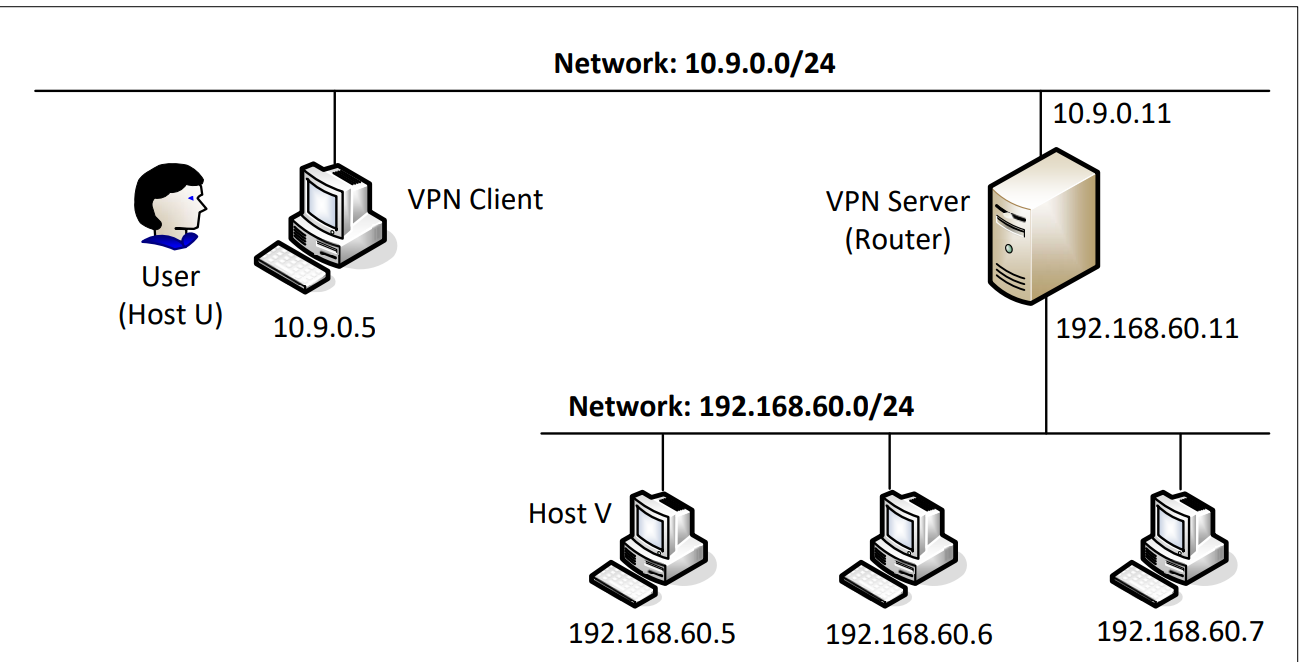
VPN Lab: The Container Version

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

实验拓扑图：



### 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. client-10.9.0.5(host U)  作用: 这个容器模拟VPN客户端。它位于一个可能不安全的网络中，通过VPN隧道连接到VPN服务器。客户端容器的IP地址为 10.9.0.5，表示它是在VPN网络中的一个节点。  2. server-router  作用: 这个容器模拟VPN服务器和网络路由器的角色。它在VPN隧道的另一端，与客户端通信并将流量路由到目标网络或互联网。  3. host-192.168.60.6(host V)  作用: 这个容器模拟一个在目标网络中的主机。它可能是VPN隧道的最终目标，也就是说，通过VPN隧道的流量最终会到达这个容器。  4. host-192.168.60.5(host V)  作用: 类似于 host-192.168.60.6，这个容器也模拟目标网络中的一个主机。它提供另一种目标来测试VPN隧道的连接性和路由配置。  一般情况下：VPN客户端 (client-10.9.0.5) 发起连接到 VPN服务器 (server-router)，并通过VPN隧道建立加密连接。VPN服务器 处理客户端的连接请求，并解密来自客户端的流量。流量通过 VPN服务器 路由到目标网络中的主机，如 host-192.168.60.6 和 host-192.168.60.5。测试客户端是否能够通过VPN隧道成功访问目标网络中的资源，从而验证VPN隧道的正确配置。 |

|  |
| --- |
| 任务：在docker中和在SEED环境测试tcpdump监听容器的网卡，比较监听网卡名称和监听报文结果 |
| 命令：   1. 在hostU上ping VPN server，测试通信   docksh 98  ping 10.9.0.11   1. 在VPN server上ping Host V（192.168.60.5），测试两者通信。   Dochsh c9  Ping 192.168.60.5   1. 在Host U上ping Host V   Docksh 98  Ping 192.168.60.5   1. 在server上运行tcpdump，监听eth0网卡   Docksh c9  Tcpdump -I eth0 -n   1. 在host U上ping 10.9.0.11 2. 在server上运行tcpdump，监听eth1网卡   Tcpdump -I eth1 -n   1. 在host V上ping 192.168.60.11   Docksh da  Ping 192.168.60.11 |
| 截图：                 (这期间client向服务器与内网设备均发送过ping，但只有服务器回应) |
| 结果说明：   1. 在hostU上ping VPN server，可以通信 2. 在VPN server上ping Host V（192.168.60.5），两者可以通信。 3. 在Host U上ping Host V，结果显示无法通信。 4. 在server上运行tcpdump，监听eth0网卡。听到了ICMP和ARP的报文，这期间client向服务器与内网设备均发送过ping，但最终只有服务器回应。（原因是因为client与服务器可以通信，与内网设备无法通信。） 5. 在host U上ping 10.9.0.11，在server上可以监听到通信。 6. 在server上运行tcpdump，监听eth1网卡，第一次ping到192.168.60.1，结果只看到了ARP报文，之后再ping192.168.60.11才可以看到ICMP报文。 7. 在host V上ping 192.168.60.11，在server上可以监听到通信。 |

## 实验步骤

### 2.1 Task 2: Create and Configure TUN Interface

### Task 2.a: Name of the Interface

|  |
| --- |
| 任务：按要求执行tun.py并观察执行结果  位置：HOST U |
| 命令：①在host U运行tun.py  Docksh 7b  Cd volumes  Chmod a+x tun.py  Tun.py  ②在主机host U上查看端口。  Ip address  ③修改程序为姓缩写“fan”  ④再次查看端口。 |
| 截图：①    ②  ③  ④ |
| 结果说明：   1. 在host U运行tun.py，可以看到tun0。 2. 在主机host U上查看端口。可以查看到tun0的端口。 3. 修改程序为姓缩写“fan”。 4. 再次查看端口。可以看到fan0的端口。 |

### Task 2.b: Set up the TUN Interface

|  |
| --- |
| 任务：按要求配置生成的TUN接口，并观察配置结果  位置：HOST U |
| 命令：   1. 在程序中加入两句代码：   os.system("ip addr add 192.168.53.99/24 dev{}".format(ifname))  os.system("ip link set dev {} up".format(ifname))   1. 重新运行程序：tun.py |
| 截图 |
| 结果说明：fan0端口被指定IP地址且state变为up而不是down，配置tun接口成功。 |

### Task 2.c: Read from the TUN Interface

|  |
| --- |
| 任务：按要求修改tun.py，并运行不同ping命令，观察tun.py运行的结果  位置：HOST U |
| 命令：   1. 修改源程序中的死循环：   while True:  #Get a packet from the tun interface  packet = os.read(tun,2048)  if packet:  ip = IP(packet)  print(ip.summary())   1. 在host U上ping 192.168.53.1，运行tun.py   Ping 192.168.53.1  Tun.py   1. 在host U上ping 192.168.60.1,查看tun.py   Ping 192.168.60.1  Tun.py |
| 截图 |
| 结果说明：  ①在host U上ping 192.168.53.1，运行tun.py，可以在tun.py运行处捕捉到icmp相关数据包，但不会产生回复。Host U通过TUN/TAP接口发送了ICMP Echo请求到目标主机。  目标主机接收到请求后，通过相同的路径（或不同的路径，取决于网络配置）发送ICMP Echo应答回Host U。  ②tun.py捕获并打印了这些数据包  ③在host U上ping 192.168.60.1,查看tun.py，可以ping通，但不能捕捉到相关数据包。网络配置使得Host U到192.168.60.0/24的流量不经过TUN/TAP接口，那么tun.py就无法捕获到这些数据包。 |

### Task 2.d: Write to the TUN Interface

|  |
| --- |
| 任务：按要求修改tun.py，并运行不同tun.py程序，观察tun.py向接口写入不同数据的结果  位置：HOST U |
| 命令：   1. 修改tun接口：   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  ip = IP(packet)  print(ip.summary())    if ICMP in ip:  newip = IP(src=ip[IP].dst, dst=ip[IP].src, ihl=ip[IP].ihl)  newip.ttl = 99  newicmp = ICMP(type=0, id=ip[ICMP].id, seq=ip[ICMP].seq)  if ip.haslayer(Raw):  data = ip[Raw].load  newpkt = newip/newicmp/data  else:  newpkt = newip/newicmp    os.write(tun, bytes(newpkt))   1. 运行tun.py并ping 192.168.53.1 2. 修改tun.py   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  ip = IP(packet)  print(ip.summary())    if ICMP in ip:  os.write(tun, bytes("Hello,world!", encoding='utf-8'))   1. 运行tun.py并ping 192.168.53.1 |
| 截图： |
| 结果说明：   1. 在第一次修改tun.py时，运行tun.py并ping 192.168.53.1，tun成功接收到报文并返回相应的ICMP报文打印出来，且能ping通。   具体过程如下：当从一个主机上 ping 192.168.53.99 时，ICMP Echo 请求报文被发送到系统的网络堆栈，并通过路由表发送到 TUN 接口。由于 TUN 接口被配置为 192.168.53.99，所有发往这个 IP 地址的数据包都会被送到 TUN 接口。脚本读取这些数据包，并判断它们是否是 ICMP Echo 请求。如果是，脚本就生成一个 ICMP Echo 回复报文，并将其写回到 TUN 接口。这个回复报文通过系统的网络堆栈返回给原始发送 ping 请求的主机，因此该主机会收到回复并显示 ping 成功。   1. 在第一次修改tun.py时，运行tun.py并ping 192.168.53.1，tun接收到报文并打印但没有返回正确的报文，无法ping通。   无法ping通的原因：在接收到 ICMP 数据包后，脚本应该生成一个 ICMP Echo 回复（type=0）的数据包，并将其写回到 TUN 接口。然而，当前的脚本并没有生成有效的 ICMP Echo 回复报文，而是直接将字符串 "Hello, world!" 写入了 TUN 接口。  这个字符串 "Hello, world!" 并不是一个有效的 ICMP 报文格式。ICMP Echo 回复必须符合 IP 和 ICMP 协议的格式规范，才能被发送方的操作系统识别并认为是一个有效的 ping 响应。 |

### 2.2 Task 3: Send the IP Packet to VPN Server Through a Tunnel

|  |
| --- |
| 任务：按要求完成tun\_server.py,并修改tun\_client.py，以及相应的配置命令，运行两个程序，并完成 ping实验  位置：HOST U / VPN Server |
| 命令：   1. 编写tun\_server.py，内容如下：   #!/usr/bin/env python3  from scapy.all import \*  IP\_A = "0.0.0.0"  PORT = 9090  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  sock.bind((IP\_A, PORT))  while True:  data, (ip, port) = sock.recvfrom(2048)  print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))  pkt = IP(data)  print(" Inside: {} --> {}".format(pkt.src, pkt.dst))   1. 客户端部分代码替换为：   # Create UDP socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  while True:  # Get a packet from the tun interface  packet = os.read(tun, 2048)  if packet:  # Send the packet via the tunnel  sock.sendto(packet, (10.9.0.11, 9090))     1. 运行两个文件,ping一下，查看结果。ping 192.168.53.2 2. 配置路由表：   ip route add 192.168.60.0/24 dev fan0  ip route   1. 尝试通信host V：ping 192.168.60.5 -c 1   查看tun\_server |
| 截图 |
| 结果说明：   1. ping 192.168.53.2时，无法ping通，但在服务器上可以接收到相关信息，理由如下：   客户机会自动将192.168.53.0/24转到我们之前设定的fan0接口，然而其并不是真实（虚拟）的接口，并不能真正发送报文。但是我们的程序却能读取并将其打包发送给我们设定的服务器。服务器将其解包，并读取到我们发送的报文实际上是从53.99发向53.2.   1. 在添加路由配置前，服务器无法接收到相关信息，路由配置后，再次ping，可以在服务器上接收到相关信息。 |

### 2.3 Task 4: Set Up the VPN Server

|  |
| --- |
| 任务：按要求完成tun\_server.py,并修改tun\_client.py，以及相应的配置命令，运行两个程序，并完成 ping实验  位置：HOST U / VPN Server |
| 命令：   1. 修改tun\_server.py：   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  IP\_A = "0.0.0.0"  PORT = 9090  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  sock.bind((IP\_A, PORT))  while True:  data, (ip, port) = sock.recvfrom(2048)  print("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))  pkt = IP(data)  print(" Inside: {} --> {}".format(pkt.src, pkt.dst))    os.write(tun, bytes(pkt))   1. 在server上运行：tun\_server.py(tun.py也要在客户端运行) 2. 再次ping一下hostV   ping 192.168.60.5 -c 1   1. 使用eth0监听：tcpdump -I eth0 -n |
| 截图 |
| 结果说明：   1. 当客户端 ping 192.168.60.5 时，ICMP Echo 请求报文通过 tun.py 的 TUN 接口被发送。这个报文最终到达服务器。 2. 服务器的 tun\_server.py 通过 UDP 套接字接收到该 ICMP 数据包。脚本解析这个数据包并试图通过 TUN 接口重新发送。tun\_server.py 在 0.0.0.0:9090 上监听的 UDP 套接字接收到来自客户端的数据包。这个数据包包含 IP 数据包（ICMP 请求）。脚本解析并打印了该数据包的源和目标地址。 3. 当在服务器上运行 tcpdump -i eth0 -n 时，看到从客户端发送到服务器的 UDP 数 据包。数据包的源 IP 地址和端口号，以及目标 IP 地址 0.0.0.0:9090 都会显示在捕获 的报文中。 |

### 2.4 Task 5: Handling Traffic in Both Directions

|  |
| --- |
| 任务：按要求修改tun\_server.py和tun\_client.py，实现客户端和服务器双向通信的功能，并使用ping和telnet命令进行实验  位置：HOST U / VPN Server |
| 命令   1. 修改tun\_server.py:   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  IP\_A = "0.0.0.0"  PORT = 9090  SERVER\_IP, SERVER\_PORT = '10.9.0.5', 9090  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  sock.bind((IP\_A, PORT))  while True:  # this will block until at least one interface is ready  ready, \_, \_ = select.select([sock, tun], [], [])    for fd in ready:  if fd is sock:  data, (SERVER\_IP, SERVER\_PORT) = sock.recvfrom(2048)  pkt = IP(data)  print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))  os.write(tun, bytes(pkt))  if fd is tun:  packet = os.read(tun, 2048)  pkt = IP(packet)  print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))  sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))   1. 修改tun.py:   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  sock = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)  SERVER\_IP, SERVER\_PORT = '10.9.0.11', 9090  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))  while True:  # this will block until at least one interface is ready  ready, \_, \_ = select.select([sock, tun], [], [])    for fd in ready:  if fd is sock:  data, (SERVER\_IP, SERVER\_PORT) = sock.recvfrom(2048)  pkt = IP(data)  print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))  os.write(tun, bytes(pkt))  if fd is tun:  packet = os.read(tun, 2048)  pkt = IP(packet)  print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))  sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))   1. 运行tun\_server.py和tun.py,通信hostV: ping 192.168.60.5 -c 1(观察结果)   ④在host U上：telnet 192.168.60.5   1. 使用wireshark抓包 hostU ping hostV |
| 截图 |
| 结果说明：   1. TUN 接口创建：创建了一个 TUN 接口，配置为 IP 地址 192.168.53.11/24。   双向通信：通过 select.select([sock, tun], [], []) 实现同时监听 UDP 套接字和 TUN 接口。当从 sock 接收到数据时（来自客户端的 UDP 数据包），将其写入到 TUN 接口。当从 TUN 接口读取到数据时，将其通过 UDP 套接字发送到客户端。该逻辑使得服务器能够处理来自客户端的数据，并将数据包路由回客户端。  ②运行 tun\_server.py 和 tun.py:服务器端和客户端通过各自的 TUN 接口进行配置并启动双向通信。双方都准备好接收和发送数据包。  ③Ping Host V (192.168.60.5):当客户端 ping 192.168.60.5 时，ICMP Echo 请求通过客户端的 TUN 接口进入，并通过 UDP 数据包发送到服务器。服务器接收到 UDP 数据包后，将其写入到服务器的 TUN 接口，路由到目标网络（192.168.60.5）。192.168.60.5 主机接收到 ICMP 请求后，生成 ICMP Echo 回复，通过相同路径返回客户端。客户端收到回复，显示 ping 成功。   1. telnet 192.168.60.5:客户端通过 TUN 接口发送 telnet 请求到服务器。服务器通UDP 数据包接收并转发请求，192.168.60.5 处理 telnet 请求并与客户端建立连接。   客户端和服务器之间通过 TUN 接口的双向通信实现 telnet 会话。（其中账号是seed，密码是dees）   1. Wireshark 抓包:当抓取 Host U 上的流量时，看到发送到服务器的 UDP 数据包，这些数据包包含从 TUN 接口捕获的 IP 数据包（例如，ICMP Echo 请求和 telnet 数据）。来自服务器的回复数据包，这些数据包通过相同的 UDP 套接字路径返回客户端。还有 ICMP Echo 请求和回复数据包，以及 telnet 协议的握手和会话数据，这些数据包封装在 UDP 数据包中并通过 TUN 接口传输。 |

### 2.5 Task 6: Tunnel-Breaking Experiment

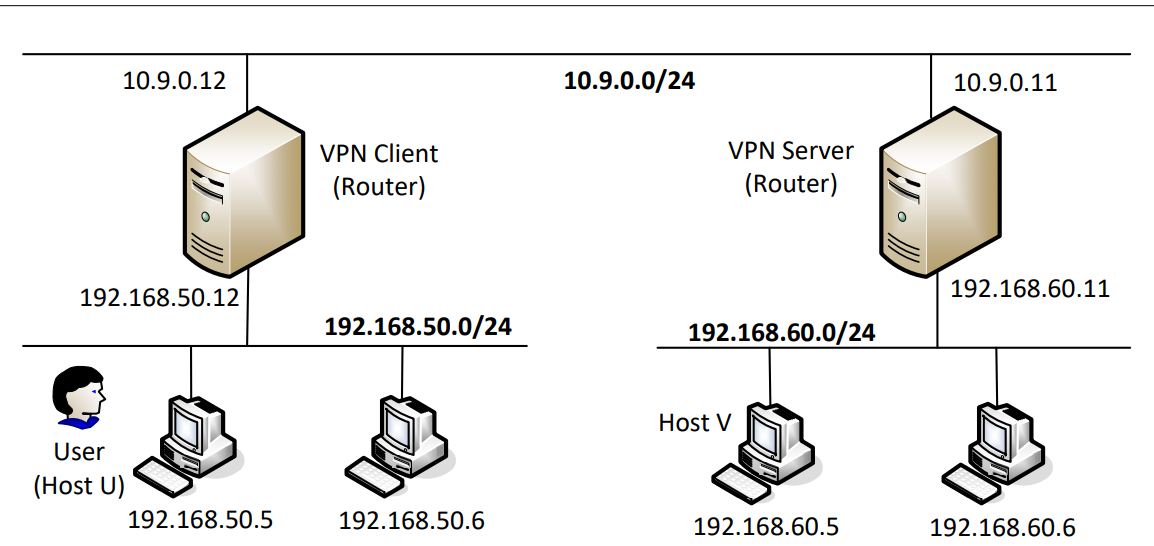
|  |
| --- |
| 任务：建立vpn通信后，关闭tun\_server或者tun\_client程序，观察ping/telnet通信的效果，然后恢复tun\_server或者tun\_client程序运行，观察ping/telnet通信的效果  位置：HOST U / VPN Server |
| 命令   1. Ctrl+c 2. telnet 192.168.60.5(一开始要tun打开才能实现远程连接，之后关闭)) 3. tun.py(U)   tun\_server.py(SERVER)   1. telnet 192.168.60.5 |
| 截图 |
| 结果说明：   1. 当 tun\_server.py 和 tun.py 都在运行时，VPN 通道是有效的。 2. 通过 telnet 192.168.60.5 连接 Host V 成功，表示客户端可以通过 VPN 服务器与目标主机进行通信。 3. 关闭 tun\_server.py：客户端与服务器之间的通信将被切断，导致 VPN 隧道失效。   现有的 ping 和 telnet 连接会中断。关闭 tun.py，客户端将无法通过 VPN 隧道发送任何数据。ping 和 telnet 会立即中断。在中断这段时间输入“test”，没有任何反应。   1. 重新启动 tun\_server.py 或 tun.py：VPN 隧道将重新建立。之前输入的“test”显示出来了。 |

### 2.6 Task 7: Routing Experiment on Host V

|  |
| --- |
| 任务：按任务书要求修改网络路由配置  位置：HOST U / VPN Server |
| 命令：   1. 在hostV上查看路由：ip route 2. 删掉default：ip route del default（再次查看路由） 3. 新增路由：ip route add 192.168.53.0/24 via 192.168.60.11 4. 在host U上ping hostV: ping 192.168.60.5 -c 1 |
| 截图 |
| 结果说明：   * 1. 初始路由状态:在 Host V 上查看路由时，存在一条默认路由，任何发往外部网络的流量都会通过这个默认网关发送。所有要将它删除。   2. 删除默认路由后，Host V 将无法再通过默认网关访问任何没有静态路由的网络。这会导致它无法直接回复来自 Host U 的 ping 请求，所有要新增静态路由。   3. 新增静态路由:添加一条指向192.168.53.0/24 的静态路由（通过 192.168.60.11），意味着 Host V 可以将发往192.168.53.0/24 网络的流量通过 192.168.60.11 这个网关发送。   4. ping 结果:192.168.60.11 是 VPN Server 的 IP，且 Host U 通过 VPN 已经成功与 Host V 建立了连接，ping 请求成功，Host V 将 ICMP Echo 回复通过新添加的静态路由传回 Host U。 |

### 2.7 Task 8: VPN Between Private Networks

拓扑图：



|  |
| --- |
| 任务：按要求搭建虚拟网络，以及完成相应的配置命令，运行客户端和服务器两个程序，并完成 ping/tetlnet实验  位置：HOST U / VPN Server |
| 命令   * 1. 启动新的docker   docker-compose -f docker-compose2.yml build  docker-compose -f docker-compose2.yml up   * 1. 修改tun\_server.py：   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tun = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TUN | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  os.system("ip route add 192.168.50.0/24 dev {}".format(ifname))  IP\_A = "0.0.0.0"  PORT = 9090  SERVER\_IP, SERVER\_PORT = '10.9.0.5', 9090  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  sock.bind((IP\_A, PORT))  while True:  # this will block until at least one interface is ready  ready, \_, \_ = select.select([sock, tun], [], [])    for fd in ready:  if fd is sock:  data, (SERVER\_IP, SERVER\_PORT) = sock.recvfrom(2048)  pkt = IP(data)  print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))  os.write(tun, bytes(pkt))  if fd is tun:  packet = os.read(tun, 2048)  pkt = IP(packet)  print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))  sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))   * 1. 运行tun.py和tun\_server.py   ④在hostU上ping hostV: ping 192.168.60.5 -c 1  ⑤进行wireshark抓包 |
| 截图 |
| 结果说明：   1. ping 命令向 192.168.60.5（Host V）发送 ICMP Echo 请求。请求通过 tun.py 程序中的 TUN 接口进入 VPN 通道，被封装成 UDP 数据包并发送到 tun\_server.py。服务器接收到 UDP 数据包后，将其解封并通过 TUN 接口转发到目标网络（192.168.60.0/24）。Host V 会收到并回复 ICMP Echo 请求，回复通过相同路径返回客户端 2. wireshark抓包结果：在服务器端，Wireshark 捕获到通过 TUN 接口传输的原始 IP 数据包。 |

### 2.8 Task 9: Experiment with the TAP Interface（\*5分）

|  |
| --- |
| 任务：按任务书要求使用TAP接口修改之前的client和server程序，并完成pign/telnet通信实验  位置：HOST U / VPN Server |
| 命令   1. 编写tap.py:   #!/usr/bin/env python3  import fcntl  import struct  import os  import time  from scapy.all import \*  TUNSETIFF = 0x400454ca  IFF\_TUN = 0x0001  IFF\_TAP = 0x0002  IFF\_NO\_PI = 0x1000  # Create the tun interface  tap = os.open("/dev/net/tun", os.O\_RDWR)  ifr = struct.pack('16sH', b'fan%d', IFF\_TAP | IFF\_NO\_PI)  ifname\_bytes = fcntl.ioctl(tap, TUNSETIFF, ifr)  # Get the interface name  ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")  print("Interface Name: {}".format(ifname))  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))  os.system("ip link set dev {} up".format(ifname))  # generate a corresponding ARP reply and write it to the TAP interface.  while True:  packet = os.read(tap, 2048)  if packet:  print("-------------------------------")  ether = Ether(packet)  print(ether.summary())    # Send a spoofed ARP response  FAKE\_MAC = "aa:bb:cc:dd:ee:ff"  if ARP in ether and ether[ARP].op == 1:  arp = ether[ARP]  newether = Ether(dst=ether.src, src=FAKE\_MAC)  newarp = ARP(psrc=arp.pdst, hwsrc=FAKE\_MAC, pdst=arp.psrc,hwdst=ether.src, op=2)  newpkt = newether/newarp    print("\*\*\*\*\* Fake response: {}".format(newpkt.summary()))  os.write(tap, bytes(newpkt))   1. 在server上测试tap.py 2. 在server上:arping -I fan0 192.168.53.33 -c 1 3. 再次运行：tap.py 4. 查看arping -I fan0 1.2.3.4 -c 1 |
| 截图 |
| 结果说明：   1. 看到 ARP 响应，MAC 地址为伪造的 aa:bb:cc:dd:ee:ff， TAP 接口成功捕获并伪造了 ARP 报文。 2. 第二个 arping 命令：看到相同的伪造 MAC 地址，尽管目标 IP 地址 1.2.3.4 在实际网络中不存在。 |

## 实验总结

1. 这次实验没有很复杂的操作，但量很大，需要比较细心和耐心地进行发送报文和接收报文的操作。
2. 在这次实验中，多次使用wireshark抓包工具，wireshark可以捕捉到报文的详细信息，是一个非常好用的工具。
3. 学会了使用 TAP 接口进行以太网帧的处理，并模拟了 ARP 欺骗攻击的基本原理。