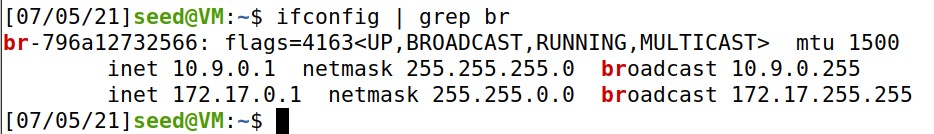
# TASK 1.1

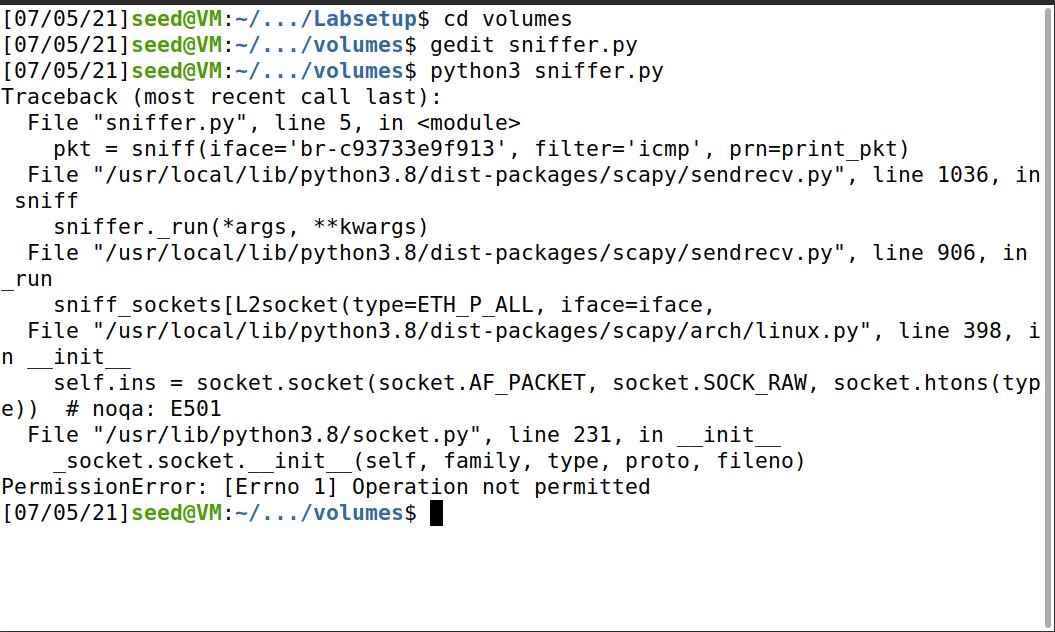
**1.1A** 用ifconfig查询自己的iface，为br-796a12732566



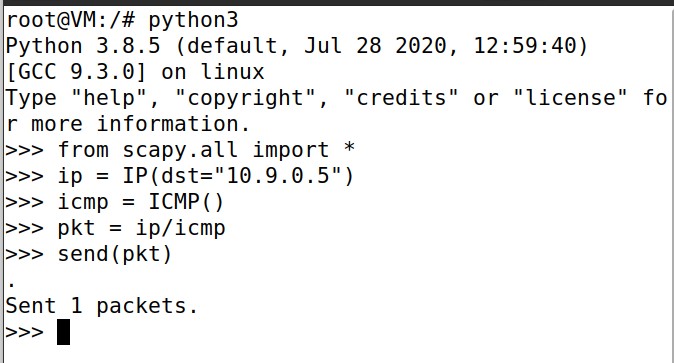
在volumes目录下新建sniffer.py，写入下图：



首先在用户态下运行发现运行失败，因为没有相应权限。如图。



进入root后再次运行sniffer.py，接着在docker上构造并发送如下报文：



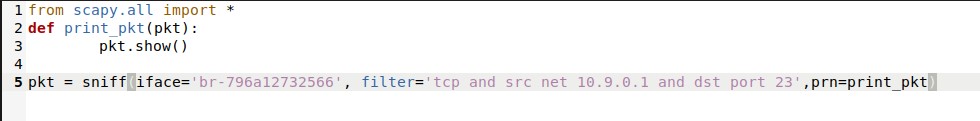
sniffer.py成功捕获如下信息：



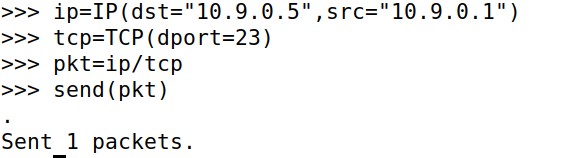
## 1.1B

捕获ICMP报文，sniffer.py代码和捕获结果同1.1A。

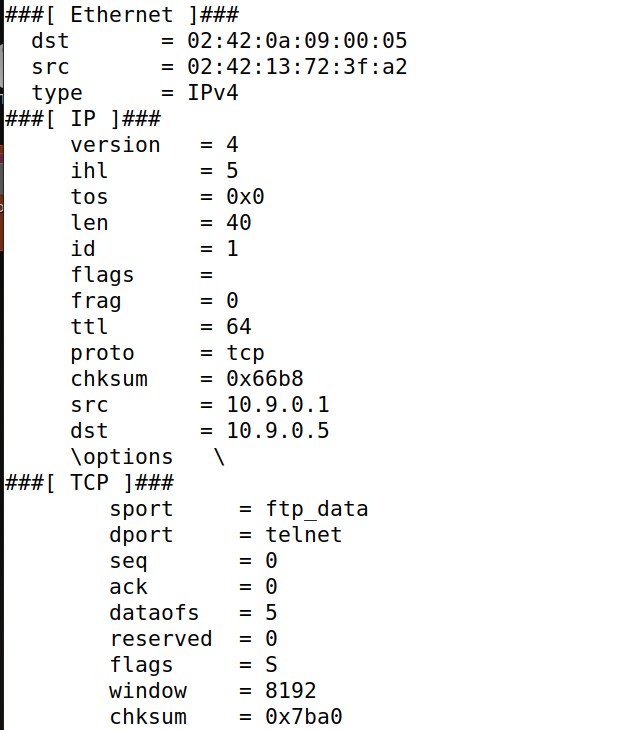
捕获特定源地址和目的端口号为23的TCP报文时，更改sniffer的filter，如下：



运行sniffer.py，在docker中重新构造并发送报文，如下：

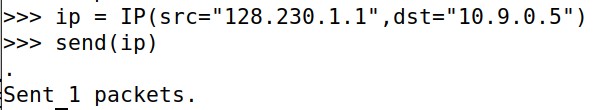


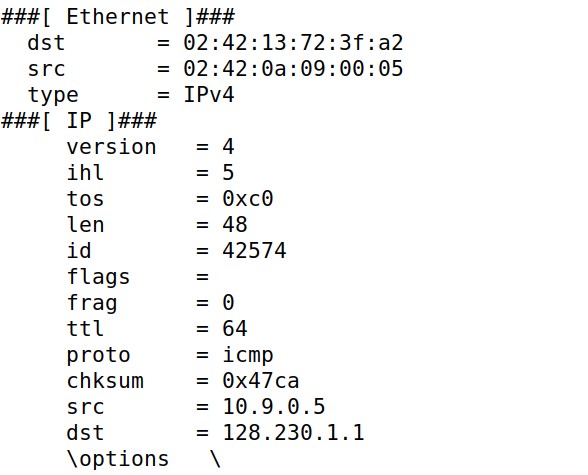
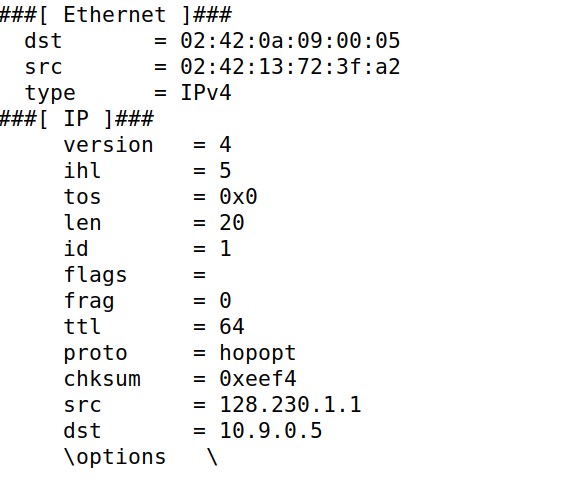
sniffer.py捕获到的结果如下，其中dport端口为telnet，默认为23。



捕获来自任意子网或去往任意子网的报文，重复上述步骤，其中sniffer.py改为：

 构造的报文为：

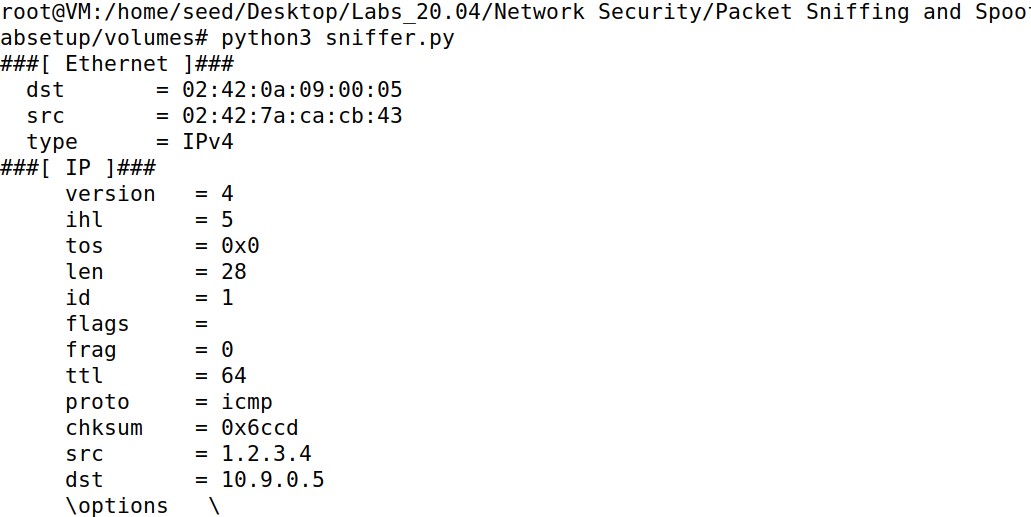
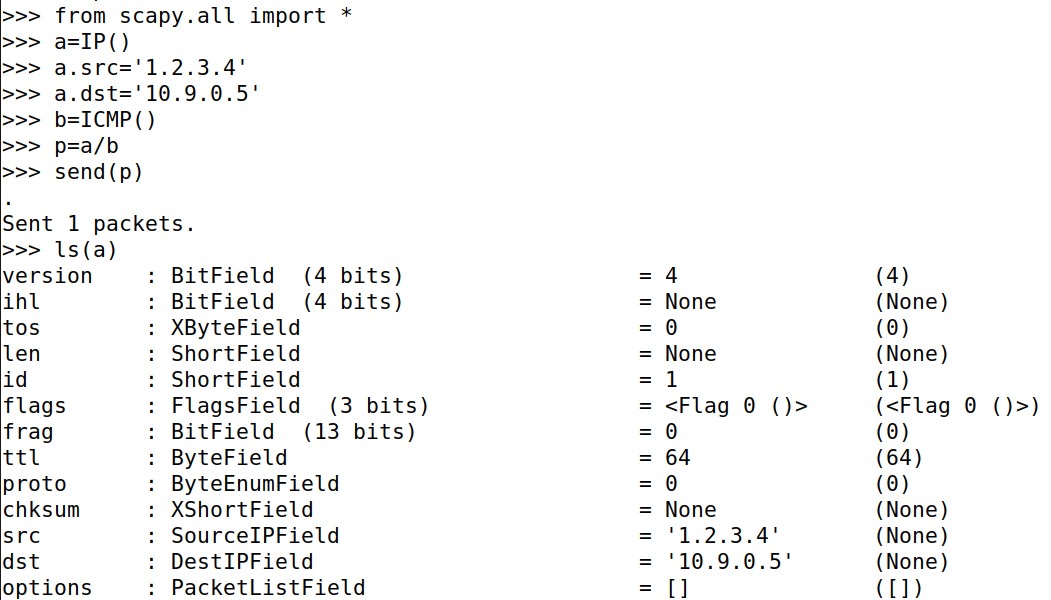
 捕获的结果为：



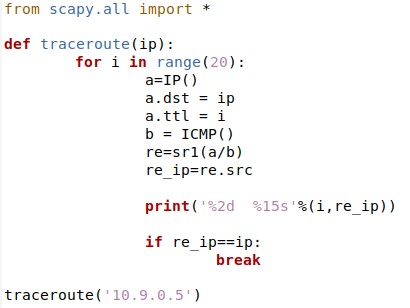
可以看到无论是 src 为 128.230.1.1 还是 dst 为 128.230.1.1，都能成功捕获到。

# TASK1.2

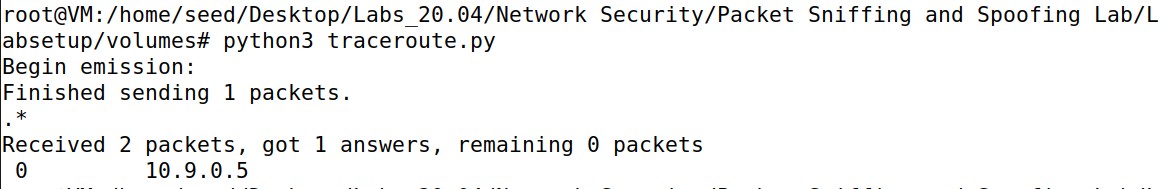
实验过程和结果如下



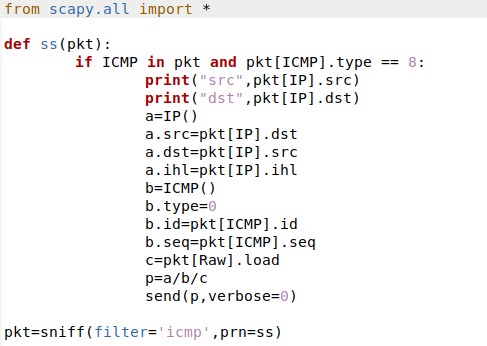
**TASK1.3** traceroute.py代码如下



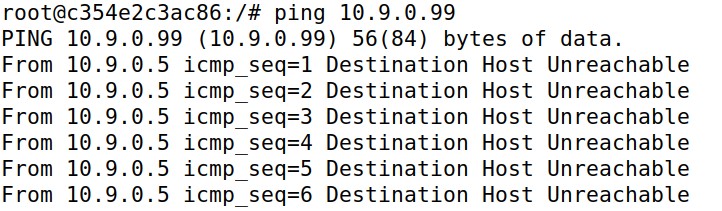
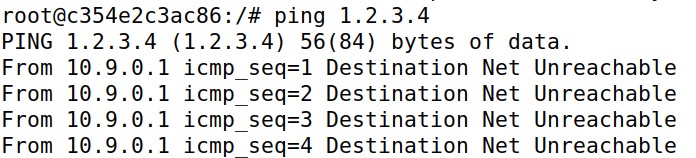
可以看到经过一跳到达了目的地址。

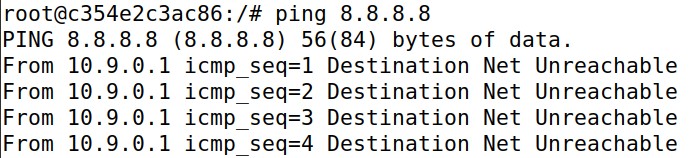


# TASK1.4 ss.py的代码如下

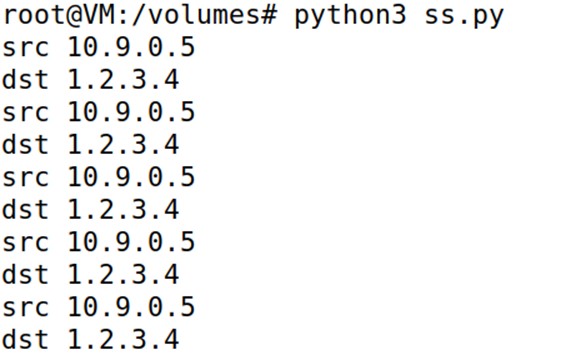
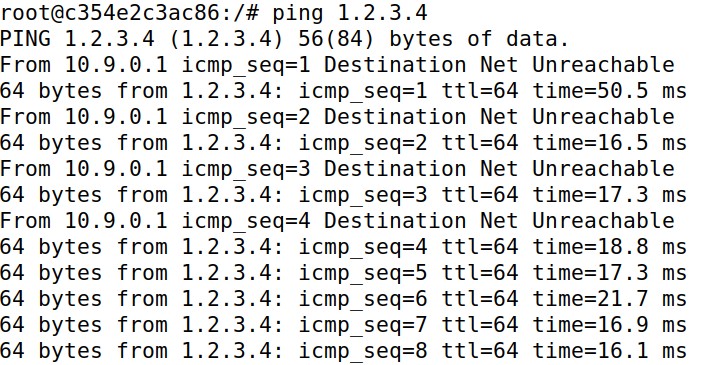


在未运行ss.py时，我们ping一下三个地址分别得到如下情况，可见三个地址都是不可到达的，但是外网的两个地址不可达是10.9.0.1告诉我们的，而内网地址不可到达是10.9.0.5告诉我们的，也就是说10.9.0.1是出内网的网关。

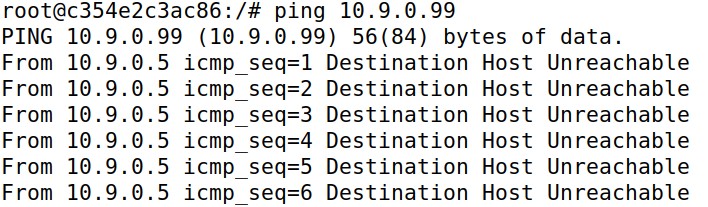




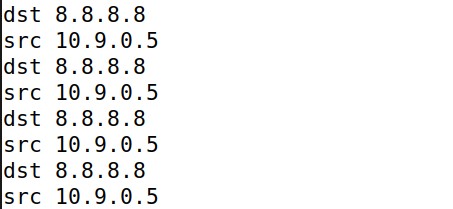
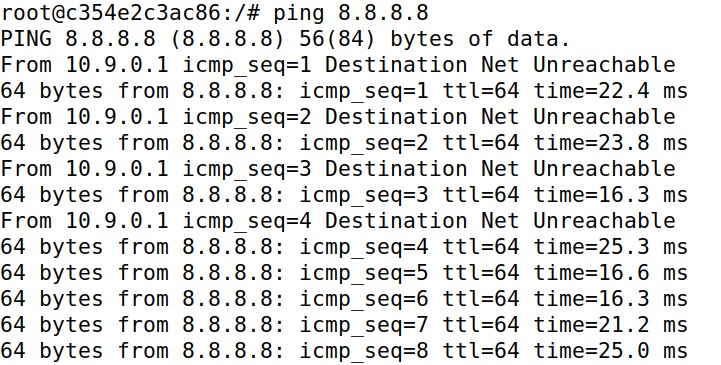
运行ss.py后再次ping这三个地址，下图是ping 1.2.3.4时的结果和ss.py的输出，可以看到此时1.2.3.4可达。



下面是ping 10.9.0.99时的结果，此时ss.py并没有任何输出，不可达。



下面是ping 8.8.8.8时的结果和ss.py的输出，地址可达。



根据上面的情况可知attacker机也就是上面说的10.9.0.1是出内网的网关，而本机是10.9.0.5，在未运行ss.py时，都不能ping通，而运行之后，外网的两个地址能 ping 通是因为，他们两个的报文要经过 attacker 机出去，所以被 attacker检测到，并伪造了返回报文，让本机误以为可以ping通，但是ping内网的地址时，不需要经过 attacker，所以 attacker 没有返回伪造报文，而 10.9.0.99这个内网地址又是不存在的，结果就和没运行ss.py时一样ping不通。