**ACS 545 Cryptography and Network Security**

**Lab 6: VPN Lab**

**Name:** Vijay Anand Pandian

**Lab Setup**

2f78bed08544 host-192.168.60.6

6affad7a355d client-10.9.0.5 - Host U

c97e85b7c297 server-router

2ab92a71a8c7 host-192.168.60.5 - Host V

**Task1 – Network Setup**

**Implementation and Output:**

**Testing -**

**Host U can communicate with VPN Server.**

**Text

Description automatically generated**

**VPN Server can communicate with Host V**

**Text

Description automatically generated**

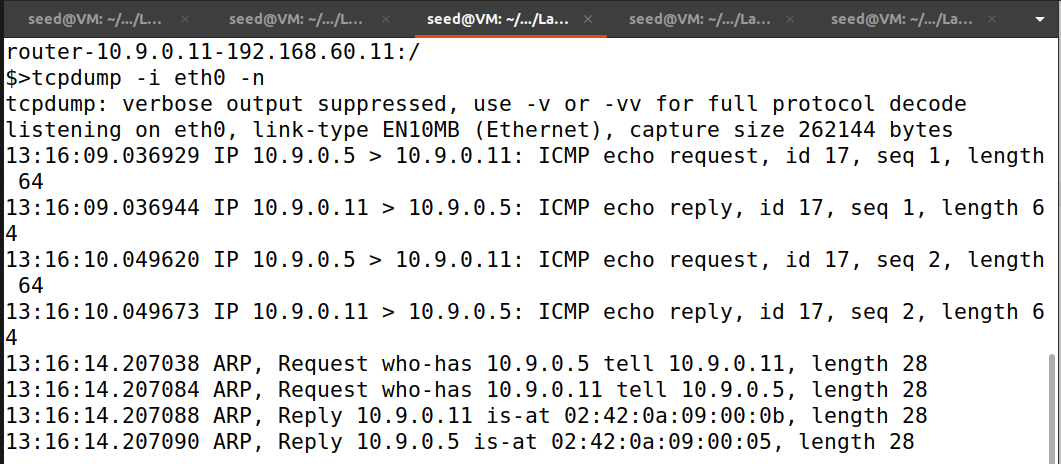
**Host U should not be able to communicate with Host V.**

**Graphical user interface, text

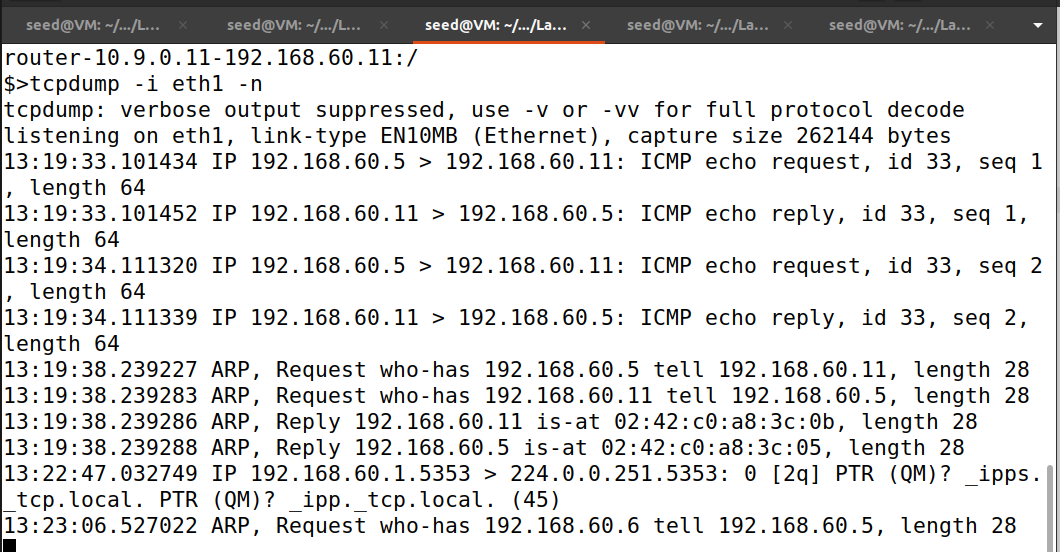
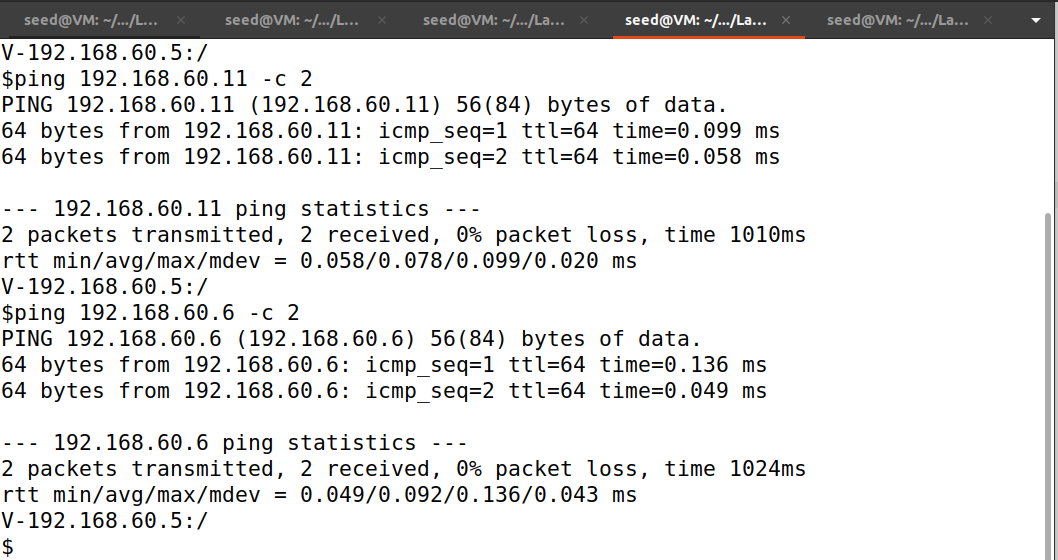
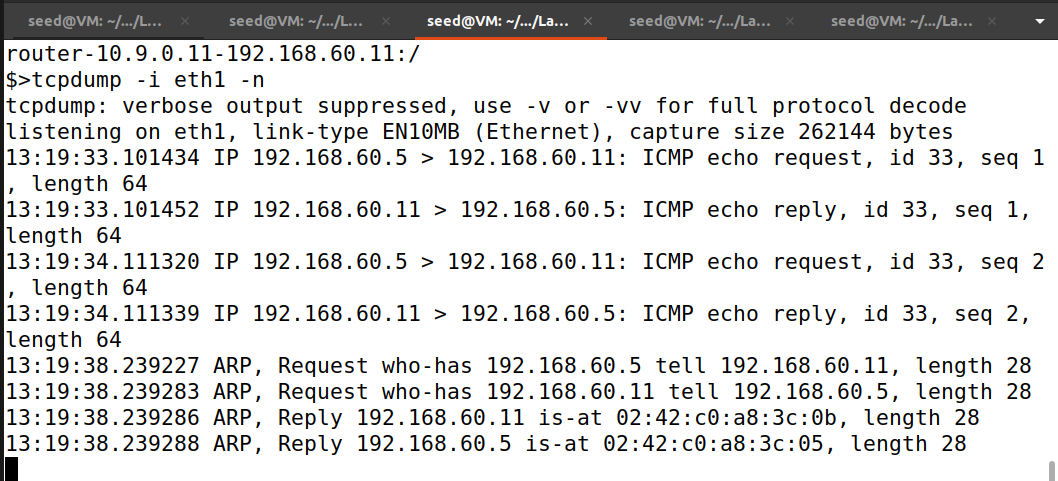
Description automatically generated**

**Run tcpdump on the router, and sniff the traffic on each of the network. Show that you can capture packets.**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Observation and Explanation:**

Lab setup was successfully done and tested all the scenarios as mentioned in the document. Above are the screenshot showing the results of the test scenario.

**Task2 - Create and Configure TUN Interface**

**Task2A – Name of the Interface**

**Code: Before renaming the tun interface**

#!/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'tun%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))

while True:

time.sleep(10)

**Code: after renaming the tun interface**

#!/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'pand%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))

while True:

time.sleep(10)

**Implementation and Output:**

**Before renaming the tun interface**

**Graphical user interface, text, email

Description automatically generated**

**After renaming the tun interface**

**Graphical user interface, text, application

Description automatically generated**

**Explanation and Observation:**

If I run the above programs we can see that it will create a tunnel interface with **tun0** before renaming and if I reran the program after renaming the tunnel interface it will show as **pand0** we can see that in the above screenshots.

**Task2B – Setup the TUN Interface**

**Code:**

#!/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'pand%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))

#Configure the tun interface

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)

**Implementation and Output:**

**Graphical user interface, text, application

Description automatically generated**

**Explanation and Observation:**

The above code will assign an IP address to the tun interface which is 192.168.53.99/24 and has been brought up, as indicated by the "UP" state in the output of the "ip addr" command which is the above screenshot.

**Task 2C - Read from the TUN Interface**

**Code:**

#!/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'pand%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))

#Configure the tun interface

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("{}:".format(ifname),ip.summary())

**Implementation and Output:**

**Graphical user interface, text

Description automatically generated**

**Explanation and Output:**

-On Host U, ping a host in the 192.168.53.0/24 network. What are printed out by the tun.py program? What has happened? Why?

After running the above code when I pinged 192.168.53.0/24 from Host U we can see that there are packets received on the tunnel interface “pand0” which are ICMP echo-requests but there are no ICMP echo-reply that is it didn’t get any acknowledgement packets.

-On Host U, ping a host in the internal network 192.168.60.0/24, Does tun.py print out anything? Why?

After running the above code and when I ping 192.168.60.5 from Host U we can see that there is nothing is printed this is because the ping packets are not sent to the tunnel interface instead they are routed through the VPN tunnel to the server and then to the internal network.

**Task 2D - Write to the TUN Interface**

**Code:**

**After getting a packet from the TUN interface, if this packet is an ICMP echo request packet, construct a corresponding echo reply packet and write it to the TUN interface. Please provide evidence to show that the code works as expected.**

#!/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'pand%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))

#Configure the tun interface

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

# Setup Routing

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

while True:

# Get a packet from the tun interface

packet = os.read(tun, 1500)

print("Received data: {}".format(packet))

if packet:

pkt = IP(packet)

print("{}:".format(ifname),pkt.summary())

# Send out a spoof packet using the tun interface

if ICMP in pkt and pkt[ICMP].type == 8:

icmp = pkt[ICMP]

newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)

newicmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)

if pkt.haslayer(Raw):

data = pkt[Raw].load

newpkt = newip/newicmp/data

else:

newpkt = newip/newicmp

os.write(tun, bytes(newpkt))

**Code:**

**Instead of writing an IP packet to the interface, write some arbitrary data to the interface, and report your observation.**

#!/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'pand%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))

#Configure the tun interface

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

# Setup Routing

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

while True:

# Get a packet from the tun interface

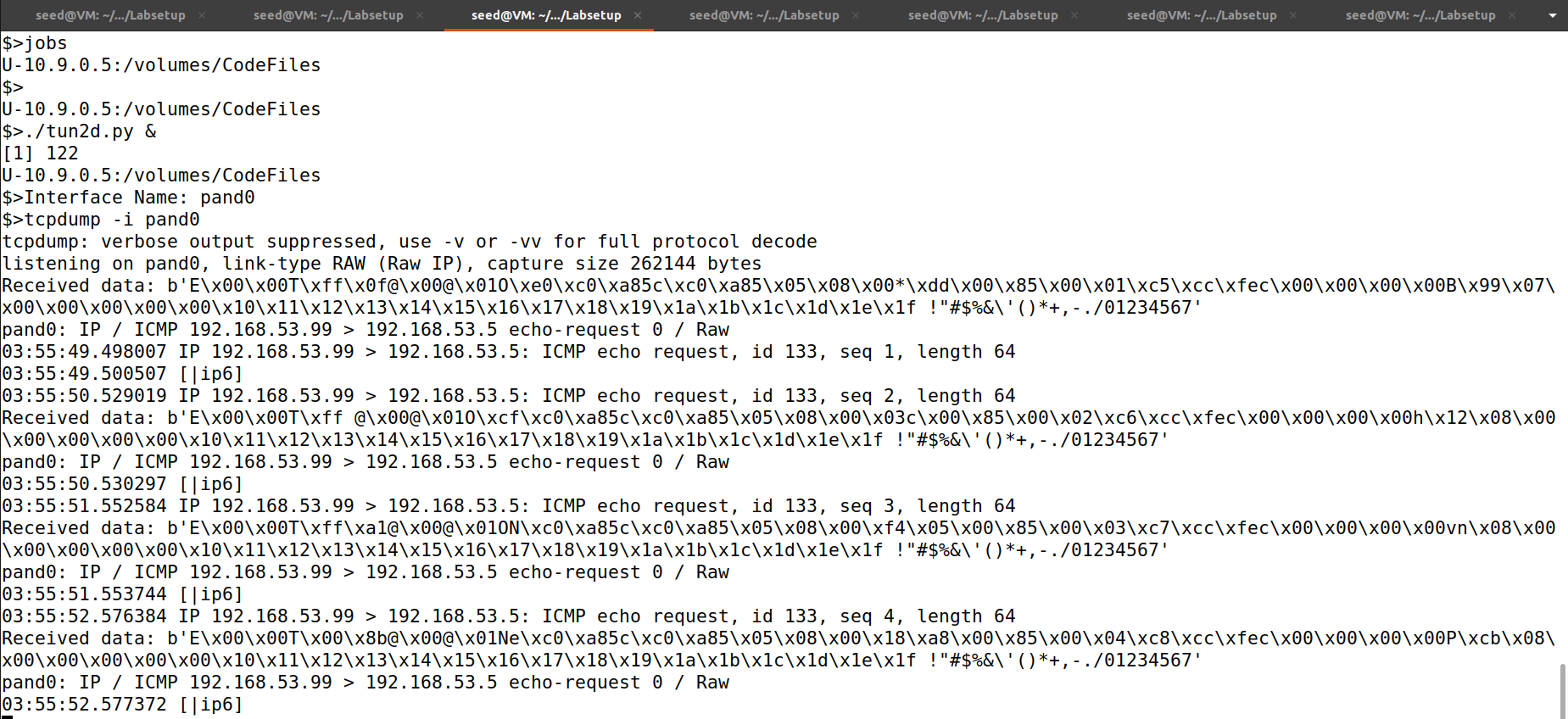
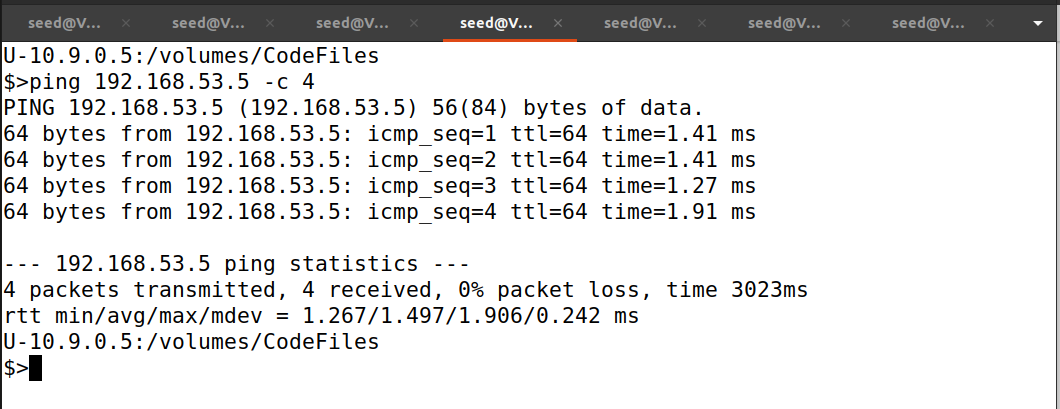
packet = os.read(tun, 1500)

print("Received data: {}".format(packet))

os.write(tun, b'arbitary data')

**Implementation and Output:**

**Graphical user interface, text

Description automatically generatedText

Description automatically generated**

**Explanation and Observation:**

1)When I ping 192.168.53.5 in Host U the above code will print the received ICMP echo-request packets and then print out the constructed ICMP echo reply packet that is sent out through the TUN interface.

2)After changing the code write arbitrary data instead of the IP packet pinged 192.168.53.5 from Host U and tcpdump in Client we can see that the ICMP echo request is being printed and the echo reply sent out through the TUN interface is not printed.

**Task3 - Send the IP Packet to VPN Server Through a Tunnel**

**Code:**

**Tun\_client.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'tun%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))

# Configure the interface

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, ("10.9.0.11", 9090))

**tun\_server.py**

#!/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))

**Implementation and Output:**

**Graphical user interface, table

Description automatically generated**

**Text

Description automatically generated with medium confidence**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text, application

Description automatically generatedText

Description automatically generated**

**Explanation and Observation:**

We will put the IP packet received from the TUN interface into the UDP pavload field of a new IP packet, and send it to another computer. In this task, we will use UDP. Namely, we put an IP packet inside the payload field of a UDP packet.

Tun Server: Listens to port 9090 and prints out whatever is received. It assumes that the data in the UDP payload. It prints out the source and destination IP address of the enclosed IP packet.

Tun Client: Sending data to another computer using UDP can be done using the standard socket programming.

We can see that connection is still not established as there is no added route at the client routing to host V through server. When we ping any IP address belonging to the 192.168.53.0/24, the packet entered to the server but does not reach the 192.168.60.0/24 network.

**Task4 - Set Up the VPN Server**

**Code:**

**Tun\_client.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'tun%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))

# Configure the interface

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, ("10.9.0.11", 9090))

**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

IP\_A = "0.0.0.0"

PORT = 9090

# Create the tun interface

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'pand%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))

#Configure the tun interface

os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

#Set up socket interface

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,data)

**Implementation and Output:**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Explanation and Observation:**

If we run the above codes and ping Host V from Host U and from the packer sniffing ‘tcpdump’ we can see that the ICMP echo-request is being printed which means packets reach Host V but the ICMP reply is not being printed because no route to go from back from Host V to Host U.

**Task5 - Handling Traffic in Both Directions**

**Code:**

**Tun\_Client.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

SERVER\_IP = "10.9.0.11"

SERVER\_PORT= 9090

# Create the tun interface

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'tun%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))

# Configure the interface

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:

#This will block until atleast one interface is ready

ready,\_,\_ = select.select([sock, tun],[],[])

for fd in ready:

if fd is sock:

print("sock ...")

data, (ip, port) = sock.recvfrom(2048)

pkt = IP(data)

print(" Inside: {} --> {}".format(pkt.src, pkt.dst))

os.write(tun,data)

if fd is tun:

print("tun...")

packet = os.read(tun, 2048)

pkt = IP(packet)

print("Return: {} -->{}".format(pkt.src, pkt.dst))

sock.sendto(packet, (SERVER\_IP, SERVER\_PORT))

**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

IP\_A = "0.0.0.0"

PORT = 9090

# Create the tun interface

tun = os.open("/dev/net/tun", os.O\_RDWR)

ifr = struct.pack('16sH', b'tun%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))

#Configure the tun interface

os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))

os.system("ip link set dev {} up".format(ifname))

#Set up socket interface

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind((IP\_A, PORT))

while True:

#This will block until atleast one interface is ready

ready,\_,\_ = select.select([sock, tun],[],[])

for fd in ready:

if fd is sock:

print("sock ...")

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,data)

if fd is tun:

print("tun...")

packet = os.read(tun, 2048)

pkt = IP(packet)

print("Return: {} -->{}".format(pkt.src, pkt.dst))

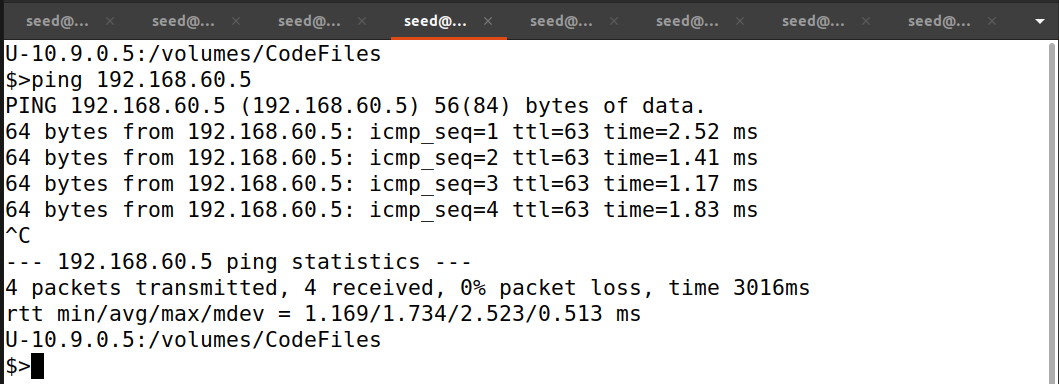
sock.sendto(packet, (ip, port))

**Implementation and Output:**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text

Description automatically generatedText

Description automatically generated**

**Explanation and Output:**

Now the tun\_client.py and tun\_program.py will now read data from two interfaces, the tun interface and the socket interface.

If we ping Host V from Host U now we can see the return packets to Host U. Here the packet flow from the Tun interface in Host U to the 10.9.0.0/24 network VPN server. Then the VPN server pass from to the server Tun interface and then it passes through 192.168.60.5. This shows that we can ping Host V from Host U.

Also if we can see that we will able to establish a telnet connection in the same way as ping with the Host V.

**Task 6 - Tunnel-Breaking Experiment**

**Implementation and Output:**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**Explanation and Observation:**

In this task, if we use the same code from the previous task. Now if we kill tun\_server or tun\_client after establishing telnet we can observe that the telnet connection gets interrupted and once I again run the tun\_client or tun\_server it got re-established since it is a container Lab. The Telnet connection got interrupted because it was based on TCP.

**Task 7 - Routing Experiment on Host V**

**Implementation and Output:**

**Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text

Description automatically generatedGraphical user interface, text, application

Description automatically generatedText

Description automatically generated**

**Explanation and Output:**

In this task we delete the default route configuration in ip table and the new route for the VPN server’s IP address is added by running the command: “ip route add 192.168.53.0/24 via 192.168.60.11”. After that ran the code from task5 and pinged Host V from Host U we can see the route.