

# VPN Tunneling Lab Report

## Task 1: Network Setup

VM	IP Address
VPN Client (Host U)	10.0.2.4
Gateway (VPN Server)	10.0.2.5 / 192.168.60.1
Host V	192.168.60.5

- Host U can communicate with VPN Server

```
[12/07/20]seed@HostU:~$ ping 10.0.2.5 -c 1
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=64 time=0.737 ms

--- 10.0.2.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

- VPN Server can communicate with Host V

```
[12/07/20]seed@vpnServer:~$ ping 192.168.60.5 -c 1
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=0.586 ms

--- 192.168.60.5 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

- Host U should not be able to communicate with Host V

```
[12/07/20]seed@HostU:~$ ping 192.168.60.5 -c 1
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.

--- 192.168.60.5 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

## Task 2: Create and Configure TUN Interface

```
#!/usr/bin/python3
# final version of tun.py

import os
import time
import fcntl
import struct
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'katherine%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

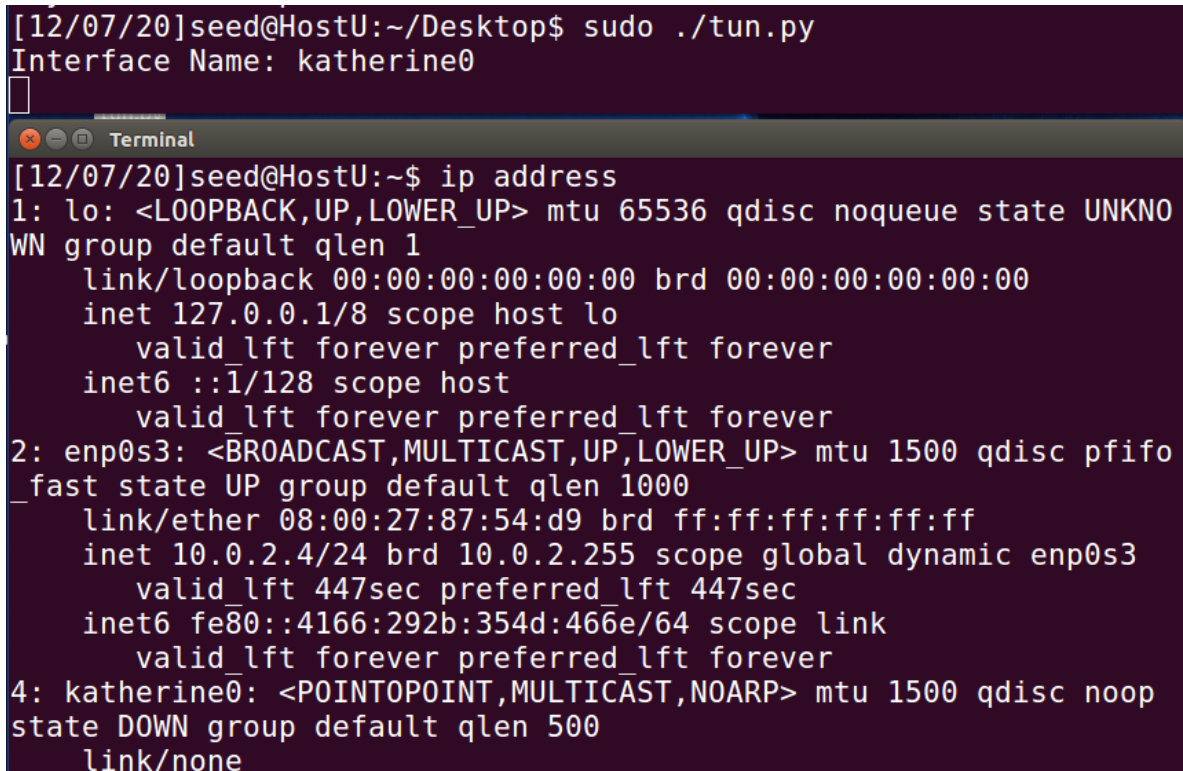
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.56.99/24 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)
    ip = IP(packet)
    #os.write(tun, b'Arbitrary') # use for writing arbitrary data
    # ICMP echo request
    if ip.proto == 1 and ip[ICMP].type == 8:
        print(ip.summary())
        newip = IP(src = ip.dst, dst = ip.src)
        newicmp = ICMP(type = "echo-reply", id = ip[ICMP].id, seq =
ip[ICMP].seq)
        newpkt = newip/newicmp/ip[Raw].load
        os.write(tun, bytes(newpkt))

```

## Task 2.a: Name of the Interface



```

[12/07/20]seed@HostU:~/Desktop$ sudo ./tun.py
Interface Name: katherine0
[12/07/20]seed@HostU:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKN
WN group default qlen 1
    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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo
_fast state UP group default qlen 1000
    link/ether 08:00:27:87:54:d9 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 447sec preferred_lft 447sec
    inet6 fe80::4166:292b:354d:466e/64 scope link
        valid_lft forever preferred_lft forever
4: katherine0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop
state DOWN group default qlen 500
    link/none

```

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

```
[12/07/20]seed@HostU:~/Desktop$ sudo ./tun.py
Interface Name: katherine0

[12/07/20]seed@HostU:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKN
WN group default qlen 1
    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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo
_fast state UP group default qlen 1000
    link/ether 08:00:27:87:54:d9 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 441sec preferred_lft 441sec
    inet6 fe80::4166:292b:354d:466e/64 scope link
        valid_lft forever preferred_lft forever
5: katherine0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500
qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.56.99/24 scope global katherine0
        valid_lft forever preferred_lft forever
    inet6 fe80::2e43:6722:cf67:db92/64 scope link flags 800
        valid_lft forever preferred_lft forever
```

在运行配置命令之后，由tun.py创建的katherine0接口被分配了一个IP地址。

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

- `$ ping 192.168.56.5`

```
IP / ICMP 192.168.56.99 > 192.168.56.5 echo-request 0 / Raw
IP / ICMP 192.168.56.99 > 192.168.56.5 echo-request 0 / Raw
IP / ICMP 192.168.56.99 > 192.168.56.5 echo-request 0 / Raw
IP / ICMP 192.168.56.99 > 192.168.56.5 echo-request 0 / Raw
IP / ICMP 192.168.56.99 > 192.168.56.5 echo-request 0 / Raw
```

向192.168.56.5发送的ICMP请求数据包被通过接口katherine0发送了，这是因为数据包的目的IP和katherine0接口的IP在同一子网中。

- `$ ping 192.168.60.5`

tun.py 没有打印出任何东西。

这是因为192.168.60.5和Host U上的三个接口的IP都不在同一子网中（enp0s3: 10.0.2.4; lo: 127.0.0.1; katherine0:192.168.56.99），也不在路由表中，所以Host U会发出一个ARP数据包询问邻居中谁的路由表中含有子网192.168.60.0/24的路由表项，如果收到了回复，就会向通向那个邻居的接口发送ICMP请求数据包。

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

- reply to a ICMP echo request packet
  - before run the tun.py

```
[12/07/20]seed@HostU:~$ ping 192.168.56.9 -c 3
PING 192.168.56.9 (192.168.56.9) 56(84) bytes of data.

--- 192.168.56.9 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2048ms
```

- after run the tun.py

```
[12/07/20]seed@HostU:~$ ping 192.168.56.9 -c 3
PING 192.168.56.9 (192.168.56.9) 56(84) bytes of data.
64 bytes from 192.168.56.9: icmp_seq=1 ttl=64 time=2.07 ms
64 bytes from 192.168.56.9: icmp_seq=2 ttl=64 time=1.62 ms
64 bytes from 192.168.56.9: icmp_seq=3 ttl=64 time=1.59 ms

--- 192.168.56.9 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
```

- write some arbitrary data to the interface

Source	Destination	Protocol	Length	Info
192.168.56.99	192.168.56.9	ICMP	84	Echo (ping) request id=0x1c95, seq=1/256, ttl=64 (no
N/A	N/A	IPv4	9	Bogus IP header length (4, must be at least 20)
192.168.56.99	192.168.56.9	ICMP	84	Echo (ping) request id=0x1c95, seq=2/512, ttl=64 (no
N/A	N/A	IPv4	9	Bogus IP header length (4, must be at least 20)
192.168.56.99	192.168.56.9	ICMP	84	Echo (ping) request id=0x1c95, seq=3/768, ttl=64 (no
N/A	N/A	IPv4	9	Bogus IP header length (4, must be at least 20)

忽略用来引发程序发包的ICMP请求数据包，向接口写入随机数据会被wireshark判定为伪造的IP数据包，但是程序本身并不会报错。

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

```
#!/usr/bin/python3
# tun_client.py

import os
import time
import fcntl
import struct
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'katherine%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 192.168.56.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
# add an entry to the routing table
os.system("sudo ip route add 192.168.60.0/24 dev {} via 192.168.56.99".format(ifname))

# Create UDP socket
```

```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
SERVER_IP = "10.0.2.5"
SERVER_PORT = 9090

while True:
    # Get a packet from the tun interface
    packet = os.read(tun, 2048)
    sock.sendto(packet, (SERVER_IP, SERVER_PORT))

```

- `$ ping 192.168.56.9`

```

[12/07/20]seed@vpnServer:~/Desktop$ sudo ./tun_server.py
10.0.2.4 : 50596 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.56.9
10.0.2.4 : 50596 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.56.9
10.0.2.4 : 50596 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.56.9
10.0.2.4 : 50596 --> 0.0.0.0 : 9090

```

VPN Server收到了从Host U的enp0s3接口 (IP: 10.0.2.4) 发来的UDP数据包, 包中payload是从Host U的katherine0接口 (IP: 192.168.56.99) 发出的发往192.168.56.9的ICMP请求报文。

- `$ ping 192.168.60.5`

```

[12/07/20]seed@vpnServer:~/Desktop$ sudo ./tun_server.py
10.0.2.4 : 46408 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.60.5
10.0.2.4 : 46408 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.60.5
10.0.2.4 : 46408 --> 0.0.0.0 : 9090
        Inside: 192.168.56.99 --> 192.168.60.5

```

因为VPN Server收到了包含Host U上的ping命令请求的UDP数据包, 并打印在终端中, 所以可以证实由Host U发往192.168.60.0/24的ping请求是经过构造的隧道被tun\_server.py收到的。

## Task 4: Set Up the VPN Server

```

#!/usr/bin/python3
# tun_server.py

import os
import time
import fcntl
import struct
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000

# CREATE A TUN INTERFACE AND CONFIGURE IT
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'katherine%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 192.168.56.66/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

# GET THE DATA FROM THE SOCKET INTERFACE
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))
    # TREAT THE RECEIVED DATA AS AN IP PACKET
    pkt = IP(data)
    print("\tInside: {} --> {}".format(pkt.src, pkt.dst))
    # WRITE THE PACKET TO THE TUN INTERFACE
    os.write(tun, bytes(pkt))

```

- Host V, enp0s3 interface, not promiscuous

icmp						
No.	Time	Source	Destination	Protocol	Length	Info
1	2020-12-07 10:08:25.4293132	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=1/256, ttl=63 (reply in 2)
2	2020-12-07 10:08:25.4293466	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=1/256, ttl=64 (request in 1)
3	2020-12-07 10:08:26.4382862	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=2/512, ttl=63 (reply in 4)
4	2020-12-07 10:08:26.4383892	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=2/512, ttl=64 (request in 3)
5	2020-12-07 10:08:27.4548781	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=3/768, ttl=63 (reply in 6)
6	2020-12-07 10:08:27.4549857	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=3/768, ttl=64 (request in 5)
7	2020-12-07 10:08:28.4797895	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=4/1024, ttl=63 (reply in 8)
8	2020-12-07 10:08:28.4798909	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=4/1024, ttl=64 (request in 7)
9	2020-12-07 10:08:29.5629562	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=5/1280, ttl=63 (reply in 10)
10	2020-12-07 10:08:29.5629988	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=5/1280, ttl=64 (request in 9)
15	2020-12-07 10:08:30.5273563	192.168.56.99	192.168.60.5	ICMP	98	Echo (ping) request id=0x0b14, seq=6/1536, ttl=63 (reply in 16)
16	2020-12-07 10:08:30.5273834	192.168.60.5	192.168.56.99	ICMP	98	Echo (ping) reply id=0x0b14, seq=6/1536, ttl=64 (request in 15)

- Host U, any interface, promiscuous

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
12	2020-12-07 10:07:54.4930983	:::1	:::1	UDP	64	50189 → 33518 Len=0
13	2020-12-07 10:08:14.5047967	:::1	:::1	UDP	64	50189 → 33518 Len=0
14	2020-12-07 10:08:26.2715653	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=1/256, ttl=64 (no response found!)
15	2020-12-07 10:08:26.2716244	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84
16	2020-12-07 10:08:27.2729671	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=2/512, ttl=64 (no response found!)
17	2020-12-07 10:08:27.2730387	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84
18	2020-12-07 10:08:28.2972739	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=3/768, ttl=64 (no response found!)
19	2020-12-07 10:08:28.2973587	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84
20	2020-12-07 10:08:29.3211380	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=4/1024, ttl=64 (no response found!)
21	2020-12-07 10:08:29.3212127	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84
22	2020-12-07 10:08:30.3456986	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=5/1280, ttl=64 (no response found!)
23	2020-12-07 10:08:30.3457371	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84
24	2020-12-07 10:08:31.3694531	PcsCompu_87:54:d9		ARP	44	Who has 10.0.2.5? Tell 10.0.2.4
25	2020-12-07 10:08:31.3699755	192.168.56.99	192.168.60.5	ICMP	100	Echo (ping) request id=0x0b14, seq=6/1536, ttl=64 (no response found!)
26	2020-12-07 10:08:31.3701438	PcsCompu_b6:3f:75		ARP	62	10.0.2.5 is at 08:00:27:b6:3f:75
27	2020-12-07 10:08:31.3702418	10.0.2.4	10.0.2.5	UDP	128	56282 → 9090 Len=84

可以看到从Host U发出的ICMP请求报文可以到达Host V（在Host V的网卡被捕获到），但是Host V响应的数据包并没有到达Host U。

## Task 5: Handling Traffic in Both Directions

```

# client version
import select

while True:
    ready, _, _ = select.select([sock, tun], [], [])
    for fd in ready:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(2048)
            pkt = IP(data)

```



```

print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
os.write(tun, bytes(pkt))
if fd is tun:
    packet = os.read(tun, 2048)
    pkt = IP(packet)
    print("From tun <==: {} --> {}".format(pkt.src, pkt.dst))
    sock.sendto(packet, (SERVER_IP, SERVER_PORT))
# server version just sendto (CLIENT_IP, CLIENT_PORT)

```

- \$ ping 192.168.60.5

```

[12/07/20]seed@HostU:~$ 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=6.72 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=4.61 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=4.79 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=6.42 ms

```

- VPN Server, any

10.0.2.4	10.0.2.5	UDP	128 8080 → 9090	Len=84	
192.168.56.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x0ebe, seq=1/256, ttl=64 (no response found!)	
192.168.56.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x0ebe, seq=1/256, ttl=63 (reply in 65)	
192.168.60.5	192.168.56.99	ICMP	100 Echo (ping) reply	id=0x0ebe, seq=1/256, ttl=64 (request in 64)	
192.168.60.5	192.168.56.99	ICMP	100 Echo (ping) reply	id=0x0ebe, seq=1/256, ttl=63	
10.0.2.5	10.0.2.4	UDP	128 9090 → 8080	Len=84	

- VPN Server, enp0s8 (192.168.60.1)

192.168.56.99	192.168.60.5	ICMP	98 Echo (ping) request	id=0x0ed6, seq=1/256,
192.168.60.5	192.168.56.99	ICMP	98 Echo (ping) reply	id=0x0ed6, seq=1/256,

- VPN Server, enp0s3

10.0.2.4	10.0.2.5	UDP	126 8080 → 9090	Len=84
10.0.2.5	10.0.2.4	UDP	126 9090 → 8080	Len=84

- VPN Server, katherine0

192.168.60.5	192.168.56.99	ICMP	84 Echo (ping) reply	id=0x0fe9, seq=258/513, ttl=63 (request in 1)
192.168.56.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0fe9, seq=259/769, ttl=64 (reply in 4)
192.168.60.5	192.168.56.99	ICMP	84 Echo (ping) reply	id=0x0fe9, seq=259/769, ttl=63 (request in 3)

- \$ telnet 192.168.60.5

```

[12/07/20]seed@HostU:~$ telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Oct 22 16:50:18 EDT 2020 from 10.0.2.4 on pts/0
/usr/lib/update-notifier/update-motd-fsck-at-reboot:[[:59: integer
expression expected:
0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

```

- VPN Server, any, promiscuous

10.0.2.4	10.0.2.5	UDP	96 8080 → 9090 Len=52
192.168.56.99	192.168.60.5	TCP	68 37240 → 23 [ACK] Seq=2354138272 Ack=1169132464 Win=30336 Len=0 TSval=687664 TSecr=
192.168.56.99	192.168.60.5	TCP	68 [TCP Dup ACK 232#1] 37240 → 23 [ACK] Seq=2354138272 Ack=1169132464 Win=30336 Len=
192.168.60.5	192.168.56.99	TELNET	92 Telnet Data ...
192.168.60.5	192.168.56.99	TCP	92 [TCP Retransmission] 23 → 37240 [PSH, ACK] Seq=1169132464 Ack=2354138272 Win=290
10.0.2.5	10.0.2.4	UDP	120 9090 → 8080 Len=76

- VPN Server, enp0s8 (192.168.60.1), not promiscuous

192.168.56.99	192.168.60.5	TELNET	67 Telnet Data ...
192.168.60.5	192.168.56.99	TELNET	67 Telnet Data ...
192.168.56.99	192.168.60.5	TCP	66 37240 → 23 [ACK] Seq=2354138273 Ack=1169132489 Win=237
192.168.56.99	192.168.60.5	TELNET	67 Telnet Data ...
192.168.60.5	192.168.56.99	TELNET	67 Telnet Data ...
192.168.56.99	192.168.60.5	TCP	66 37240 → 23 [ACK] Seq=2354138274 Ack=1169132490 Win=237

- VPN Server, enp0s3, not promiscuous

10.0.2.4	10.0.2.5	UDP	102 8080 → 9090 Len=60
10.0.2.5	10.0.2.4	UDP	102 9090 → 8080 Len=60
10.0.2.4	10.0.2.5	UDP	94 8080 → 9090 Len=52
10.0.2.4	10.0.2.5	UDP	121 8080 → 9090 Len=79
10.0.2.5	10.0.2.4	UDP	94 9090 → 8080 Len=52
10.0.2.5	10.0.2.4	UDP	106 9090 → 8080 Len=64

- How packets flow?
  - tun\_client.py经过路由配置将所有到192.168.60.5子网的数据包pkt都发送到katherine0网卡，程序由katherine0读取数据包pkt之后，用UDP协议将数据包pkt作为payload发送给10.0.2.5。【该过程在VPN Server的any接口和enp0s3接口都可观察到】
  - VPN Server接收到UDP数据包之后，tun\_server.py将payload部分当做IP数据包进行解析得到数据包pkt，再将pkt写入到katherine0接口。【这一条数据包在any接口中被捕获到，显示为无响应数据包；在katherine0接口中被捕获为有响应的数据包】
  - 由于VPN Server开启了数据包转发功能，在发现pkt的目的IP是192.168.60.5的时候，会进行转发，pkt到达192.168.60.5，之后VPN Server就会接收到来自192.168.60.5的响应。【该过程体现为VPN Server的enp0s8接口捕获数据包，和any接口捕获的中间的两个ICMP数据包】
  - 由于来自192.168.60.5的响应由于是发往192.168.56.99，所以会被发往同一子网的katherine0接口【该过程在katherine0接口被捕获为有对应请求的响应，在any接口中被捕获为没有对应请求的响应】，当程序从接口获得数据包之后，会将它作为UDP包的payload发向10.0.2.4。【该过程可在VPN Server的any接口和enp0s3接口可观察到】
  - 10.0.2.4接收到UDP数据包之后，tun\_client.py会拆开这个数据包得到来自192.168.60.5的响应，并将它写入到katherine0接口，于是katherine0接口就收到了自己发出的pkt的响应数据包。

## Task 6: Tunnel-Breaking Experiment

- break the VPN tunnel

当终止tun\_server.py的运行进程之后，在Host U上的输入不会再显示在终端里面，但是TCP连接没有断开。

- reconnect the VPN tunnel

当重新运行tun\_server.py之后，之前在Host U上输入一起出现在了终端上面。

- explanation

当VPN tunnel断开之后，在Host U的第一个输入字符没有收到响应数据包，就不会再显示在终端，telnet会持续发送带有第一个字符的数据包给192.168.60.5。当VPN tunnel被重新连接上之后，telnet下一个重传的带有第一个字符的数据包就会收到响应，之后telnet就会将在此期间终端的所有输入字符（除第一个字符之外）作为一个数据包发给192.168.60.5，于是所有的输入字符会一起出现在终端上。

- VPN Server, any

- before reconnection

10.0.2.4	10.0.2.5	UDP	97 8080 → 9090 Len=53
10.0.2.5	10.0.2.4	ICMP	125 Destination unreachable (Port unreachable)
10.0.2.4	10.0.2.5	UDP	97 8080 → 9090 Len=53
10.0.2.5	10.0.2.4	ICMP	125 Destination unreachable (Port unreachable)
10.0.2.4	10.0.2.5	UDP	97 8080 → 9090 Len=53
10.0.2.5	10.0.2.4	ICMP	125 Destination unreachable (Port unreachable)
10.0.2.4	10.0.2.5	UDP	97 8080 → 9090 Len=53
10.0.2.5	10.0.2.4	ICMP	125 Destination unreachable (Port unreachable)
:::1	:::1	UDP	64 51110 → 33144 Len=0

- after reconnection



285	2020-12-07 11:34:26.5518595...	127.0.0.1	127.0.0.1	TCP	76 34170 → 5037 [SYN] Seq...
286	2020-12-07 11:34:26.5518639...	127.0.0.1	127.0.0.1	TCP	56 5037 → 34170 [RST, ACK]
287	2020-12-07 11:34:27.3749795...	PcsCompu_d1:b1:09		ARP	44 Who has 1.0.0.1? Tell...
288	2020-12-07 11:34:27.5019894...	PcsCompu_d1:b1:09		ARP	44 Who has 1.1.1.1? Tell...
289	2020-12-07 11:34:28.3985439...	PcsCompu_d1:b1:09		ARP	44 Who has 1.0.0.1? Tell...
290	2020-12-07 11:34:28.4975157...	10.0.2.4	10.0.2.5	UDP	97 8080 → 9090 Len=53
291	2020-12-07 11:34:28.4985716...	192.168.56.99	192.168.60.5	TELNET	69 Telnet Data ...
292	2020-12-07 11:34:28.4985861...	192.168.56.99	192.168.60.5	TCP	69 [TCP Keep-Alive] 37244
293	2020-12-07 11:34:28.4992148...	192.168.60.5	192.168.56.99	TELNET	69 Telnet Data ...
294	2020-12-07 11:34:28.4992269...	192.168.60.5	192.168.56.99	TCP	69 [TCP Keep-Alive] 23 →
295	2020-12-07 11:34:28.5001004...	10.0.2.5	10.0.2.4	UDP	97 9090 → 8080 Len=53
296	2020-12-07 11:34:28.5023814...	10.0.2.4	10.0.2.5	UDP	99 8080 → 9090 Len=55
297	2020-12-07 11:34:28.5031343...	192.168.56.99	192.168.60.5	TELNET	71 Telnet Data ...
298	2020-12-07 11:34:28.5031537...	192.168.56.99	192.168.60.5	TCP	71 [TCP Retransmission] 3

Frame 297: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface 0  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.56.99, Dst: 192.168.60.5  
Transmission Control Protocol, Src Port: 37244, Dst Port: 23, Seq: 945023378, Ack: 172968141, Len: 3  
Telnet  
Data: sls

## Task 7: Routing Experiment on Host V

```
$ sudo ip route del 0.0.0.0/0
$ sudo ip route add default dev enp0s3 via 192.168.60.1
```

## Task 8: Experiment with the TUN IP Address

- Where are the packets dropped?
  - VPN Server, any

10.0.2.4	10.0.2.5	UDP	128 8080 → 9090 Len=84	
192.168.30.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x0a26, seq=2/512, ttl=64 (no response found!)
10.0.2.4	10.0.2.5	UDP	128 8080 → 9090 Len=84	
192.168.30.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x0a26, seq=3/768, ttl=64 (no response found!)
10.0.2.4	10.0.2.5	UDP	128 8080 → 9090 Len=84	
192.168.30.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x0a26, seq=4/1024, ttl=64 (no response found!)

- VPN Server, katherine0

192.168.30.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0a34, seq=60/15360, ttl=64 (no response found!)
192.168.30.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0a34, seq=61/15616, ttl=64 (no response found!)
192.168.30.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0a34, seq=62/15872, ttl=64 (no response found!)
192.168.30.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0a34, seq=63/16128, ttl=64 (no response found!)
192.168.30.99	192.168.60.5	ICMP	84 Echo (ping) request	id=0x0a34, seq=64/16384, ttl=64 (no response found!)

- capture nothing from enp0s8

VPN Server没有转发ICMP请求数据包给192.168.60.5，数据包在VPN Server的katherine0接口被收到之后丢弃。

- Why are the packets dropped?

反向路径过滤：根据包的源地址查找路由的出接口，然后比较包的原始入接口是否和查到的出接口一致，如果不一致的话根据反向路径过滤规则就会将这个包丢弃。

ICMP请求数据包是从katherine0接口被VPN Server收到的，但是其目的IP是与enp0s8接口的IP地址在一个子网中，需要从enp0s8接口发出，违背了反向路径过滤规则所以被丢弃了。

- How to solve this problem?

关闭VPN Server上katherine0接口的反向过滤检查

```
$ sudo sysctl -w net.ipv4.conf.katherine0.rp_filter=0
```

10.0.2.4	10.0.2.5	UDP	128 8080 → 9090 Len=84	
192.168.30.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x097d, seq=589/19714, ttl=64 (no response found!)
192.168.30.99	192.168.60.5	ICMP	100 Echo (ping) request	id=0x097d, seq=589/19714, ttl=63 (reply in 1375)
192.168.60.5	192.168.30.99	ICMP	100 Echo (ping) reply	id=0x097d, seq=589/19714, ttl=64 (request in 1374)
PcsCompu_d1:b...		ARP	44 Who has 192.168.30.99? Tell	192.168.60.1

之前VPN Server的katherine0和Host U的katherine0接口的IP地址在同一个子网中，所以不需要配置路由表就可以相互通讯，但是现在Host U的katherine0的IP改为了192.168.30.99，所以VPN Server无法从路由表中找到该从哪个接口发送这个数据包。在VPN Server上配置路由表：

```
$ sudo ip route add 192.168.30.0/24 dev katherine0 via 192.168.56.66
```

```

10.0.2.4      10.0.2.5      UDP      128 8080 → 9090 Len=84
192.168.30.99 192.168.60.5 ICMP     100 Echo (ping) request id=0x097d, seq=673/41218, ttl=64 (no response found!)
192.168.30.99 192.168.60.5 ICMP     100 Echo (ping) request id=0x097d, seq=673/41218, ttl=63 (reply in 20)
192.168.60.5   192.168.30.99 ICMP     100 Echo (ping) reply id=0x097d, seq=673/41218, ttl=64 (request in 19)
192.168.60.5   192.168.30.99 ICMP     100 Echo (ping) reply id=0x097d, seq=673/41218, ttl=63
10.0.2.5      10.0.2.4      UDP      128 9090 → 8080 Len=84

```

```

[12/08/20]seed@HostU:~$ 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=667 ttl=63 time=4.95 ms
64 bytes from 192.168.60.5: icmp_seq=668 ttl=63 time=3.55 ms
64 bytes from 192.168.60.5: icmp_seq=669 ttl=63 time=4.35 ms
64 bytes from 192.168.60.5: icmp_seq=670 ttl=63 time=3.84 ms

```

## Task 9: Experiment with the TAP Interface

- `$ ping 192.168.56.88 -c 3`

```

-----
Ether / ARP who has 192.168.56.88 says 192.168.56.99
***** Fake response: Ether / ARP is at 00:00:00:00:00:00 says 192.168.56.88
-----
Ether / IP / ICMP 192.168.56.99 > 192.168.56.88 echo-request 0 / Raw
-----
Ether / IP / ICMP 192.168.56.99 > 192.168.56.88 echo-request 0 / Raw
-----
Ether / IP / ICMP 192.168.56.99 > 192.168.56.88 echo-request 0 / Raw

```

```

[12/08/20]seed@HostU:~$ ping 192.168.56.88 -c 3
PING 192.168.56.88 (192.168.56.88) 56(84) bytes of data.

--- 192.168.56.88 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2031ms

```

向192.168.56.0/24子网中的IP地址进行ping的时候首先会发出一个ARP数据包询问谁有192.168.56.88这个被ping的IP地址，程序构造发出了一个假的ARP响应，声明自己有这个IP，于是之后的ping请求包都会被发到katherine0接口，但是由于程序并没有处理ping请求，所以ping命令的终端没有收到响应。

- `$ arping -I katherine0 192.168.56.33`

```

[12/08/20]seed@HostU:~$ arping -I katherine0 192.168.56.33
ARPING 192.168.56.33 from 192.168.56.99 katherine0
Unicast reply from 192.168.56.33 [FF:FF:FF:FF:FF:FF] 3.028ms
Unicast reply from 192.168.56.33 [FF:FF:FF:FF:FF:FF] 4.145ms
Unicast reply from 192.168.56.33 [FF:FF:FF:FF:FF:FF] 4.489ms

```

- `$ arping -I katherine0 1.2.3.4`

```

[12/08/20]seed@HostU:~$ arping -I katherine0 1.2.3.4
ARPING 1.2.3.4 from 192.168.56.99 katherine0
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF] 2.519ms
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF] 3.929ms
Unicast reply from 1.2.3.4 [FF:FF:FF:FF:FF:FF] 3.515ms

```

```

#!/usr/bin/python3
# tap.py

```

```

import os
import time
import fcntl
import struct

```

```

import select
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'katherine%d', IFF_TAP | 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 192.168.56.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
# add an entry to the routing table
os.system("sudo ip route add 192.168.60.0/24 dev {} via 192.168.56.99".format(ifname))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0", 8080))
SERVER_IP = "10.0.2.5"
SERVER_PORT = 9090

while True:
    packet = os.read(tun, 2048)
    if True:
        print("-----")
        ether = Ether(packet)
        print(ether.summary())

        # Send a spoofed ARP response
        if ARP in ether and ether[ARP].op == 1:
            arp = ether[ARP]
            newether = Ether(src = ether.dst, dst = ether.src)
            newarp = ARP(op = 2, hwsrc = arp.hwdst, psrc = arp.pdst, hwdst =
arp.hwsrc, pdst = arp.psrc)
            newpkt = newether/newarp

            print("***** Fake response: {}".format(newpkt.summary()))
            os.write(tun, bytes(newpkt))

```