Advanced Computer Networking and Security

VPN Tunneling Lab

Name: Sainagadurgadevi Penumatsa

Student ID: 700728723

Task 1: Network Setup

Setting up the lab environment

```
[11/14/21]seed@VM:~$ cd Lab/VPN/Labsetup/
[11/14/21]seed@VM:~/.../Labsetup$ dcbuild
VPN_Client uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Router uses an image, skipping
[11/14/21]seed@VM:~/.../Labsetup$ dcup -d
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating host-192.168.60.6 ... done
Creating client-10.9.0.5   ... done
Creating host-192.168.60.5 ... done
Creating server-router     ... done
```

Host U can communicate with VPN Server. I am able to ping VPN Server from host U container and receive response from VPN Server (10.9.0.11). Also, I am able to sniff the packets on VPN Server container interface eth0 using tcpdump.

```
[11/22/21]seed@VM:~/.../Labsetup$ settitle hostU
[11/22/21]seed@VM:~/.../Labsetup$ docksh e2
root@e2ab05a57c60:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.239 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.124 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.246 ms
```

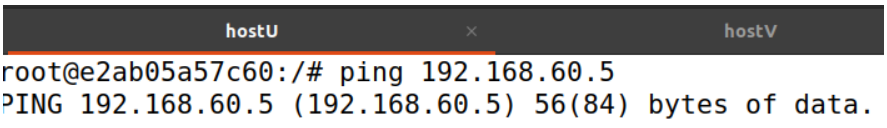
```
root@00825dc39b2c:/# tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
03:07:46.695970 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 1, length 64
03:07:46.695991 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 1, length 64
03:07:47.703564 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 2, length 64
03:07:47.703627 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 2, length 64
03:07:48.727162 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 16, seq 3, length 64
03:07:48.727214 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 16, seq 3, length 64
^C
```

VPN server can communicate with host V. I am able to ping Host V from VPN Server container and receive response from Host V container (192.168.60.5). I can sniff packets as server on interface eth1 using tcpdump.

```
root@00825dc39b2c:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.136 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.077 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.145 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.189 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.063 ms
```

```
root@00825dc39b2c:/# tcpdump -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
03:09:53.124438 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 34, seq 1, leng
03:09:53.124477 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 34, seq 1, length
03:09:54.136036 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 34, seq 2, leng
03:09:54.136082 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 34, seq 2, length
03:09:55.161367 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 34, seq 3, leng
03:09:55.161460 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 34, seq 3, length
03:09:56.185770 IP 192.168.60.11 > 192.168.60.5: ICMP echo request, id 34, seq 4, leng
03:09:56.185890 IP 192.168.60.5 > 192.168.60.11: ICMP echo reply, id 34, seq 4, length
```

Host U is not able to communicate with Host V. I do not receive response for the ping command to 192.168.60.5 on Host U container. On Server container, the packets are not captured on both eth0, eth1 interfaces.




```
root@00825dc39b2c:/# tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

```
root@00825dc39b2c:/# tcpdump -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

Changed the interface name in the program to prefix with my lastname 'penum'

```
1 #!/usr/bin/env python3
2
3 import fcntl
4 import struct
5 import os
6 import time
7 from scapy.all import *
8
9 TUNSETIFF = 0x400454ca
10 IFF_TUN   = 0x0001
11 IFF_TAP   = 0x0002
12 IFF_NO_PI = 0x1000
13
14 # Create the tun interface
15 tun = os.open("/dev/net/tun", os.O_RDWR)
16 ifr = struct.pack('16sH', b'penumatsa%d', IFF_TUN | IFF_NO_PI)
17 ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
18
19 # Get the interface name
20 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
21 print("Interface Name: {}".format(ifname))
22
23 while True:
24    time.sleep(10)
25
```

Switch to host U container (10.9.0.5) and run the python program. The interface name is shown as 'penum0'.

```
root@112bb6b20b28:/volumes# ls
Task2.a.py  tun.py
root@112bb6b20b28:/volumes# chmod a+x Task2.a.py
root@112bb6b20b28:/volumes# python3 Task2.a.py
Interface Name: penum0
```

In another shell of hostU, the ip address command lists the interface name 'penum0'

```
root@112bb6b20b28:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: penum0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@112bb6b20b28:/#
```

Task 2.b: Set up the TUN Interface

Added the below lines of code in the python program to configure the IP address to the interface we created and to start the interface.
os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

```python
#!/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'penum%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:
   time.sleep(10)
```

Run the above python program on the host U container.

```
root@112bb6b20b28:/volumes# chmod a+x Task2.b.py
root@112bb6b20b28:/volumes# python3 Task2.b.py
Interface Name: penum0
```

Ip address command now shows the interface is UP and IP address 192.168.53.99/24 we configured in the python code.

```
root@112bb6b20b28:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
5: penum0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group defaul
    link/none
    inet 192.168.53.99/24 scope global penum0
       valid_lft forever preferred_lft forever
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@112bb6b20b28:/#
```

Task 2.c: Read from the TUN Interface

```
19 # Get the interface name
20 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
21 print("Interface Name: {}".format(ifname))
22 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
23 os.system("ip link set dev {} up".format(ifname))
24
25 while True:
26 # Get a packet from the tun interface
27         packet = os.read(tun, 2048)
28         if packet:
29                 ip = IP(packet)
30                 print(ip.summary())
31
```

Run the python code on Host U container.

```
root@112bb6b20b28:/volumes# python3 Task2.c.py
Interface Name: penum0
```

On host U, ping a host in the 192.168.53.0/24 network

```
root@112bb6b20b28:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

Tun program is picking these ICMP packets as the tun interface (192.168.53.99) and the IP address we are pinging i.e., 192.168.53.1 are on the same subnet. As

the packets are passing through the tun interface, the tun interface IP address(192.168.53.99) shows as the sender's IP address in the below screenshot.

```
root@112bb6b20b28:/volumes# python3 Task2.c.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

On Host U, ping a host in the internal network 192.168.60.0/24.

```
root@112bb6b20b28:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
```

```
root@112bb6b20b28:/volumes# python3 Task2.c.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

The tun interface does not pick up these packets because only the packets sent to subnet 192.168.53.0/24 are passed through tun interface as per the routing table and the packets sent to other subnets are passed through the default interface which is eth0 interface. As we are listening on tun interface as part of the program, the packets are not showing up in the tun interface.

```
root@112bb6b20b28:/# ip r
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.53.0/24 dev penum0 proto kernel scope link src 192.168.53.99
root@112bb6b20b28:/#
```

Task 2.d: Write to the TUN Interface

Spoofing a ICMP reply packet and writing to the Tun interface

Python code to capture the ICMP echo request packets and spoof the reply packets and write them into the tun interface.

```
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 and ip[ICMP].type ==8:
                        newip = IP(src = ip.dst, dst=ip.src)
                        icmp = ICMP(type=0,id = ip[ICMP].id, seq = ip[ICMP].seq)
                        newpkt = newip/icmp/ip[Raw].load
                        os.write(tun, bytes(newpkt))
```

Running the python program to write the packets to the tun interface.

```
root@112bb6b20b28:/volumes# python3 Task2.d.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

Ping a non-existing IP address 192.168.53.1. Ideally, we should not receive any reply packets. Because we have spoofed the ICMP reply packets as part of our python code, we could see the response packets from 192.168.53.1

```
root@112bb6b20b28:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
64 bytes from 192.168.53.1: icmp_seq=1 ttl=64 time=1.92 ms
64 bytes from 192.168.53.1: icmp_seq=2 ttl=64 time=2.89 ms
64 bytes from 192.168.53.1: icmp_seq=3 ttl=64 time=2.58 ms
64 bytes from 192.168.53.1: icmp_seq=4 ttl=64 time=4.40 ms
64 bytes from 192.168.53.1: icmp_seq=5 ttl=64 time=5.08 ms
```

Writing the arbitrary data to the Tun interface

Python program to write the ‘data’ to the Tun interface

```python
#!/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'penum%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:
                print(bytes(b'data'))
                os.write(tun, bytes(b'data'))

```

Run the Task2.d.2.py in the Host U container

```
root@e2ab05a57c60:/volumes# python3 Task2.d.2.py
Interface Name: penum0
b'data'
b'data'
b'data'
b'data'
```

On host U, ping 192.168.53.1. Reply packet is not seen as the reply packets are not spoofed and this is a non-existing IP. The data is sent via the tun interface as shown in the above screenshot.

```
root@e2ab05a57c60:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

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

Python Program to implement the tun client to send the packets through the tun interface to the VPN server

Task3_tun_client.py

```
SERVER_IP = "10.9.0.11"
SERVER_PORT = 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'penum%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))

# 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:
                pkt = IP(packet)
                print(pkt.summary())

                # Send the packet via the tunnel
                sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Python program to implement the tun server to receive the packets from the VPN client through the tun interface

```
 6 import time
 7 from scapy.all import *
 8
 9 IP_A = "0.0.0.0"
10 PORT = 9090
11
12 TUNSETIFF = 0x400454ca
13 IFF_TUN   = 0x0001
14 IFF_TAP   = 0x0002
15 IFF_NO_PI = 0x1000
16
17 # Create the tun interface
18 tun = os.open("/dev/net/tun", os.O_RDWR)
19 ifr = struct.pack('16sH', b'penum%d', IFF_TUN | IFF_NO_PI)
20 ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
21
22 # Get the interface name
23 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
24 print("Interface Name: {}".format(ifname))
25 os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))
26 os.system("ip link set dev {} up".format(ifname))
27
28 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
29 sock.bind((IP_A, PORT))
30
31 while True:
32         data, (ip, port) = sock.recvfrom(2048)
33         print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
34         pkt = IP(data)
35         print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
36         os.write(tun, data)
```

Ran the Task3_tun_client.py program on the VPN Client (Host U) container

```
root@112bb6b20b28:/volumes# python3 Task3_tun_client.py
Interface Name: penum0
```

Ran the Task3_tun_server.py program on the VPN Server container

```
root@d165d78418ec:/volumes# python3 Task3_tun_server.py
Interface Name: penum0
```

From VPN Client (Host U) container, pinging IP address 192.168.53.1 belonging to the 192.168.53.0/24 subnet

```
root@112bb6b20b28:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
```

On the VPN Client (Host U) the packet is picked up by the tun interface and the source IP address shows as 192.168.53.99 which is tun interface IP address.

```
root@112bb6b20b28:/volumes# python3 Task3_tun_client.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
```

The packets sent by the VPN client using the tun interface are received on the VPN Server and the original IP packet (192.168.53.99->192.168.53.1) is carried as a payload inside the IP packet(10.9.0.5->0.0.0.0)

```
root@d165d78418ec:/volumes# python3 Task3_tun_server.py
Interface Name: penum0
10.9.0.5:49340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:49340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:49340 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.9.0.5:49340 --> 0.0.0.0:9090
```

Modify the Task3_tun_client.py program by adding a routing entry to redirect the packets sent to 192.168.60.0/24 subnet to the tun interface.

Task3_tun_client.py

```
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'penum%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))

# Set up routing
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))

# 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:
                pkt = IP(packet)
                print(pkt.summary())

                # Send the packet via the tunnel
                sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

Run the Task3_tun_client program on the VPN Client(Host U) container and the Task3_tun_Server program on the VPN Server container

```
root@112bb6b20b28:/volumes# python3 Task3_tun_client.py
Interface Name: penum0
```

```
root@d165d78418ec:/volumes# python3 Task3_tun_server.py
Interface Name: penum0
```

Ping host V IP address 192.168.60.5 on the hostU container

```
root@112bb6b20b28:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

On VPN client observed that the IP packets travel through the tun interface with the sender's IP Address as 192.168.53.99

```
root@112bb6b20b28:/volumes# python3 Task3_tun_client.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

On VPN Server, the packets from the VPN client were received by the tun interface on the VPN Server. The original IP packet(192.168.53.99 -> 192.168.60.5) is carried as a payload inside the IP packet(10.9.0.5->0.0.0.0)

```
root@d165d78418ec:/volumes# python3 Task3_tun_server.py
Interface Name: penum0
10.9.0.5:36634 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:36634 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:36634 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:36634 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:36634 --> 0.0.0.0:9090
```

Task 4: Set Up the VPN Server

Run the Task3_tun_client.py on the Client container(10.9.0.5)

```
root@e2ab05a57c60:/volumes# python3 Task3_tun_client.py
Interface Name: penum0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

Run the Task3_tun_server.py program on the Server container

```
root@00825dc39b2c:/volumes# python3 Task3_tun_server.py
Interface Name: penum0
10.9.0.5:57976 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:57976 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:57976 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:57976 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:57976 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.60.5
10.9.0.5:57976 --> 0.0.0.0:9090
```

Ping host V from host U container

```
root@e2ab05a57c60:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
```

On host V container, ran the tcpdump and the ICMP packets from 192.168.53.99 are reached to 192.168.60.5 through the tunnel.

```
root@b145ef80f305:/# tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
03:58:54.622239 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 89, seq 1, length 64
03:58:54.622299 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 89, seq 1, length 64
03:58:55.641487 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 89, seq 2, length 64
03:58:55.641516 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 89, seq 2, length 64
03:58:56.668449 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 89, seq 3, length 64
03:58:56.668501 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 89, seq 3, length 64
03:58:57.692530 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 89, seq 4, length 64
03:58:57.692664 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 89, seq 4, length 64
03:58:58.714994 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 89, seq 5, length 64
03:58:58.715023 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 89, seq 5, length 64
```

Task 5: Handling Traffic in Both Directions

Run the Task5_tun_client_select.py program on the VPN Client container

```
root@e2ab05a57c60:/volumes# python3 Task5_tun_client_select.py
Interface Name: penum0
```

Run the Task5_tun_server_select.py program on the VPN Server container

```
root@00825dc39b2c:/volumes# python3 Task5_tun_server_select.py
Interface Name: penum0
```

Ping host V 192.168.60.5 from the Host U container

The reply packets are sent from the host V machine 192.168.60.5 to the Host U and the ttl value is reduced by 1 and shows as 63 as the packets passes via VPN server which is acting as a router.

```
root@e2ab05a57c60:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=811 ttl=63 time=18.6 ms
64 bytes from 192.168.60.5: icmp_seq=812 ttl=63 time=10.0 ms
64 bytes from 192.168.60.5: icmp_seq=813 ttl=63 time=13.7 ms
64 bytes from 192.168.60.5: icmp_seq=814 ttl=63 time=10.4 ms
64 bytes from 192.168.60.5: icmp_seq=815 ttl=63 time=10.3 ms
64 bytes from 192.168.60.5: icmp_seq=816 ttl=63 time=12.5 ms
```

Below is the screenshot showing the packets passing through the tun interface from the VPN Client with sender's IP address as 192.168.53.99

```
root@e2ab05a57c60:/volumes# python3 Task5_tun_client_select.py
Interface Name: penum0
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
```

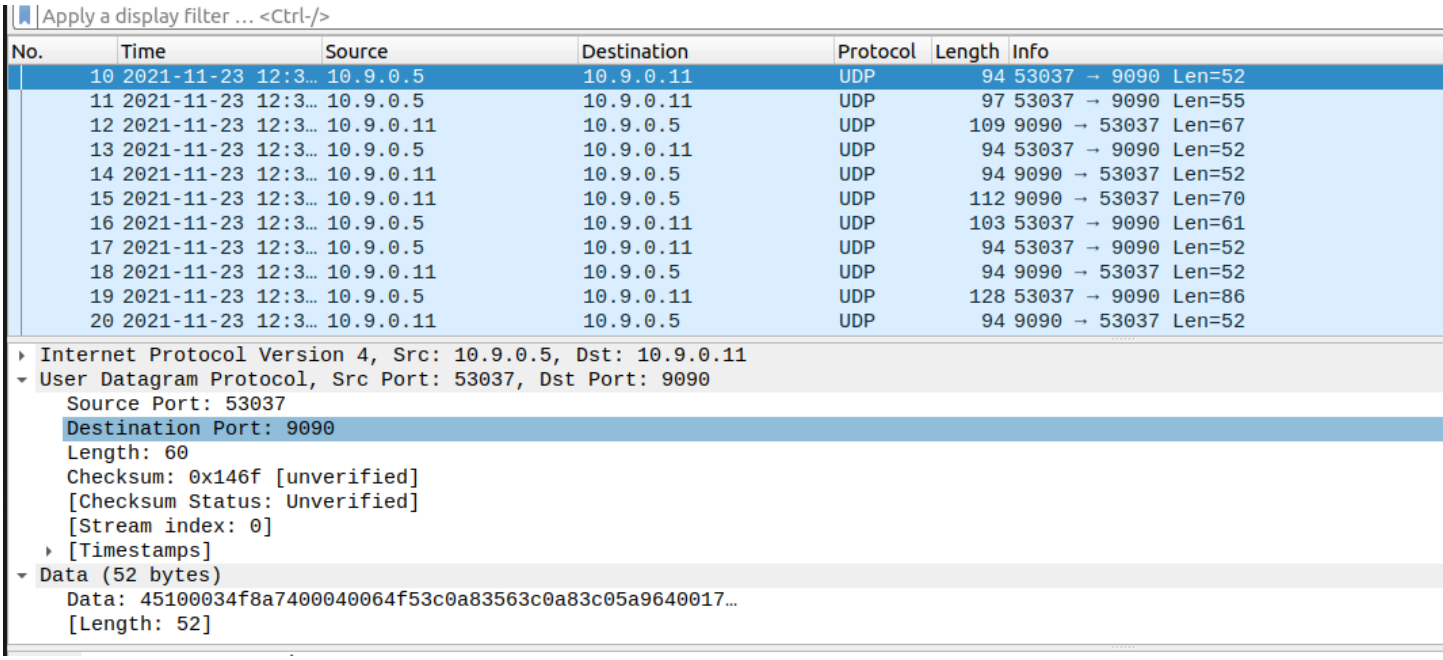
Screenshot showing the packets are received on VPN Server through the tun interface. The original packet 192.168.53.99 -> 192.168.60.5 is carried as data inside the IP packet 10.9.0.5->0.0.0.0

```
root@00825dc39b2c:/volumes# python3 Task5_tun_server_select.py
Interface Name: penum0
sock...
10.9.0.5:54691 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:54691 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:54691 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:54691 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
```

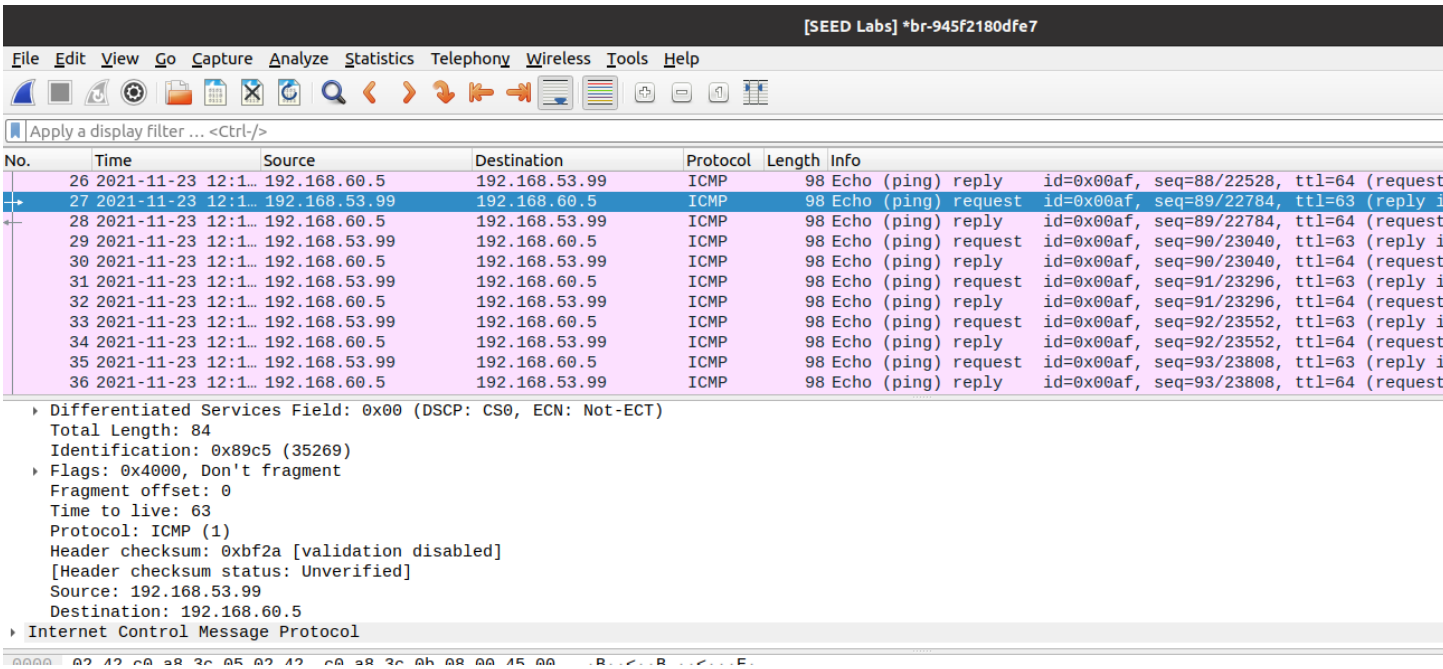
Below is the screenshot of tcpdump on host V that the original IP packets 192.168.53.99->192.168.60.5 are received.

```
root@b145ef80f305:/# tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
05:28:03.585832 IP6 fe80::f018:96ff:fecb:9bb1 > ff02::2: ICMP6, router solicitation, length 16
05:34:36.802577 IP6 fe80::42:3cff:fe5e:ddf6 > ff02::2: ICMP6, router solicitation, length 16
05:39:04.206771 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 102, seq 811, length 64
05:39:04.207221 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 102, seq 811, length 64
05:39:05.202068 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 102, seq 812, length 64
05:39:05.202243 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 102, seq 812, length 64
05:39:06.209592 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 102, seq 813, length 64
05:39:06.209711 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 102, seq 813, length 64
```

Wireshark screenshot showing the UDP packets from 10.9.0.5 to 10.9.0.11 with ICMP packets as data inside UDP packets



Below is the screenshot of the Wireshark for the ping command showing the ICMP Echo requests from 192.168.53.99 -> 192.168.60.5 and the ICMP reply packets from 192.168.60.5 -> 192.168.53.99 stripping off the extra UDP headers



Telnet connection from host U to host V

Run the Task5_tun_client_select.py on VPN Client container and Task5_tun_server_select.py on VPN Server container.

On host U container, telnet to host V(192.168.60.5) and enter the username and password.

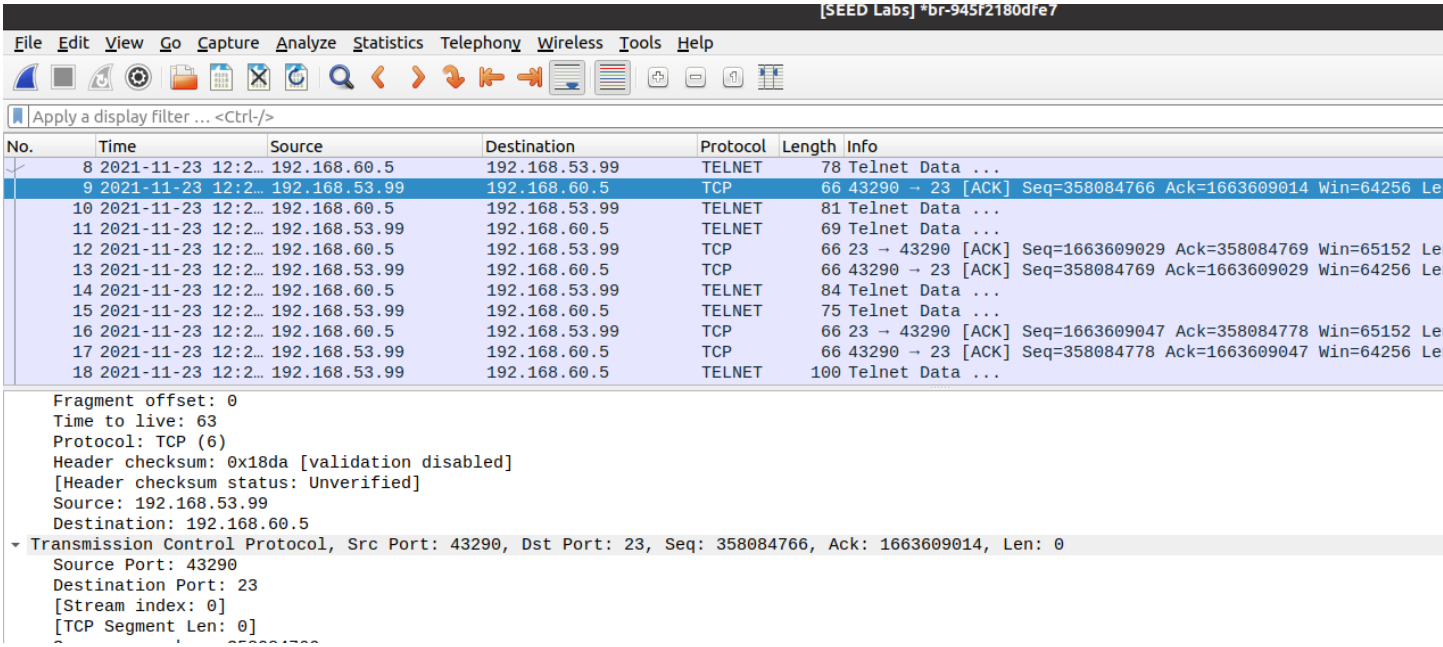
```
root@e2ab05a57c60:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b145ef80f305 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Nov 23 06:00:09 UTC 2021 from 192.168.53.99 on pts/2
seed@b145ef80f305:~$
```

Screenshot of the Wireshark from the Telnet command depicting the packets captured from 192.168.53.99->192.168.60.5 and 192.168.60.5->192.168.53.99



Task 6: Tunnel-Breaking Experiment

Run the Task5_tun_client_select.py program on the VPN Client container

```
root@e2ab05a57c60:/volumes# python3 Task5_tun_client_select.py
Interface Name: penum0
From tun ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
```

Run the Task5_tun_server_select.py on the VPN Server container.

```
root@00825dc39b2c:/volumes# python3 Task5_tun_server_select.py
Interface Name: penum0
sock...
10.9.0.5:47609 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:47609 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
sock...
10.9.0.5:47609 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
```

Telnet from Host U to Host V. The telnet connection is successful.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle client
[11/23/21]seed@VM:~/.../Labsetup$ docksh e2
root@e2ab05a57c60:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b145ef80f305 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Nov 23 17:35:23 UTC 2021 from 192.168.53.99 on pts/4
seed@b145ef80f305:~$
```

Stopped the Task5_tun_client_select.py program on the VPN Client container by pressing Ctrl+C keys.

Typed some random characters in the telnet window. The characters typed are not seen on the window because the connection is broken.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle client
[11/23/21]seed@VM:~/.../Labsetup$ docksh e2
root@e2ab05a57c60:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b145ef80f305 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Nov 23 17:35:23 UTC 2021 from 192.168.53.99 on pts/4
seed@b145ef80f305:~$
```

Reconnecting the tunnel again by running the Task5_tun_client_select.py program on VPN Client container.

The tunnel is re-established, and the telnet connection is reconnected, and the characters typed when the connection is broken are buffered and sent out and shown on the telnet window as soon as the connection is reconnected.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle client
[11/23/21]seed@VM:~/.../Labsetup$ docksh e2
root@e2ab05a57c60:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b145ef80f305 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Tue Nov 23 17:35:23 UTC 2021 from 192.168.53.99 on pts/4
seed@b145ef80f305:~$ dkjfakldsjfldasjfiueoirusdlkfjuoewrjkldsfjioesud
```

Task 7: Routing Experiment on Host V

On Host V, deleted the default entry and added the routing entry to route all the packets going to network 192.168.53.0/24 through router IP address 192.168.60.11

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle hostV
[11/23/21]seed@VM:~/.../Labsetup$ docksh a1
root@a1d76ff4ff6b:/# ip r
default via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@a1d76ff4ff6b:/# ip route del default
root@a1d76ff4ff6b:/# ip r
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@a1d76ff4ff6b:/# ip route add 192.168.53.0/24 via 192.168.60.11
root@a1d76ff4ff6b:/# ip r
192.168.53.0/24 via 192.168.60.11 dev eth0
192.168.60.0/24 dev eth0 proto kernel scope link src 192.168.60.5
root@a1d76ff4ff6b:/#
```

Ran the Task5_tun_client_select.py on host U container

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle hostU
[11/23/21]seed@VM:~/.../Labsetup$ docksh e6
root@e68b96c2749e:/# cd volumes/
root@e68b96c2749e:/volumes# python3 Task5_tun_client_select.py
Interface Name: penum0
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun ==>: 192.168.53.99 --> 192.168.60.5
```

Ran the Task5_tun_server_select.py on VPN Server container

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle VPNServer
[11/23/21]seed@VM:~/.../Labsetup$ docksh 86
root@8690d4a6da95:/# cd volumes/
root@8690d4a6da95:/volumes# python3 Task5_tun_server_select.py
Interface Name: penum0
sock...
10.9.0.5:46826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:46826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.53.99
sock...
10.9.0.5:46826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.53.99 --> 192.168.60.5
```

On Client, ping host V 192.168.60.5 and we see the reply packets from the host V machine with the ttl value as 63.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle client
[11/23/21]seed@VM:~/.../Labsetup$ docksh e6
root@e68b96c2749e:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=13.0 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=7.47 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=10.4 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=10.1 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=7.32 ms
```

On VPN Server, the ICMP packets are captured using tcpdump. This means the routing entries are setup correctly.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle VPNServer
[11/23/21]seed@VM:~/.../Labsetup$ docksh 86
root@8690d4a6da95:/# tcpdump -n -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
21:29:37.990703 ARP, Request who-has 192.168.60.5 tell 192.168.60.11, length 28
21:29:37.990833 ARP, Reply 192.168.60.5 is-at 02:42:c0:a8:3c:05, length 28
21:29:37.990840 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 29, seq 1, length 6
21:29:37.990901 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 29, seq 1, length 64
21:29:38.991529 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 29, seq 2, length 6
21:29:38.991602 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 29, seq 2, length 64
21:29:39.996936 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 29, seq 3, length 6
21:29:39.997099 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 29, seq 3, length 64
```

Task 8: VPN Between Private Networks

Setting up the VPN between private networks

```
[11/23/21]seed@VM:~/.../Labsetup$ docker-compose -f docker-compose2.yml build
HostA uses an image, skipping
HostB uses an image, skipping
VPN_Client uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Router uses an image, skipping
[11/23/21]seed@VM:~/.../Labsetup$ docker-compose -f docker-compose2.yml up -d
Creating network "net-192.168.50.0" with the default driver
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating server-router     ... done
Creating host-192.168.50.5 ... done
Creating host-192.168.60.5 ... done
Creating client-10.9.0.5   ... done
Creating host-192.168.50.6 ... done
Creating host-192.168.60.6 ... done
[11/23/21]seed@VM:~/.../Labsetup$ dockps
036c361e361d  host-192.168.50.6
b67b19b7477a  host-192.168.60.6
67fefabb36ee  client-10.9.0.5
95b3064d908a  host-192.168.60.5
ec1c6f1da079  host-192.168.50.5
5c81c32e1be6  server-router
[11/23/21]seed@VM:~/.../Labsetup$
```

We already have added a routing entry in the client program to route the packets going to network 192.168.60.0/24 to go via the tun interface.

```
# 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))

# Set up routing
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))

# Create UDP socket
```

As part of this task, I have extended the existing Task5_tun_server_select.py program to create Task8_tun_server_select.py and added the below routing entry in the server program to route the packets going to 192.168.50.0/24 to go via tun interface.

os.system("ip route add 192.168.50.0/24 dev {}".format(ifname))

Task8_tun_server_select.py

```
16
17 # Create the tun interface
18 tun = os.open("/dev/net/tun", os.O_RDWR)
19 ifr = struct.pack('16sH', b'penum%d', IFF_TUN | IFF_NO_PI)
20 ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
21
22 # Get the interface name
23 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
24 print("Interface Name: {}".format(ifname))
25 os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))
26 os.system("ip link set dev {} up".format(ifname))
27
28 # Set up routing
29 os.system("ip route add 192.168.50.0/24 dev {}".format(ifname))
30
31 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
32 sock.bind((IP_A, PORT))
33
34 fds = [sock,tun]
35
36 while True:
37         #this will block until at least one socket is ready
```

Run the Task5_tun_client_select.py program on VPN Client container. The packets from 192.168.50.5-> 192.168.60.5 are going through the VPN tunnel

```
root@67fefabb36ee:/volumes# python3 Task5_tun_client_select.py
Interface Name: penum0
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.50.5
From tun ==>: 192.168.50.5 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.50.5
```

Run the Task8_tun_server_select.py program on VPN Server container. The original packet 192.168.60.5->192.168.50.5 is carried inside the packet 10.9.0.12->0.0.0.0

```
root@5c81c32e1be6:/volumes# python3 Task8_tun_server_select.py
Interface Name: penum0
sock...
10.9.0.12:47826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.50.5 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.50.5
sock...
10.9.0.12:47826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.50.5 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.50.5
sock...
10.9.0.12:47826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.50.5 --> 192.168.60.5
tun...
Return: 192.168.60.5 --> 192.168.50.5
sock...
10.9.0.12:47826 --> 0.0.0.0:9090
Inside Tunnel: 192.168.50.5 --> 192.168.60.5
```

Ping host V 192.168.60.5 from host U container 192.168.50.5. The reply packets are received from host V 192.168.60.5.

```
root@ec1c6f1da079:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=62 time=11.8 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=62 time=7.60 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=62 time=13.4 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=62 time=10.5 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=62 time=9.19 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=62 time=5.13 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=62 time=13.4 ms
```

On host V container 192.168.60.5, ran tcpdump and the ICMP echo request packets from host U to host V and the ICMP reply packets from host V to host U are captured.

```
root@95b3064d908a:/# tcpdump -n -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:39:50.999488 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 33, seq 1, length 64
22:39:50.999574 IP 192.168.60.5 > 192.168.50.5: ICMP echo reply, id 33, seq 1, length 64
22:39:51.999029 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 33, seq 2, length 64
22:39:51.999205 IP 192.168.60.5 > 192.168.50.5: ICMP echo reply, id 33, seq 2, length 64
22:39:53.008823 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 33, seq 3, length 64
22:39:53.009003 IP 192.168.60.5 > 192.168.50.5: ICMP echo reply, id 33, seq 3, length 64
22:39:54.008022 IP 192.168.50.5 > 192.168.60.5: ICMP echo request, id 33, seq 4, length 64
```

Task 9: Experiment with the TAP Interface

Setting up the first lab setup environment

```
[11/23/21]seed@VM:~/.../Labsetup$ dcbuild
VPN_Client uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Router uses an image, skipping
[11/23/21]seed@VM:~/.../Labsetup$ dcup -d
Creating network "net-10.9.0.0" with the default d
Creating network "net-192.168.60.0" with the defau
WARNING: Found orphan containers (host-192.168.50.
ice in your compose file, you can run this command
Creating server-router     ... done
Creating client-10.9.0.5   ... done
Creating host-192.168.60.6 ... done
Creating host-192.168.60.5 ... done
```

Python program Task9_tun_client_select.py to configure a TAP interface using IFF_TAP. The code in the while loop reads the from the TAP interface and casts the data to Ether object and prints its fields.

Task9_tun_client_select.py

```
 7 from scapy.all import *
 8
 9 SERVER_IP = "10.9.0.11"
10 SERVER_PORT = 9090
11
12 TUNSETIFF = 0x400454ca
13 IFF_TUN   = 0x0001
14 IFF_TAP   = 0x0002
15 IFF_NO_PI = 0x1000
16
17 # Create the tun interface
18 tap = os.open("/dev/net/tun", os.O_RDWR)
19 ifr = struct.pack('16sH', b'penum%d', IFF_TAP | IFF_NO_PI)
20 ifname_bytes  = fcntl.ioctl(tap, TUNSETIFF, ifr)
21
22 # Get the interface name
23 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
24 print("Interface Name: {}".format(ifname))
25 os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
26 os.system("ip link set dev {} up".format(ifname))
27
28 # Set up routing
29 os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
30
31 # Create UDP socket
32 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
33
34 while True:
35         packet = os.read(tap, 2048)
36         if packet:
37                 ether = Ether(packet)
38                 print(ether.summary())
```

On host U (VPN Client) container, run the Task9_tun_client_select.py program

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle hostU
[11/23/21]seed@VM:~/.../Labsetup$ docksh f5
root@f5b2e942e98b:/# cd volumes/
root@f5b2e942e98b:/volumes# python3 Task9_tun_client_select.py
Interface Name: penum0
```

In other shell of host U (VPN Client) container, ping IP address 192.168.53.1 in 192.168.53.0/24 network. We do not see a response from 192.168.53.1 and shows as Destination Host Unreachable as this is a non-existing IP.

```
[11/23/21]seed@VM:~/.../Labsetup$ settitle client
[11/23/21]seed@VM:~/.../Labsetup$ docksh f5
root@f5b2e942e98b:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
From 192.168.53.99 icmp_seq=1 Destination Host Unreachable
From 192.168.53.99 icmp_seq=2 Destination Host Unreachable
From 192.168.53.99 icmp_seq=3 Destination Host Unreachable
From 192.168.53.99 icmp_seq=4 Destination Host Unreachable
From 192.168.53.99 icmp_seq=5 Destination Host Unreachable
From 192.168.53.99 icmp_seq=6 Destination Host Unreachable
```

In the VPN Client window, where the client program is run, the Ether packets are captured in the Tap interface penum0. From the below screenshot only the request packets from source 192.168.53.99 are sent to 192.168.53.1 and the reply packets are not sent.

```
root@f5b2e942e98b:/volumes# python3 Task9_tun_client_select.py
Interface Name: penum0
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
Ether / ARP who has 192.168.53.1 says 192.168.53.99
```

Program Task9_tun_client_arpreq_check.py to spoof the ARP reply and write to the TAP interface.

Open — Task9_tun_client_arpreq_check.py — ~/Lab/VPN/Labsetup/volumes

```
# Create the tun interface
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'penum%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))

# Set up routing
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

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))
```

On the VPN Client container, run Task9_tun_client_arpreq_check.py program.

hostU

```
root@f5b2e942e98b:/volumes# python3 Task9_tun_client_arpreq_check.py
Interface Name: penum0
```

arping -I tap0 192.168.53.33

Running the arping command to send out the Arp requests to the IP address 192.168.53.33 via the penum0 interface. Spoofed reply packets are sent from the Fake MAC address aa:bb:cc:dd:ee:ff and IP address 192.168.53.33

```
root@f5b2e942e98b:/# arping -I penum0 192.168.53.33
ARPING 192.168.53.33
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=0 time=23.266 msec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=1 time=9.329 msec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=2 time=14.603 msec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=3 time=8.296 msec
42 bytes from aa:bb:cc:dd:ee:ff (192.168.53.33): index=4 time=6.002 msec
```

In the host U container, where the client program is run, the fake ARP response packets are captured in the penum0 TAP interface.

```
hostU
root@f5b2e942e98b:/volumes# python3 Task9_tun_client_arpreq_check.py
Interface Name: penum0
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
--------------------------------
Ether / ARP who has 192.168.53.33 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 192.168.53.33
--------------------------------
```

arping -I penum0 1.2.3.4

1.2.3.4 is a non-existing IP, but Fake response is sent in the TAP interface from 1.2.3.4

```
root@f5b2e942e98b:/volumes# python3 Task9_tun_client_arpreq_check.py
Interface Name: penum0
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
Ether / ARP who has 1.2.3.4 says 192.168.53.99 / Padding
***** Fake response: Ether / ARP is at aa:bb:cc:dd:ee:ff says 1.2.3.4
--------------------------------
```

```
root@f5b2e942e98b:/volumes# arping -I penum0 1.2.3.4
ARPING 1.2.3.4
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=0 time=1.287 msec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=1 time=1.535 msec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=2 time=1.209 msec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=3 time=1.383 msec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=4 time=1.537 msec
42 bytes from aa:bb:cc:dd:ee:ff (1.2.3.4): index=5 time=1.471 msec
```