Packet Sniffing and Spoofing 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. d88950f5b1f9 host-10.9.0.5：用于模拟或代理一个特定的网络环境或服务器，其 IP 地址为 10.9.0.5。用于测试、开发或生产环境。 2. c7f1fc492fb9 seed-attacker：被用于发起对 host-10.9.0.5 或其他系统的模拟攻击，以测试目标系统的安全性。在渗透测试场景中，这个容器可以作为一个“攻击者”的基地，从中发起各种安全评估活动。 |

|  |
| --- |
| 任务：在在SEED环境测使用ifconfig/docker network命令查看容器的网卡和网络信息 |
| 命令：  Ifconfig |
| 截图： |
| 结果说明：  1. br-4eedb1db6a4d  类型: 这是一个桥接网络接口（br 开头），通常是由 Docker 创建的，用于容器之间的通信。  IP地址: inet 10.9.0.1 表示该接口的 IPv4 地址是 10.9.0.1，子网掩码是 255.255.255.0，广播地址是 10.9.0.255。  状态: UP,BROADCAST,RUNNING,MULTICAST 表明这个接口已启动，并且正在运行，支持广播和多播。  数据包: 该接口目前已经接收（RX）和发送（TX）了一些数据包，但没有发生任何错误、丢包或冲突。  2. docker0  类型: 这是 Docker 的默认桥接网络接口。通常用于连接未指定网络模式的 Docker 容器。  IP地址: inet 172.17.0.1 表示该接口的 IPv4 地址是 172.17.0.1，子网掩码是 255.255.0.0，广播地址是 172.17.255.255。  状态: UP,BROADCAST,MULTICAST 表明该接口已启动，但在 state DOWN 表明目前没有活动连接。  数据包: 当前接口上没有任何接收或发送的数据包，这表明可能没有容器连接到这个网络，或者没有流量通过。  3. ens33  类型: 这是一个物理以太网接口，是主机连接到物理网络的主要接口。  IPv6地址: inet6 fe80::63c4:fde4:65a3:3a2e 表示该接口的 IPv6 地址是链路本地地址。  状态: UP,BROADCAST,RUNNING,MULTICAST 表明接口正在正常运行。  数据包: 接口已经发送和接收了一些数据包，且没有出现错误或丢包。  4. lo (Loopback Interface)  类型: lo 是回环接口，用于主机本地通信，允许主机与自己通信。  IPv4地址: inet 127.0.0.1 是回环地址，子网掩码是 255.0.0.0。  IPv6地址: inet6 ::1 是回环地址的 IPv6 版本。  状态: UP,LOOPBACK,RUNNING 表示接口正在正常工作。  数据包: 接口已经接收和发送了数据包，主要用于主机内部的网络操作，没有发生错误。  5. veth9b006f5  类型: 这是一个虚拟以太网接口（veth），通常是 Docker 创建的，作为容器与主机之间的连接。  IPv6地址: inet6 fe80::808e:50ff:feb9:713c 表示该接口的 IPv6 地址是链路本地地址。  状态: UP,BROADCAST,RUNNING,MULTICAST 表明接口正在运行。  数据包: 接口已经发送了一些数据包，但还没有接收任何数据包。这可能是因为容器刚启动，或者还没有通过该接口进行通信。 |

## 实验步骤

### 2.1 Using Scapy to Sniff and Spoof Packets

|  |
| --- |
| 任务：按照任务书要求使用程序和交互式shell两种方式生成一个IP报文  位置：SEED |
| 命令：   1. 创建程序my\_code.py 2. Chmod a+x my\_code.py   运行:my\_code.py   1. 用交互式shell的方法生成IP报文   Python3  From scapy.all import \*a =IP()  a.show() |
| 截图 |
|  |

### 2.2 Task 1.1: Sniffing Packets

|  |
| --- |
| 任务：按要求使用不同权限执行任务书上的sniff程序，观察执行结果并进行说明  位置：SEED |
| 命令：   1. 在labsetup上中的volumes文件夹内创建sniff.py文件（touch sniff.py），内容如下：   #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()    pkt = sniff(iface='br-4eedb1db6a4d ', filter='icmp', prn=print\_pkt)   1. 进入攻击主机赋予权限，然后监听：   Ls  Chmod +x sniff.py  Ping www.baidu.com |
| 截图      (宿主机没有权限) |
| 结果说明：  1.网络嗅探需要 root 权限，因为它涉及到低层次的网络操作（如数据包捕获）。普通用户权限无法捕获数据包。  2. 执行 ping www.baidu.com ：  在 root 权限下：  数据包捕获：ping www.baidu.com 发送的 ICMP 数据包被 sniff.py 捕获，并在终端上显示详细的包信息。 |

|  |
| --- |
| 任务：按要求为sniff编写不同的filter对报文进行过滤，并观察执行效果  位置：HOST U |
| 命令：   1. 修改sniff为：   #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()    pkt = sniff(iface='br-4eedb1db6a4d', filter='src host 10.9.0.5 and tcp dst port 23', prn=print\_pkt)   1. 由于 23 端口为 telnet，故我们在 host 上 telnet attacker 一下：telnet 10.9.0.1，观察attacke：sniff.py 2. 修改sniff：   #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()    pkt = sniff(iface='br-4eedb1db6a4d ', filter='net 128.230.0.0/16', prn=print\_pkt)  4.在host上：ping 128.230.0.1 -c 1，观察attack：sniff.py |
| 截图 |
| 结果说明：  1.第一个过滤器结果（src host 10.9.0.5 and tcp dst port 23）：  当 10.9.0.5 发送符合过滤条件的 TCP 流量到端口 23 时，嗅探器捕获并显示数据包。  过滤器非常具体，只会捕获匹配该条件的流量。执行 telnet 10.9.0.1 从其他 IP 地址发起的流量不会被捕获。  2.第二个过滤器结果（net 128.230.0.0/16）：  ping 128.230.0.1 时，ICMP Echo 请求和回复数据包都在该 IP 地址范围内，因此它们会被 sniff.py 捕获。在终端中看到 ICMP 数据包的详细信息，包括源 IP、目标 IP、TTL、ICMP 类型等。 |

### 2.3 Task 1.2: Spoofing ICMP Packets

|  |
| --- |
| 任务：修改任务书代码生成并发送一个伪造的ICMP Echo Request报文，并设置该报文的源地址  位置：SEED |
| 命令   1. 编写spoofer.py:   #!/usr/bin/env python3  from scapy.all import \*  a = IP()  a.src = '10.10.0.1'  a.dst = '10.10.0.2'  b = ICMP()  p = a/b  send(p)   1. 复制进host: docker cp spoofer.py d88950f5b1f9:/spoofer.py 2. Sniff.py改为：   #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()    pkt = sniff(iface='br-4eedb1db6a4d', filter='icmp', prn=print\_pkt)   1. 在被攻击主机上提升spoofer的权限，并运行：   Chmod +x spoofer.py  Spoofer.py  5.attacter主机运行sniff.py,观察结果。 |
| 截图: |
| 结果说明：  1. 捕获伪造的 ICMP 数据包：sniff.py 会捕获网络中所有 ICMP 数据包。 spoofer.py 发送了一个伪造的 ICMP Echo Request，sniff.py 捕获到这个数据包并在终端上显示详细信息。（sent packet 1）  2.sniff.py 显示该 ICMP 数据包的完整内容，包括 IP 层的源地址和目的地址，以及 ICMP 层的类型（Echo Request）。 |

### 2.4 Task 1.3: Traceroute

|  |
| --- |
| 任务：按任务书要求编写一个使用ICMP协议的traceroute程序，  位置：SEED |
| 命令   1. 编写tracerouter.py：   #!/usr/bin/env python3  from scapy.all import \*  import sys  import socket  # 获取目标 IP  argumentList = sys.argv  MAX\_TTL = 255  dstHostname = argumentList[1]  dstIP = socket.gethostbyname(dstHostname)  continuousLost = 0  MAX\_LOST = 6  print("dst IP: " + dstIP)  # 创建 IP 包和 ICMP 包  a = IP()  a.dst = dstIP  a.ttl = 1  b = ICMP()  while a.ttl <= MAX\_TTL:  # 发送数据包  reply = sr1(a/b, verbose=0, timeout=3)  print(str(a.ttl), end="\t")    if reply is None:  print("===PACKET LOST===")  continuousLost += 1  if continuousLost >= MAX\_LOST:  print("Unable to reach " + dstIP + "! Stop.")  break  else:  print(reply.src)  continuousLost = 0  if reply.src == dstIP:  break  # 增加 TTL 值  a.ttl += 1   1. ttl += 1 2. 在host上运行tracerouter.py 1.2.3.4,观察结果。 3. 在host上运行tracerouter.py 8.8.8.8,观察结果。 |
| 截图 |
| 结果说明：   1. 在 1.2.3.4 这种不可达的地址上，只看到几次 TTL 递增的输出，接着是连续的 ===PACKET LOST=== 提示，最终程序提示无法到达该 IP 地址并终止。 2. 看到一系列的 TTL 值和相应的 IP 地址输出，每个 IP 地址代表一个经过的路由器。   最终看到 8.8.8.8 的 IP 地址，表示已经到达目标主机，Traceroute 成功完成。  网络正常且无丢包，可以看到所有跳数的 IP 地址，包含 15个左右的中间路由器。 |

### 2.5 Task 1.4: Sniffing and-then Spoofing

|  |
| --- |
| 任务：编写一个监听-伪造响应的程序，并从容器中对三个不同的地址进行ping操作，观察实验结果并进行说明  位置：USER-container，SEED |
| 命令   1. 编写sniff\_spoofer.py程序：   #!/usr/bin/env python3  from scapy.all import \*  def spoof\_pkt(pkt):  if ICMP in pkt and pkt[ICMP].type == 8:    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)  icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)  data = pkt[Raw].load  newpkt = ip/icmp/data    send(newpkt, verbose=0)  pkt = sniff(iface='br-4eedb1db6a4d ', filter='icmp', prn=spoof\_pkt)   1. 将该程序在attacker上运行。 2. 在host上   ping 1.2.3.4 -c 1  ping 10.9.0.99 -c 1  ping 8.8.8.8 -c 2   1. 查看：   ip route get 1.2.3.4  ip route get 10.9.0.99  ip route get 8.8.8.8 |
| 截图 |
| 结果说明：   1. ping 1.2.3.4：这个地址是不可达的。在无伪造响应情况下，ping 命令超时，提示目标主机不可达。在伪造响应情况下，由于 sniff\_spoofer.py 伪造了一个回应，ping 显示这个地址响应了请求。 2. ping 10.9.0.99：这是一个私有 IP 地址，在伪造响应情况下，ping 显示响应，因为伪造程序发送了虚假的 Echo Reply。 3. ping 8.8.8.8：这是 Google 的公共 DNS 服务器，是可达的。在无伪造响应情况下，可以收到正常的 ping 响应。在伪造响应情况下，ping 命令仍会显示收到的 ICMP 回应。且不止一个，其中一个是伪造的，其他的是自身的。   4．ip route get 1.2.3.4:  路由信息：显示 1.2.3.4 通过 10.9.0.1 路由到达，使用的网络接口是 eth0。  源 IP 地址：使用的源地址是 10.9.0.5。  1.2.3.4 是一个外部网络中的地址，网络栈决定通过默认网关 10.9.0.1 来转发这些数据包。因此，所有发往 1.2.3.4 的流量都会经过 10.9.0.1 这个网关。  5. ip route get 10.9.0.99：  路由信息：显示 10.9.0.99 直接通过接口 eth0 路由，源 IP 地址是 10.9.0.5。  10.9.0.99 与 10.9.0.5 在同一个子网内，因此数据包不需要经过网关，直接通过接口 eth0 进行传输。  6．ip route get 8.8.8.8:  路由信息：显示 8.8.8.8 通过 10.9.0.1 路由到达，使用的网络接口是 eth0，源 IP 地址是 10.9.0.5。  8.8.8.8 是 Google 的公共 DNS 服务器，属于外部网络。因此，流量也会通过默认网关 10.9.0.1 进行路由。 |

### 三、实验总结

1. 通过 sniff\_spoofer.py，我们能够伪造 ICMP Echo Reply，成功影响 ping 命令的结果。这种技术在网络攻击中可能被用来欺骗网络管理工具，使其认为目标主机在线。

2. 尽管 ping 结果被伪造，ip route get 命令依然能够准确地展示目标地址的路由信息。这个命令提供了有关数据包传输路径的重要信息，有助于理解和排查网络问题。

3. 通过本次实验，我对网络流量的深入理解，尤其是在使用工具进行网络调试和安全测试时。