Lab4-VPN

实验环境

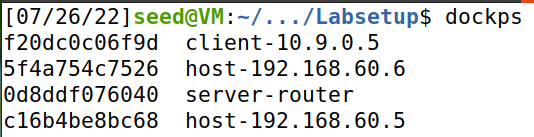
1. 进入实验目录

/home/seed/Desktop/Labs\_20.04/Network Security/VPN Tunneling Lab/Labsetup

1. 开启容器

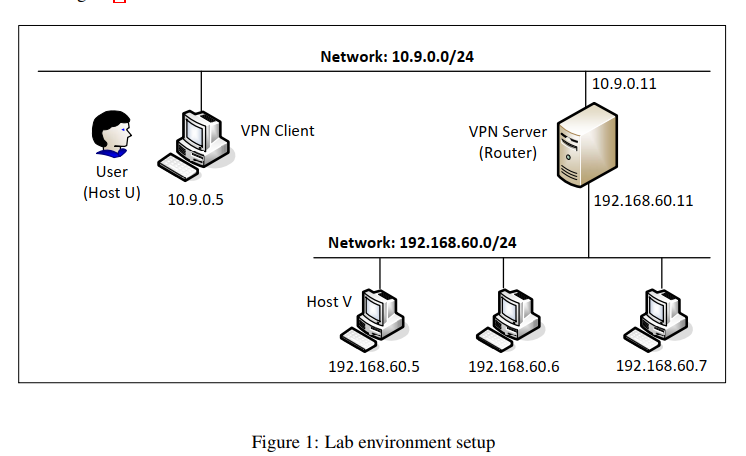
dcup

然后dockps 查看容器ID 之后docksh <ID>



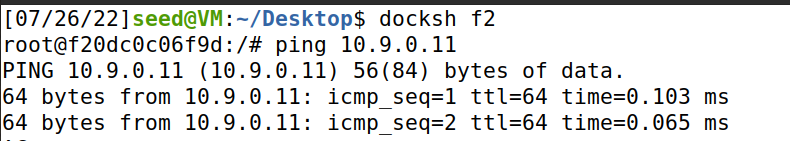
我们连接server-router、host-192.168.60.5、client-10.9.0.5三个容器

实验网络拓扑如下：

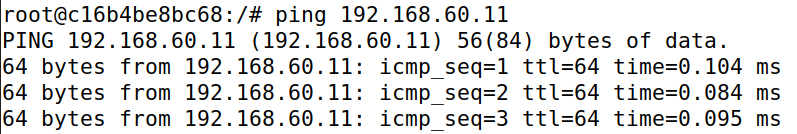


Task1: Testing Network

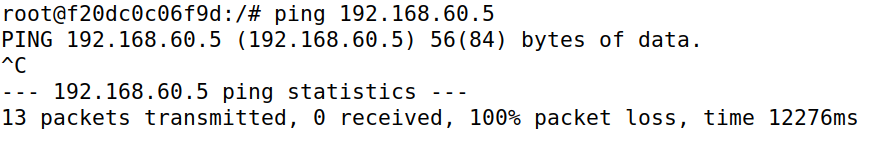
1. 在hostU上ping VPN server，测试两者通信



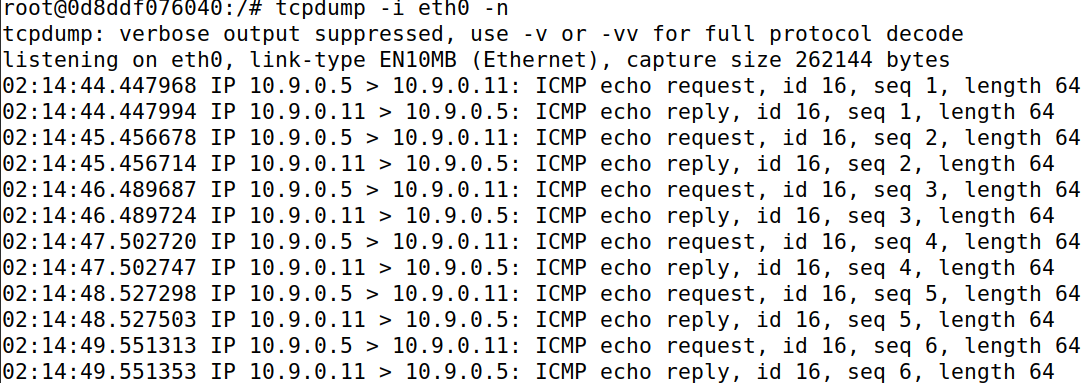
1. 在Host V(192.168.60.5)上ping VPN Server，测试两者通信。



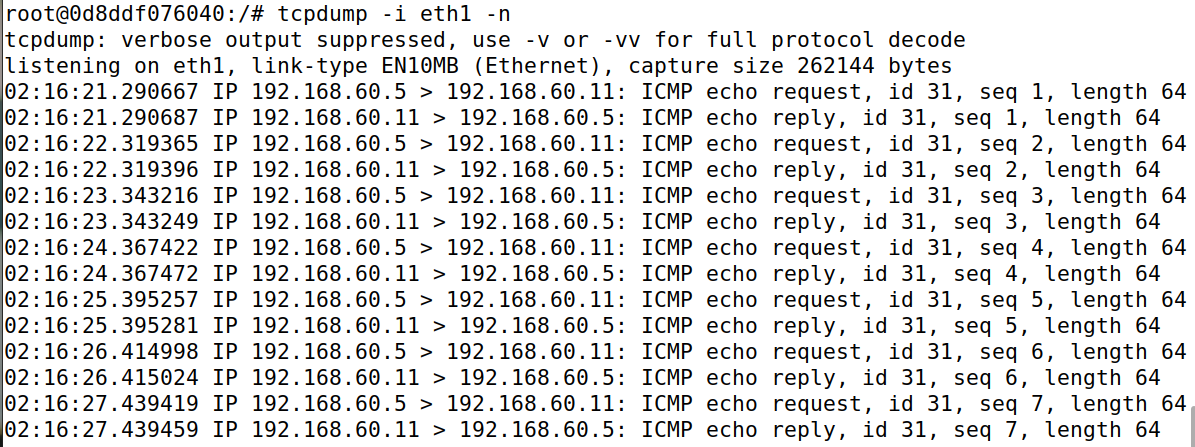
1. 在HostU上ping HostV，两者不可通信。



1. 在Server上运行tcpdump，监听eth0网卡，然后在Host U上ping Server，结果如下，



1. 在Server上运行tcpdump，监听**eth1**网卡，然后在Host V上ping Server，结果如下，



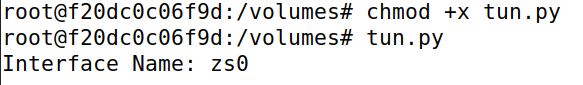
Task2: Create and Configure TUN Interface

Task 2.A: Name of the Interface

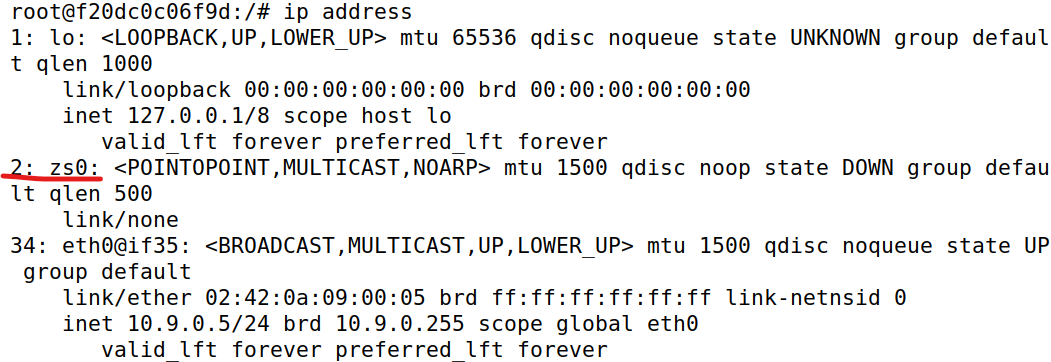
1. 将tun.py中的网卡名称tun修改为自己名字的首字母简写，这里以张三为例，修改为zs。



1. 在Host U上为tun.py添加运行权限并运行。



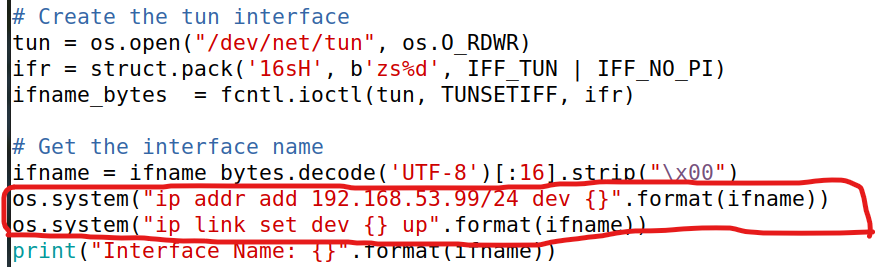
1. 另开一个shell进入Host U容器，查看容器内的网卡。



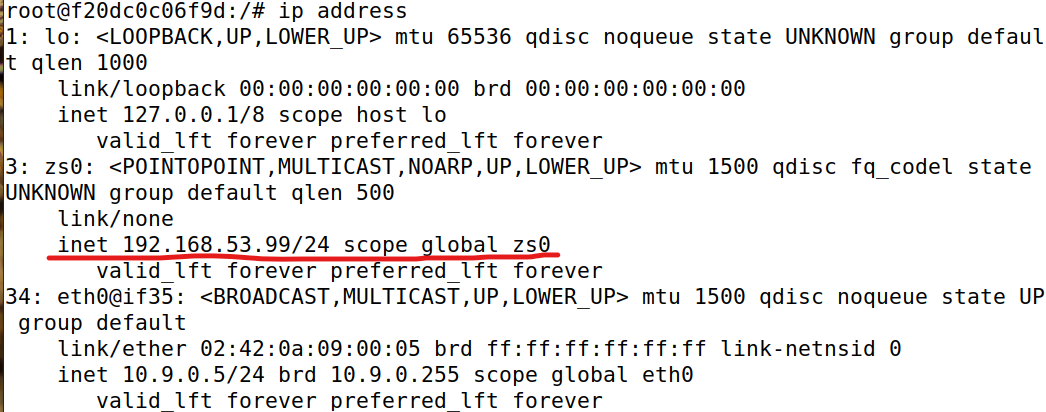
Task 2.b: Set up the TUN Interface

在网卡可用之前，我们需要做两件事，一是为其分配地址，而是开启网卡。

1. 在tun.py中添加以下红框内的代码分配IP地址



1. 再次在Host U上运行tun.py文件。
2. 另开shell在Host U上执行ip address查看结果：

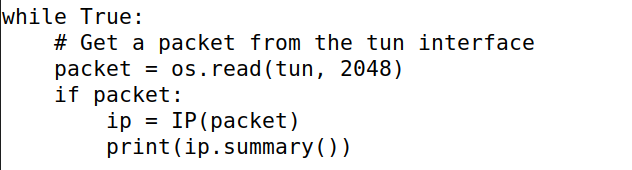


可以看到已经有地址了。

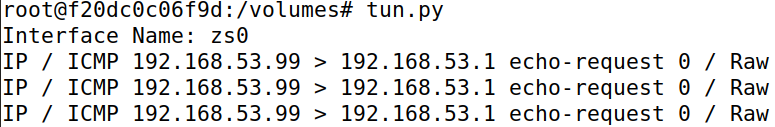
Task 2.c: Read from the TUN Interface

这个实验将会从TUN interface中读取数据。

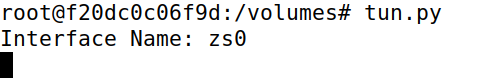
1. 修改tun.py文件，修改while循环



1. 在Host U上运行tun.py文件。
2. 另开shell在Host U上ping 192.168.53.0/24子网的地址，比如192.168.53.1，查看tun.py的输出:



1. 在shell上继续ping 192.168.60.0/24的地址，比如192.168.60.5，查看tun.py的输出。

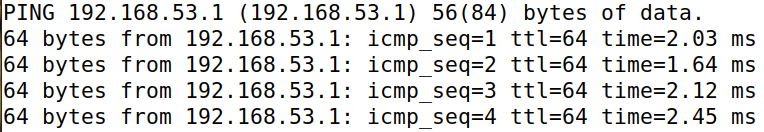


可以看到是没有输出的。

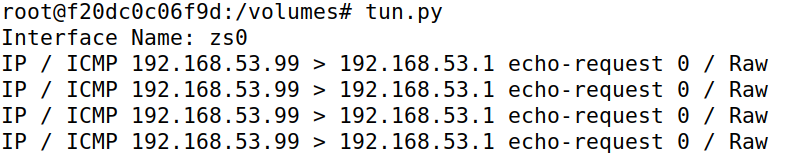
Task 2.d: Write to the TUN Interface

向TUN interface写入数据.

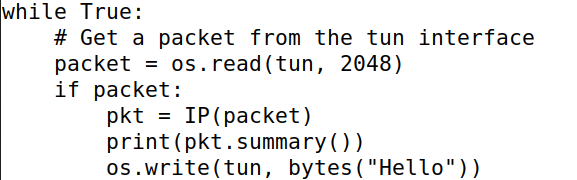
1. 修改tun.py while循环内容如下，收到ICMP request后写入echo reply packet。
2. **while** True:
3. # Get a packet from the tun interface
4. packet = os.read(tun, 2048)
5. **if** packet:
6. pkt = IP(packet)
7. **print**(pkt.summary())
8. **if** ICMP **in** pkt:
9. newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
10. newip.ttl = 64
11. newicmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
12. **if** pkt.haslayer(Raw):
13. data = pkt[Raw].load
14. newpkt = newip/newicmp/data
15. **else**:
16. newpkt = newip/newicmp
17. os.write(tun, bytes(newpkt))
18. 运行tun.py.
19. 另开shell在Host U上ping 192.168.53.1，结果如下：



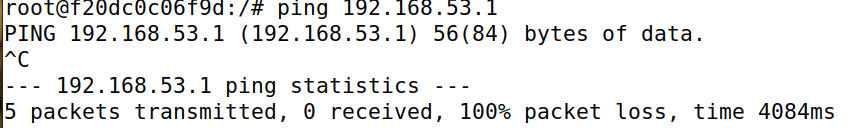
发现有回应了，Tun.py输出如下:



1. 再次修改while循环如下:



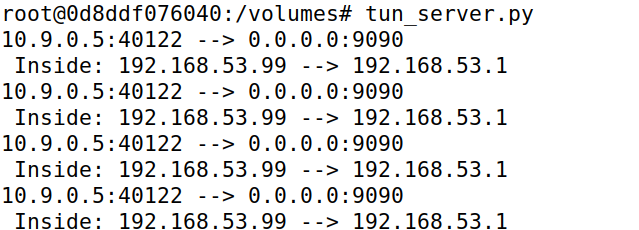
1. 再次运行tun.py
2. 另开shell 同样ping 192.168.55.1，结果如下:



发现ping不通了。

Task3: Send the IP Packet to VPN Server Through a Tunnel

1. 在VPN Server容器创建tun\_server.py文件，内容如下:
2. #!/usr/bin/env python3
3. **from** scapy.all **import** \*
4. IP\_A = "0.0.0.0"
5. PORT = 9090
6. sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
7. sock.bind((IP\_A, PORT))
8. **while** True:
9. data, (ip, port) = sock.recvfrom(2048)
10. **print**("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))
11. pkt = IP(data)
12. **print**(" Inside: {} --> {}".format(pkt.src, pkt.dst))
13. 在Host U上创建tun\_client.py文件，内容如下：
14. #!/usr/bin/env python3
16. **import** fcntl
17. **import** struct
18. **import** os
19. **import** time
20. **from** scapy.all **import** \*
22. TUNSETIFF = 0x400454ca
23. IFF\_TUN   = 0x0001
24. IFF\_TAP   = 0x0002
25. IFF\_NO\_PI = 0x1000
27. # Create the tun interface
28. tun = os.open("/dev/net/tun", os.O\_RDWR)
29. ifr = struct.pack('16sH', b'zs%d', IFF\_TUN | IFF\_NO\_PI)
30. ifname\_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
32. # Get the interface name
33. ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")
34. os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
35. os.system("ip link set dev {} up".format(ifname))
36. **print**("Interface Name: {}".format(ifname))
38. # Create UDP socket
39. sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
40. SERVER\_IP = "10.9.0.11"
41. SERVER\_PORT = 9090
42. **while** True:
43. # Get a packet from the tun interface
44. packet = os.read(tun, 2048)
45. **if** packet:
46. # Send the packet via the tunnel
47. sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))
48. 在VPN Server(10.9.0.11)上运行tun\_server.py，在Host U上运行tun\_client.py.
49. 然后另开shell，在Host U上Ping 192.168.53.0/24，比如192.168.53.1，然后查看VPN server的输出，如下所示:



解释: ping的流量进入zs0网卡之后被tun\_client.py程序读取，然后发送给了tun\_server。

1. 然后在Host U上继续ping 192.168.60.5也就是Host V，查看结果

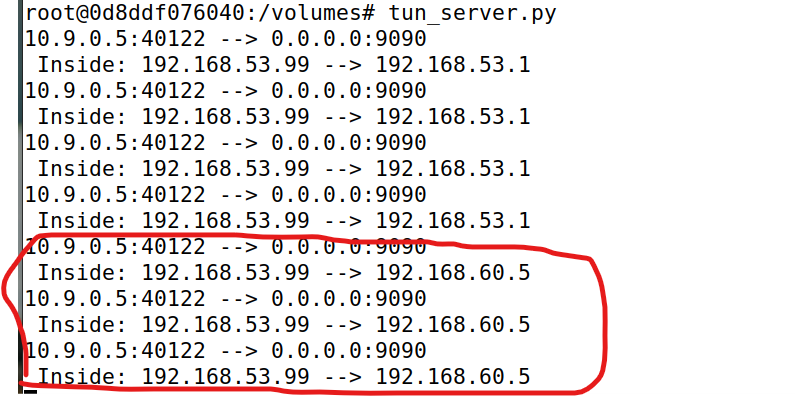
发现VPN Server没有输出了，因为流量没有被zs0网卡接收，也就送不到VPN server了。

1. 既然无法被网卡zs0接收，我们就自己设定路由，使流量转到zs0网卡上，在Host U上运行以下命令:

ip route add 192.168.60.0/24 dev zs0(自己的网卡，名字都不相同)

然后再ping 192.168.60.5，查看tun\_server.py的输出:

**注意，tun\_client.py一直都是运行状态**



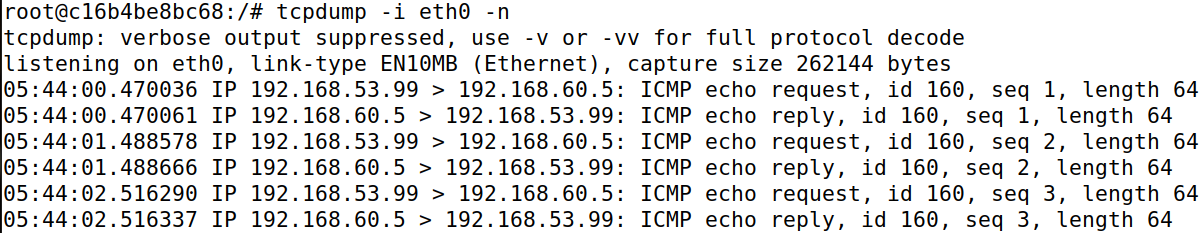
发现在VPN Server上tun\_server.py有了关于192.168.60.5的输出。

Task4 : Spoofing NS Records for Another Domain

1. 修改tun\_server.py文件，如下所示:
2. #!/usr/bin/env python3
4. **import** fcntl
5. **import** struct
6. **import** os
7. **import** time
8. **from** scapy.all **import** \*
10. TUNSETIFF = 0x400454ca
11. IFF\_TUN   = 0x0001
12. IFF\_TAP   = 0x0002
13. IFF\_NO\_PI = 0x1000
15. # Create the tun interface
16. tun = os.open("/dev/net/tun", os.O\_RDWR)
17. ifr = struct.pack('16sH', b'zs%d', IFF\_TUN | IFF\_NO\_PI)
18. ifname\_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
20. # Get the interface name
21. ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")
22. os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
23. os.system("ip link set dev {} up".format(ifname))
24. **print**("Interface Name: {}".format(ifname))
26. IP\_A = "0.0.0.0"
27. PORT = 9090
28. sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
29. sock.bind((IP\_A, PORT))
30. **while** True:
31. data, (ip, port) = sock.recvfrom(2048)
32. **print**("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))
33. pkt = IP(data)
34. **print**(" Inside: {} --> {}".format(pkt.src, pkt.dst))
35. os.write(tun, bytes(pkt))
36. 在VPN Server上运行tun\_server.py文件
37. 在Host U上运行tun\_client.py文件。
38. 在Host V(192.168.60.5)上开启监听:

Tcpdump -i eth0 -n

1. 另开shell，在Host U上ping 192.168.60.5，查看Host V上是否有输出：

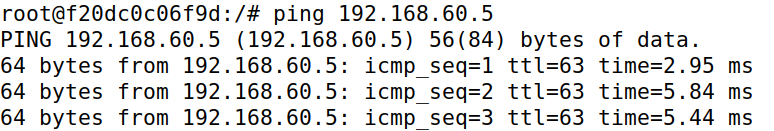


Task 5: Handling Traffic in Both Directions

1. 修改tun\_server.py，内容如下:
2. #!/usr/bin/env python3
4. **import** fcntl
5. **import** struct
6. **import** os
7. **import** time
8. **from** scapy.all **import** \*
9. **import** select
11. TUNSETIFF = 0x400454ca
12. IFF\_TUN   = 0x0001
13. IFF\_TAP   = 0x0002
14. IFF\_NO\_PI = 0x1000
16. # Create the tun interface
17. tun = os.open("/dev/net/tun", os.O\_RDWR)
18. ifr = struct.pack('16sH', b'zs%d', IFF\_TUN | IFF\_NO\_PI)
19. ifname\_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
21. # Get the interface name
22. ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")
23. os.system("ip addr add 192.168.53.11/24 dev {}".format(ifname))
24. os.system("ip link set dev {} up".format(ifname))
25. **print**("Interface Name: {}".format(ifname))
27. IP\_A = "0.0.0.0"
28. PORT = 9090
29. sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
30. sock.bind((IP\_A, PORT))
31. fds = [sock, tun]
32. **while** True:
33. ready, \_, \_ = select.select([sock,tun], [], [])
34. **for** fd **in** ready:
35. **if** fd **is** sock:
36. data, (ip, port) = sock.recvfrom(2048)
37. **print**("{}:{} --> {}:{}".format(ip, port, IP\_A, PORT))
38. pkt = IP(data)
39. #print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
40. os.write(tun, bytes(pkt))
41. **if** fd **is** tun:
42. packet = os.read(tun, 2048)
43. pkt = IP(packet)
44. **print**("Return {} -> {}".format(pkt.src, pkt.dst))
45. sock.sendto(packet, (ip, port))
46. 修改tun\_client.py，内容如下:
47. #!/usr/bin/env python3
49. **import** fcntl
50. **import** struct
51. **import** os
52. **import** time
53. **from** scapy.all **import** \*
54. **import** select
56. TUNSETIFF = 0x400454ca
57. IFF\_TUN   = 0x0001
58. IFF\_TAP   = 0x0002
59. IFF\_NO\_PI = 0x1000
61. # Create the tun interface
62. tun = os.open("/dev/net/tun", os.O\_RDWR)
63. ifr = struct.pack('16sH', b'zs%d', IFF\_TUN | IFF\_NO\_PI)
64. ifname\_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
66. # Get the interface name
67. ifname = ifname\_bytes.decode('UTF-8')[:16].strip("\x00")
68. os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
69. os.system("ip link set dev {} up".format(ifname))
70. **print**("Interface Name: {}".format(ifname))
72. # Create UDP socket
73. sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)
74. SERVER\_IP = "10.9.0.11"
75. SERVER\_PORT = 9090
76. fds = [sock, tun]
77. **while** True:
78. ready, \_, \_ = select.select([sock, tun], [], [])
79. **for** fd **in** ready:
80. **if** fd **is** sock:
81. data, (ip, port) = sock.recvfrom(2048)
82. pkt = IP(data)
83. **print**("From socket: {} --> {}".format(pkt.src, pkt.dst))
84. os.write(tun, data)
85. **if** fd **is** tun:
86. packet = os.read(tun, 2048)
87. **if** packet:
88. pkt = IP(packet)
89. **print**(pkt.summary())
90. sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))
91. 在VPN Server上运行tun\_server.py
92. 在Host U上运行tun\_client.py
93. 在ping 192.168.60.5之前，确保ip route中有关于192.168.60.0/24的条目，如果没有就自己加上，例如:

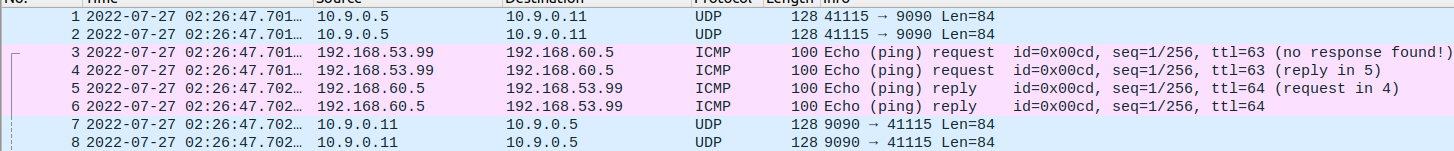
Ip route add 192.168.60.0/24 dev zs0(自己的网卡名称)

1. 在Host U上ping 192.168.60.5，结果如下:



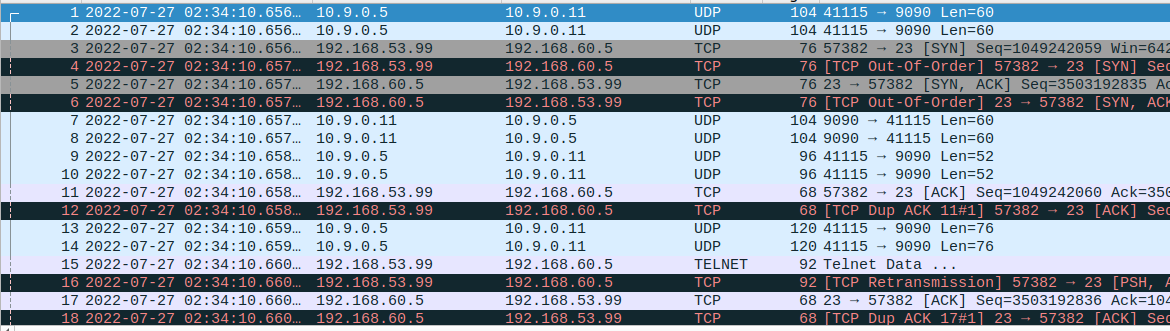
发现有回复了。

1. 开启wireshark，选择any 网卡选项，然后再次ping 192.168.60.5，wiresahrk结果如下:



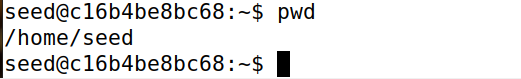
可以看出整个流是从10.9.0.5 -> 10.9.0.11 -> 192.168.60.5的。

1. 从Host U telnet连接Host V时也是，如下所示:



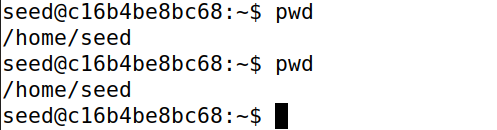
Task 6: Tunnel-Breaking Experiment

1. 像上个实验那样从Host U发起对Host V的telnet连接
2. 终止tun\_client.py程序的运行。
3. 在Host U上输入一些内容，然后看是否有结果



发现命令不会显示

1. 重新运行tun\_client.py
2. 在Host U的telnet连接中看结果如何



发现刚才输入的命令重现了。