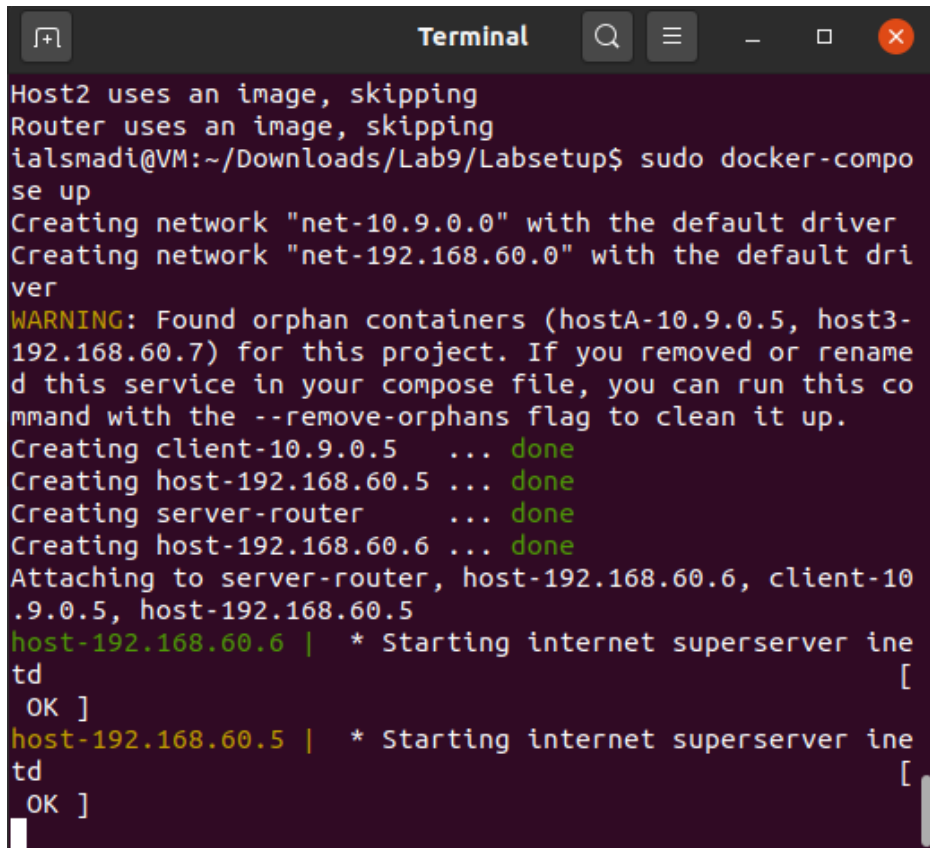


Lab9_VPN Lab: The Container Version

A Virtual Private Network (VPN) is a private network built on top of a public network, usually the Internet. Computers inside a VPN can communicate securely, just like if they were on a real private network that is physically isolated from outside, even though their traffic may go through a public network. VPN enables employees to securely access a company's intranet while traveling; it also allows companies to expand their private networks to places across the country and around the world.

The objective of this lab is to help students understand how VPN works. We focus on a specific type of VPN (the most common type), which is built on top of the transport layer. We will build a very simple VPN from the scratch, and use the process to illustrate how each piece of the VPN technology works. A real VPN program has two essential pieces, tunneling and encryption. This lab only focuses on the tunneling part, helping students understand the tunneling technology. The tunnel in this lab is not encrypted. There is another SEED lab that focuses on the encryption part.

- Start the network

A terminal window titled "Terminal" with a search icon, a menu icon, and window control buttons. The terminal output shows the execution of 'sudo docker-compose up'. It reports that Host2 and Router use existing images and are skipped. Two networks are created: 'net-10.9.0.0' and 'net-192.168.60.0'. A warning message indicates the presence of orphan containers. Four containers are created: 'client-10.9.0.5', 'host-192.168.60.5', 'server-router', and 'host-192.168.60.6', all marked as 'done'. The terminal then shows the process of attaching to the 'server-router' container and starting the 'internet superset' service in the 'td' namespace, with 'OK' responses for both the host and client containers.

```
Host2 uses an image, skipping
Router uses an image, skipping
ialsmadi@VM:~/Downloads/Lab9/Labsetup$ sudo docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
WARNING: Found orphan containers (hostA-10.9.0.5, host3-192.168.60.7) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up.
Creating client-10.9.0.5 ... done
Creating host-192.168.60.5 ... done
Creating server-router ... done
Creating host-192.168.60.6 ... done
Attaching to server-router, host-192.168.60.6, client-10.9.0.5, host-192.168.60.5
host-192.168.60.6 | * Starting internet superset in
td
OK ]
host-192.168.60.5 | * Starting internet superset in
td
OK ]
```

- Code for task

```
Terminal
GNU nano 4.8      vpn_task2.py      Modified
#!/usr/bin/python3

import fcntl
import struct
import os
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)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
```

```
GNU nano 4.8      vpn_task2.py      Modified
# 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)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

# Set up 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 True:
        pkt = IP(packet)
        print(pkt.summary())

    # Send out a spoof packet using the tun interface
    if ICMP in pkt:
```

```
GNU nano 4.8      vpn_task2.py      Modified
while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if True:
        pkt = IP(packet)
        print(pkt.summary())

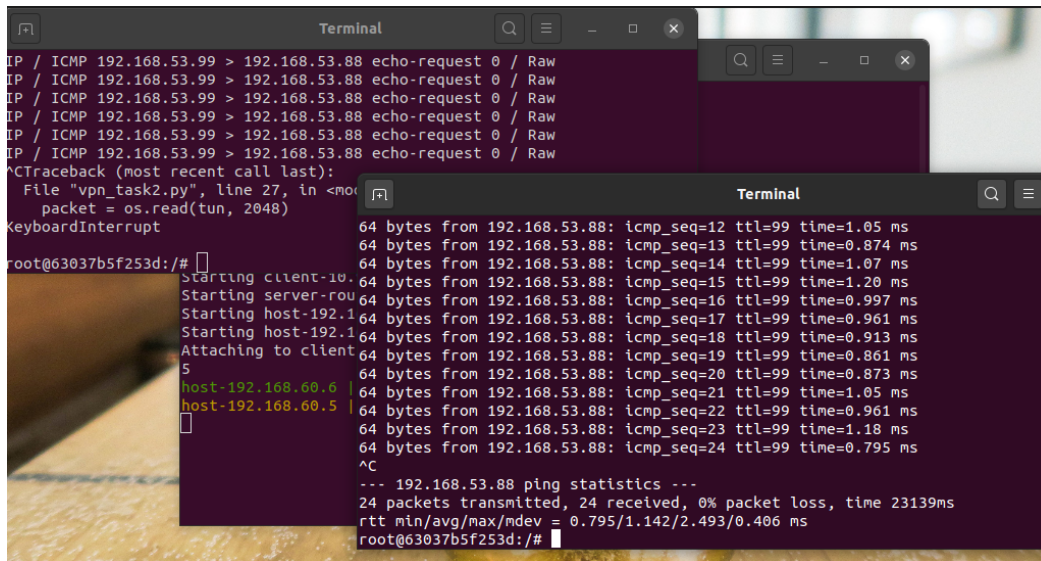
    # Send out a spoof packet using the tun interface
    if ICMP in pkt:
        newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ih>
        newip.ttl = 99
        newicmp = ICMP(type=0, id=pkt[ICMP].id, seq=p>
        if pkt.haslayer(Raw):
            data = pkt[Raw].load
            newpkt = newip/newicmp/data
        else:
            newpkt = newip/newicmp

    os.write(tun, bytes(newpkt))
```

- Run the code

```
ialsmedi@VM:~$ dig @8.8.8.8 www.example.com
Host2 us; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
Router (1 server found)
ialsmedi;; global options: +cmd
se up;; connection timed out; no servers could be reached
Creating
Creating
ialsmedi@VM:~$ cd Downloads
ver
ialsmedi@VM:~/Downloads$ sudo rm Lab9
WARNING:rm: cannot remove 'Lab9': Is a directory
192.168.ialsmedi@VM:~/Downloads$ sudo rm -r Lab9
d this sialsmedi@VM:~/Downloads$ sudo docker exec -it client-10.9
mmand wi.0.5 bash
Creatingroot@63037b5f253d:/# ls
Creatingbin etc lib32 media proc sbin tmp volumes
Creatingboot home lib64 mnt root srv usr
Creatingdev lib libx32 opt run sys var
Attachirroot@63037b5f253d:/# nano vpn_task2.py
.9.0.5,root@63037b5f253d:/# python3 v
host-19:var/ volumes/ vpn_task2.py
tdroot@63037b5f253d:/# python3 vpn_task2.py
OK ]Interface Name: tun0
host-19:
td
OK ]
```

- Then go to another VPN Client terminal, ping an host in the 192.168.53.0/24 network. Even though this host does not exist, we will get a response, because `tun_complete.py` will get the ping packet, and it will automatically send out a reply.



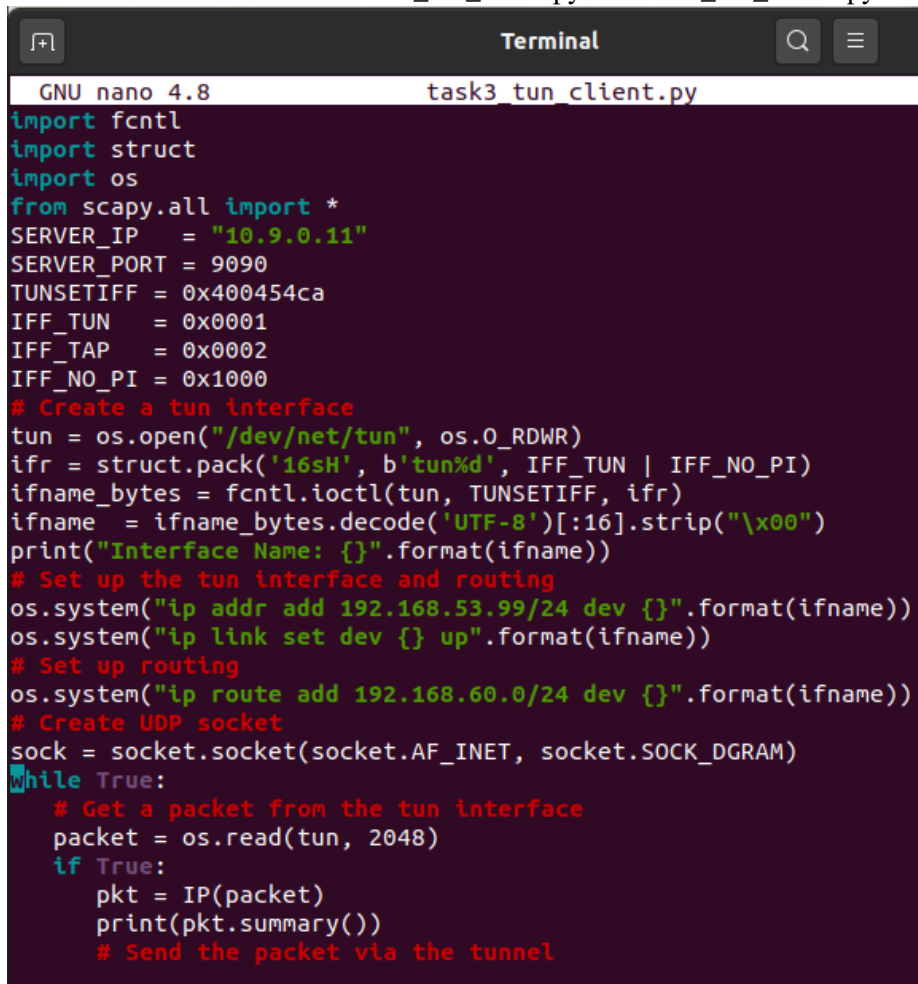
The image shows two terminal windows. The top window displays a series of ICMP echo requests from 192.168.53.99 to 192.168.53.88, all showing 0 bytes received. Below this, a traceback is shown for a file named 'vpn_task2.py' at line 27, indicating a 'KeyboardInterrupt'. The bottom window shows a series of ICMP echo requests from 192.168.53.88 to 192.168.53.99, with varying byte counts and times. At the bottom of the bottom window, a ping statistics summary is displayed: '192.168.53.88 ping statistics --- 24 packets transmitted, 24 received, 0% packet loss, time 23139ms rtt min/avg/max/mdev = 0.795/1.142/2.493/0.406 ms'.

```
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.88 echo-request 0 / Raw
^CTraceback (most recent call last):
  File "vpn_task2.py", line 27, in <module>
    packet = os.read(tun, 2048)
KeyboardInterrupt

root@63037b5f253d:/#
Starting client-10.
Starting server-rou
Starting host-192.1
Starting host-192.1
Attaching to client
5
host-192.168.60.6
host-192.168.60.5
64 bytes from 192.168.53.88: icmp_seq=12 ttl=99 time=1.05 ms
64 bytes from 192.168.53.88: icmp_seq=13 ttl=99 time=0.874 ms
64 bytes from 192.168.53.88: icmp_seq=14 ttl=99 time=1.07 ms
64 bytes from 192.168.53.88: icmp_seq=15 ttl=99 time=1.20 ms
64 bytes from 192.168.53.88: icmp_seq=16 ttl=99 time=0.997 ms
64 bytes from 192.168.53.88: icmp_seq=17 ttl=99 time=0.961 ms
64 bytes from 192.168.53.88: icmp_seq=18 ttl=99 time=0.913 ms
64 bytes from 192.168.53.88: icmp_seq=19 ttl=99 time=0.861 ms
64 bytes from 192.168.53.88: icmp_seq=20 ttl=99 time=0.873 ms
64 bytes from 192.168.53.88: icmp_seq=21 ttl=99 time=1.05 ms
64 bytes from 192.168.53.88: icmp_seq=22 ttl=99 time=0.961 ms
64 bytes from 192.168.53.88: icmp_seq=23 ttl=99 time=1.18 ms
64 bytes from 192.168.53.88: icmp_seq=24 ttl=99 time=0.795 ms
^C
--- 192.168.53.88 ping statistics ---
24 packets transmitted, 24 received, 0% packet loss, time 23139ms
rtt min/avg/max/mdev = 0.795/1.142/2.493/0.406 ms
root@63037b5f253d:/#
```

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

- This task includes two codes task3_tun_client.py and task3_tun_server.py



The image shows a terminal window with the GNU nano 4.8 editor open to the file 'task3_tun_client.py'. The code defines constants for SERVER_IP, SERVER_PORT, TUNSETIFF, and various IFF flags. It then creates a tun interface, sets up IP address and routing, creates a UDP socket, and enters a loop to read packets from the tun interface and send them via the socket.

```
GNU nano 4.8 task3_tun_client.py
import fcntl
import struct
import os
from scapy.all import *
SERVER_IP = "10.9.0.11"
SERVER_PORT = 9090
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create a 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)
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))
# Set up the tun interface and routing
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 True:
        pkt = IP(packet)
        print(pkt.summary())
        # Send the packet via the tunnel
```

```

16 GNU nano 4.8 task3_tun_server.py
16 !/usr/bin/python3
16 import fcntl
16 import struct
16 import os
16 from scapy.all import *
16 IP_A = "0.0.0.0"
16 PORT = 9090
16 TUNSETIFF = 0x400454ca
16 IFF_TUN = 0x0001
16 IFF_TAP = 0x0002
16 IFF_NO_PI = 0x1000
16 # Create a tun interface
16 tun = os.open("/dev/net/tun", os.O_RDWR)
16 ifr = struct.pack('16sH', b'tun%d' % IFF_TUN | IFF_NO_PI)
16 ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
16 ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
16 print("Interface Name: {}".format(ifname))
16 # Set up the tun interface
16 os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))
16 os.system("ip link set dev {} up".format(ifname))
16 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16 sock.bind((IP_A, PORT))
16 while True:
16     data, (ip, port) = sock.recvfrom(2048)
16     pkt = IP(data)
16     print("{}: {} --> {}: {}".format(ip, port, IP_A, PORT))
16     print("    Inside: {} --> {}".format(pkt.src, pkt.dst))
16     os.write(tun, data)

```

Keep client and server running from previous task

- Get a terminal on 192.168.60.5, start tcpdump.
- Check to see whether the packets from the VPN client have arrived yet.

[illegible]

- While we ping `192.168.60.5` on the VPN client, from the `tcpdump` results, we can see that the ping packets have arrived, and responses packets have been sent out.
- If we can set up the return path, the VPN client should be able to get the response. That's the purpose of the next task, Task 5.

Continue the rest of the tasks