ICMP()
10. p=a/b
11. send(p)
12. 在**虚拟机上开启wireshark监听网桥**，这里是br-e7d94d34eb3b



**注意：**

网桥是用来给容器通信的，如果报文的发出是容器是host-10.9.0.5，可能是不经过网桥，wireshark在网桥上监听不到报文。



因此，需要在seed-attacker容器上运行try\_ttl.py文件才能被监听到。

可以先验证一下，在seed-attacker容器上运行ping试一试

（seed-attacker容器需要apt-get install ping来安装ping工具）。

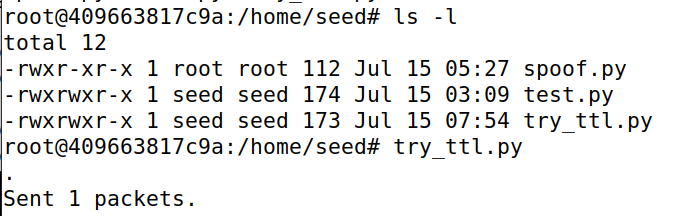


1. 在被监听主机上运行try\_ttl.py文件(py文件运行之前别忘记加运行权限)

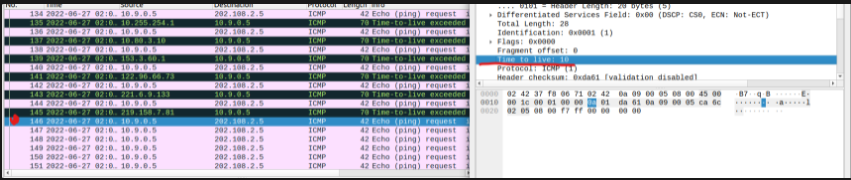
**先拷贝py文件过去：**



运行：

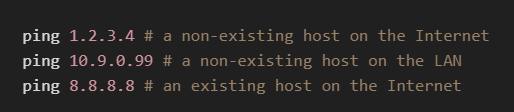


1. 在wireshark中可以看到结果，找到第一个没有回复Time-to-live Exceeded的包，对应下面画红点的包，然后发现TTL为10，结果可能会有所不同，这个没有关系。



**1.4 Sniffing and-then Spoofing**

有三个IP地址，如下：



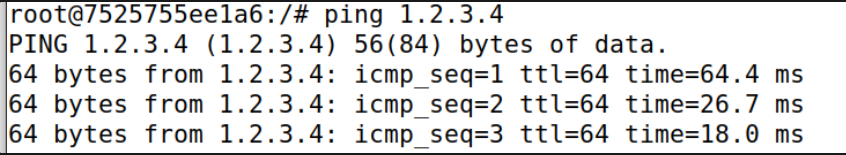
首先，在被监听主机中依次访问三个IP地址，发现只有第三个地址可以Ping通。

1. **在attacker主机（）中创建sniff\_and\_spoof.py，内容如下**

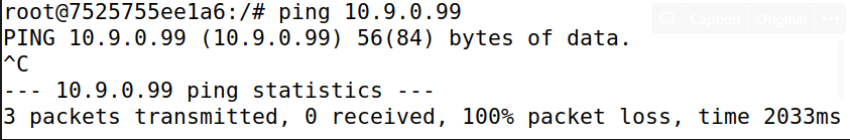


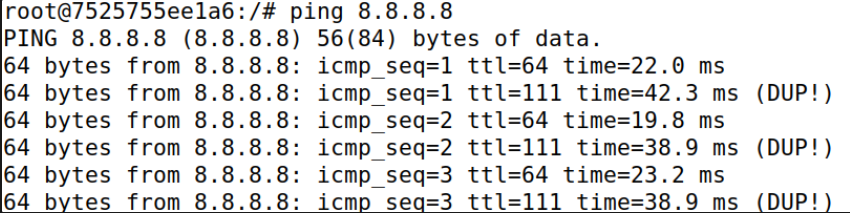
之后chmod +x sniff\_and\_spoof.py 添加运行权限并运行

1. #!/usr/bin/env python3
2. from scapy.all import \*
4. def spoof\_pkt(pkt):
5. **if** ICMP in pkt and pkt[ICMP].type == 8:
6. print("Original Packet.......")
7. print("Source IP : ", pkt[IP].src)
8. print("Destination IP:", pkt[IP].dst)
10. ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
11. icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
12. data = pkt[Raw].load
13. newpkt = ip/icmp/data
15. print("Spoofed Packet.......")
16. print("Source IP : ", newpkt[IP].src)
17. print("Destination IP : ", newpkt[IP].dst)
18. send(newpkt, verbose=0)
20. pkt = sniff(iface='br-e7d94d34eb3b', filter='icmp', prn=spoof\_pkt)
21. **在被监听主机ping三个IP地址，结果如下:**



不存在的地址1.2.3.4之所以得到回应是因为ICMP包要经过网关，然后被嗅探到，之后attacker的sniff\_and\_spoof程序返回响应。



10.9.0.99因为是子网内部的包，不会发到子网外，不需要经过网关，但是这个地址本身不存在，所以不会有响应。  


8.8.8.8是存在的公网IP地址，不仅会得到正常响应，还会在被监听到后spoof发送响应，所以会有DUP的标志。