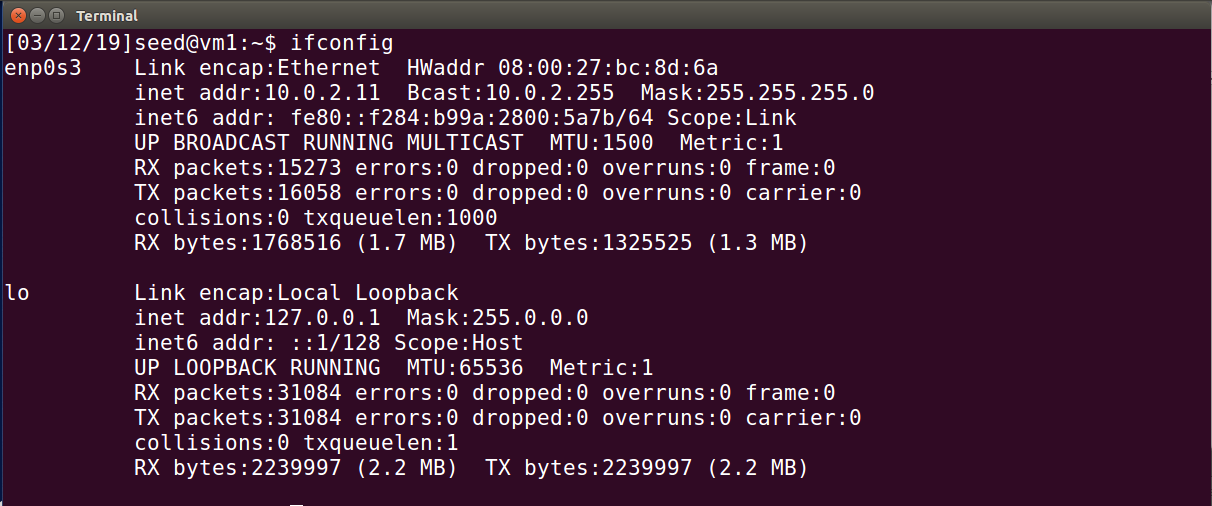
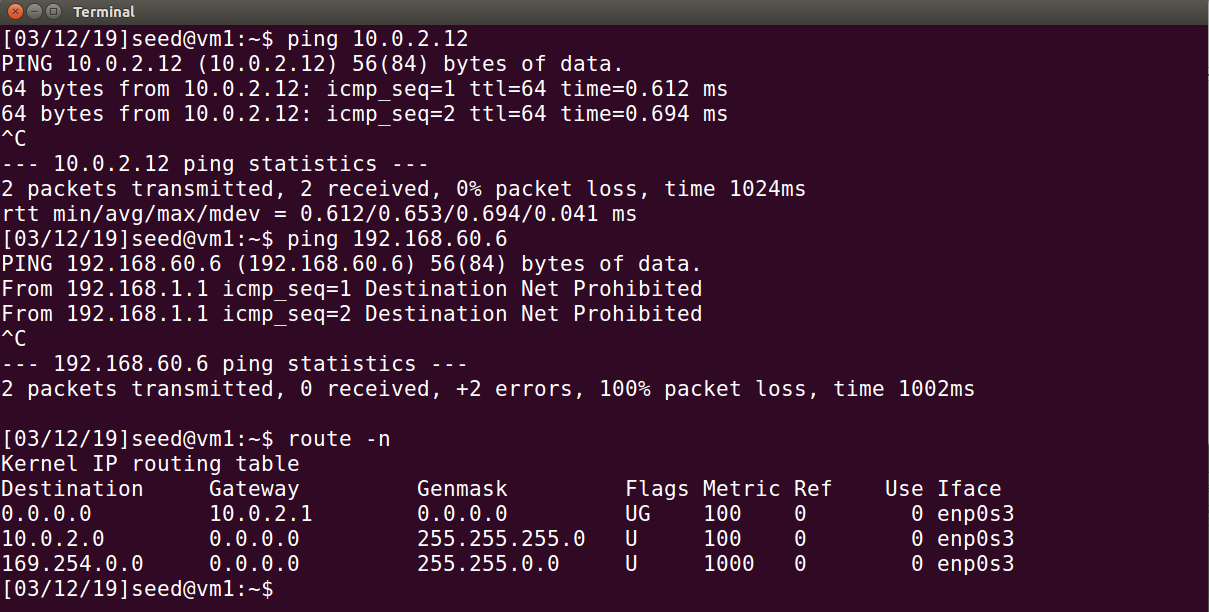
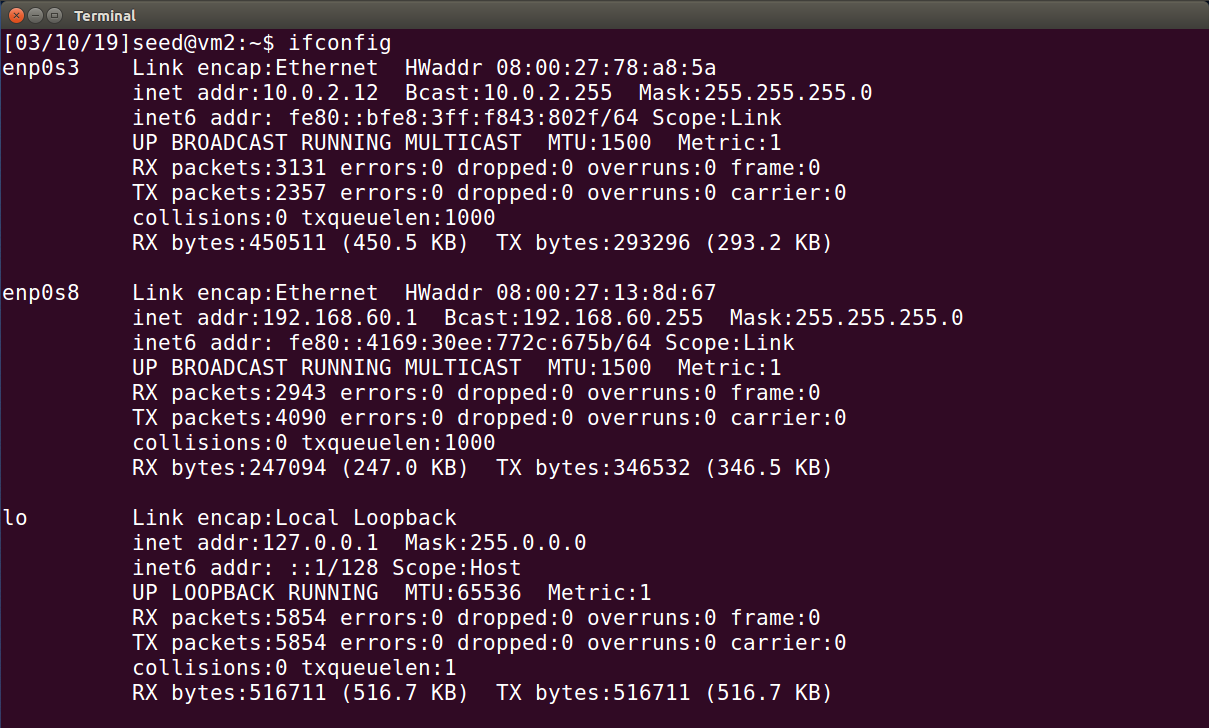
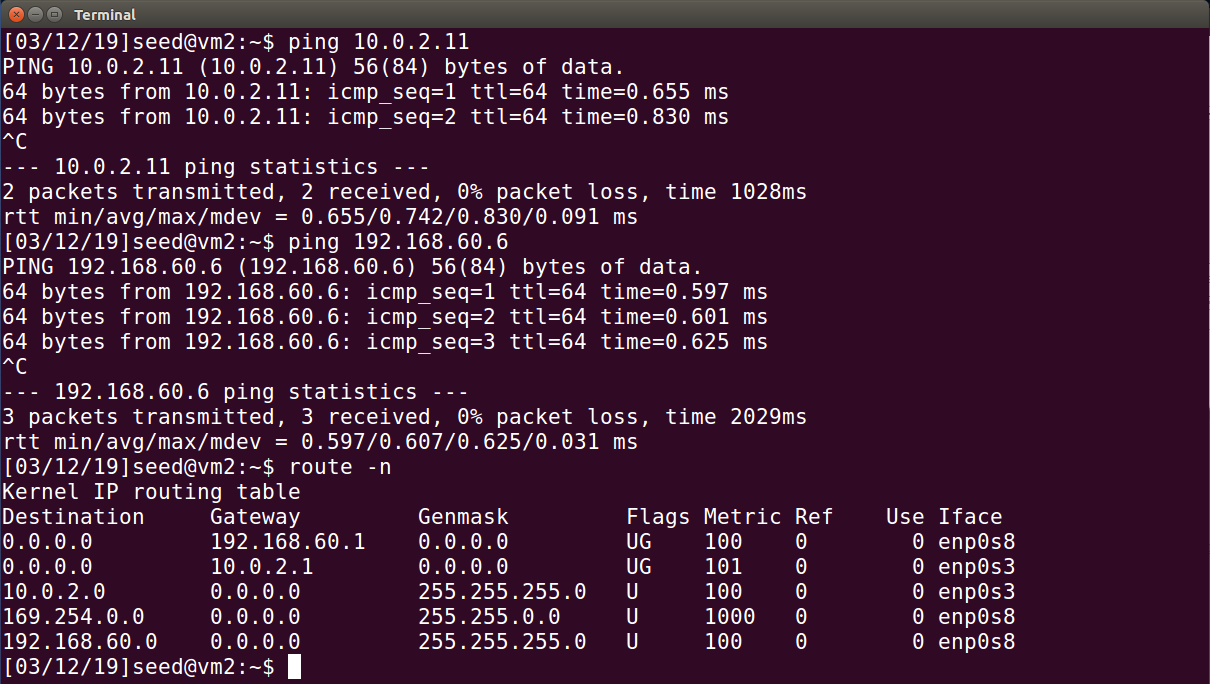
Task 1: VM setup for this lab



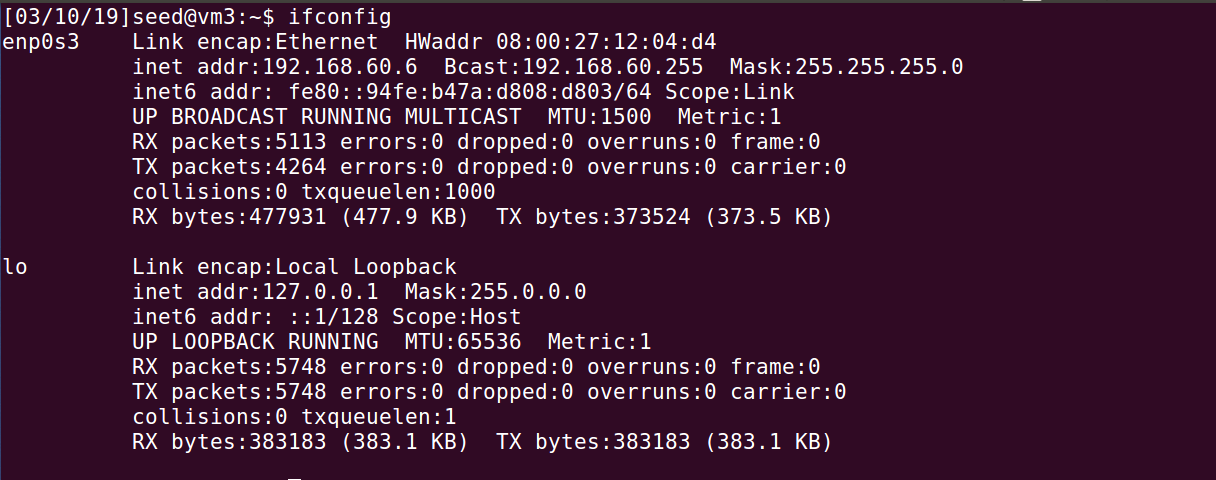


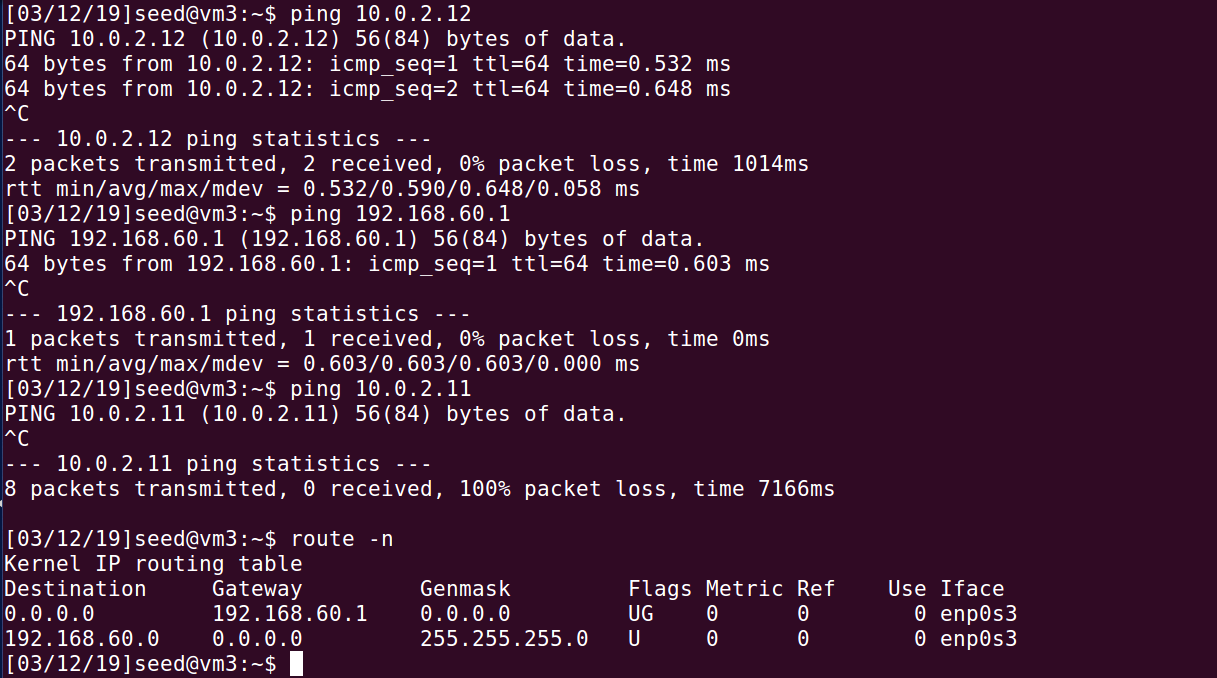
Observation: Configured Host U with a VPN Client – Nat Network adapter with IP 10.0.2.11. It is connected to the VPN server through the same LAN. It can ping the server but cannot reach Host V. The goal of this lab is to configure a VPN tunnel through a VPN server to allow Host U to talk to Host V.





Observation: Configured VPN Server – Nat Network adapter with IP 10.0.2.12 and VPN Server – Host Only adapter with IP 192.168.232.101. With the dual adapters it can talk to both Host U and Host V.

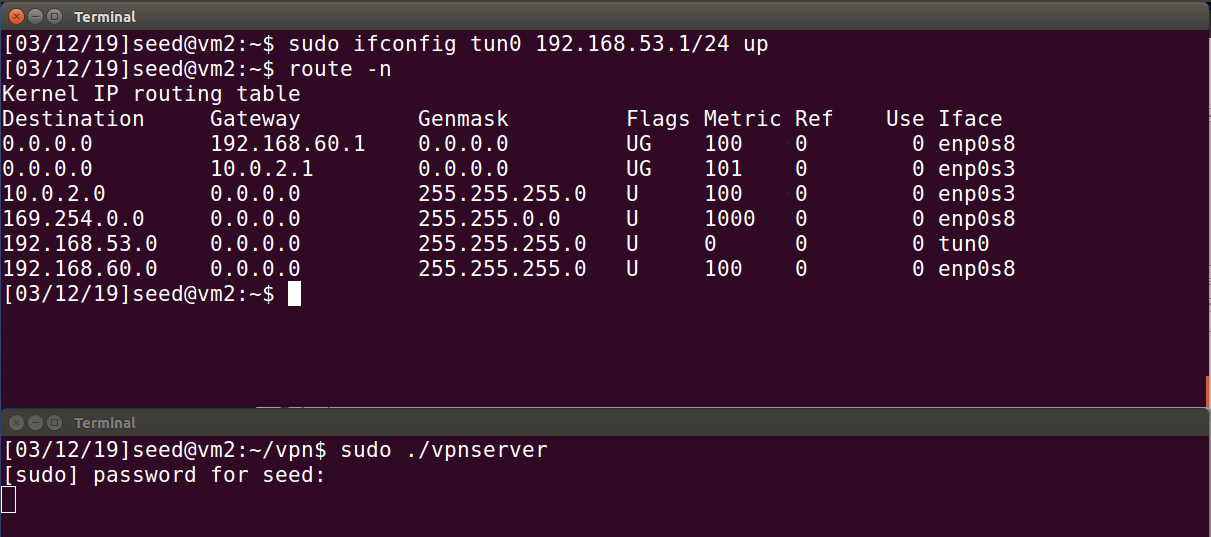




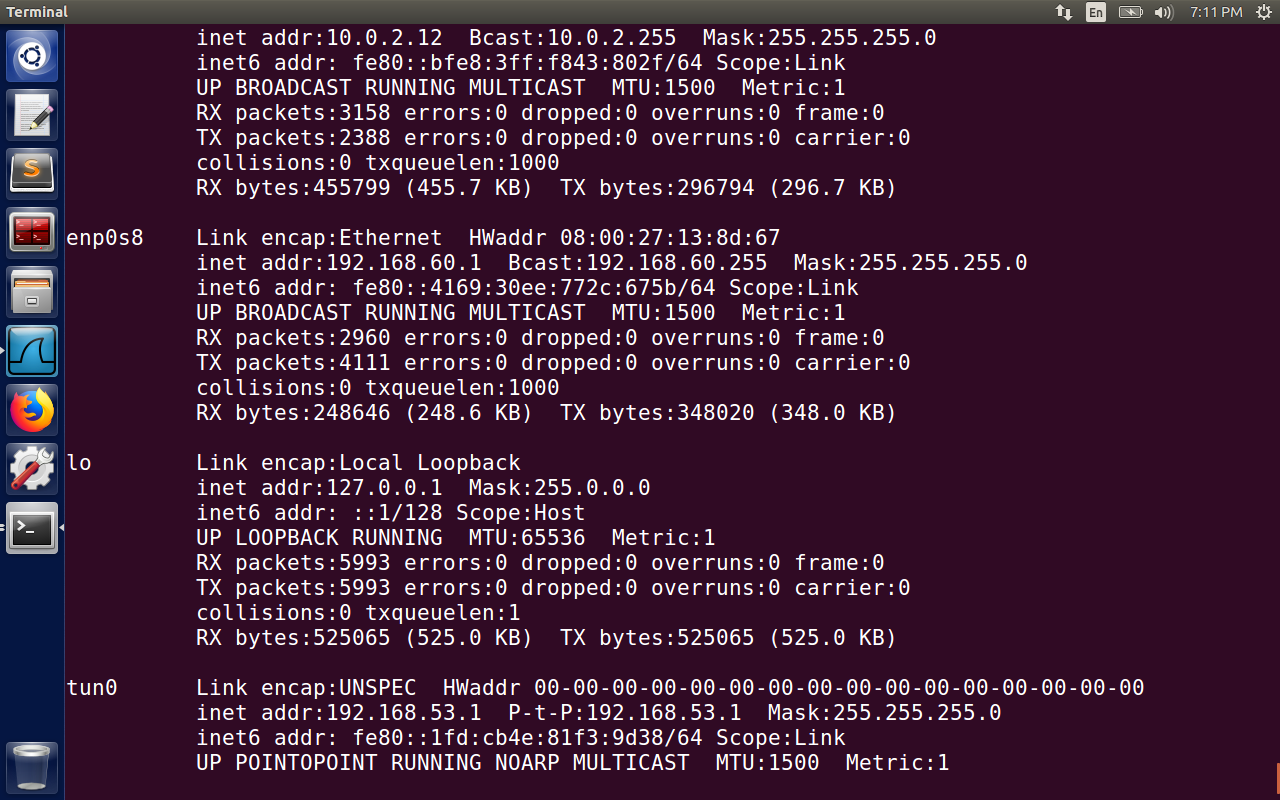
Observation: Host V is a computer inside the private network, it has a Gateway – Host Only adapter with IP 192.168.60.6.

Task 2: Creating a VPN Tunnel using TUN/TAP

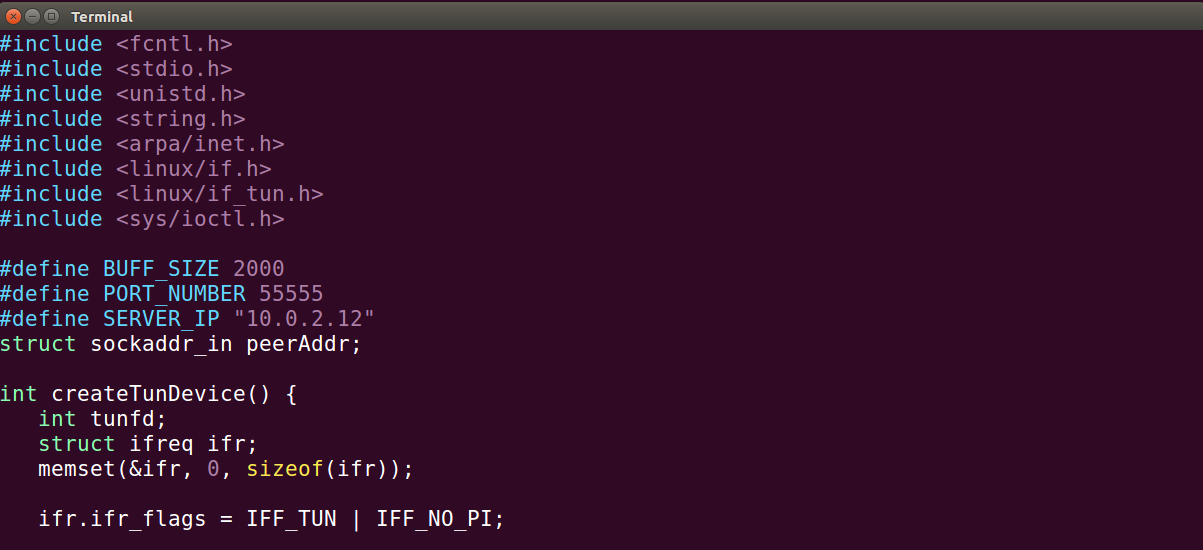
Step 1: Run VPN Server.



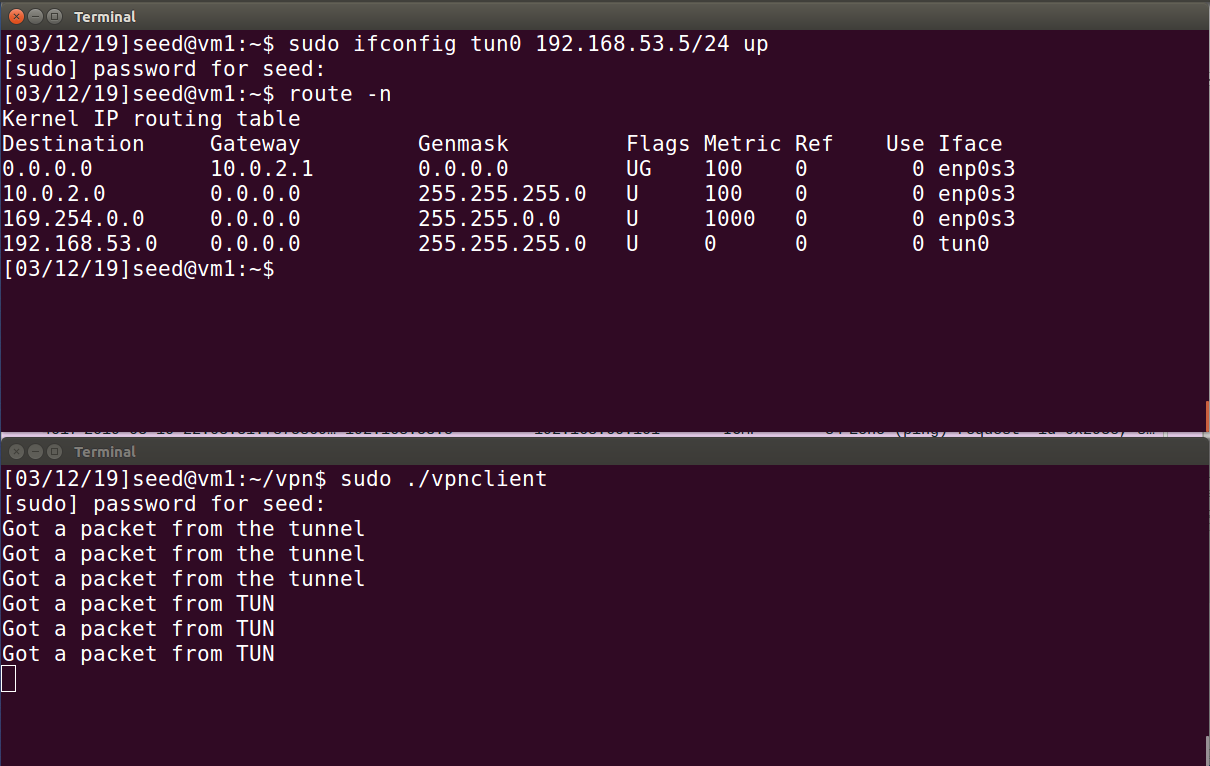
Observation: On VPN Server, I first run the server program. Configure the tun0 interface. I use 192.168.53.1/24 as IP for the TUN interface for the VPN Server. The above command assign the IP address to the tun0, bring it up and the corresponding route was added to routing table automatically.



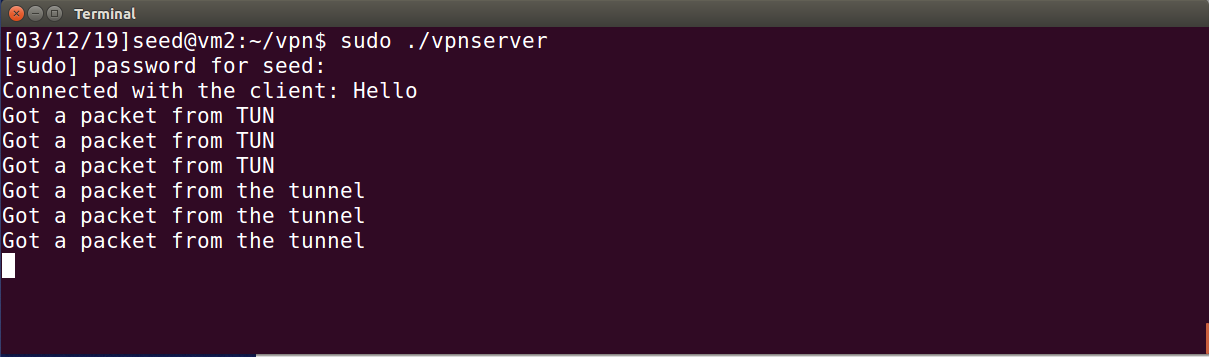
Observation: Ifconfig command above shows the IP configured to the tun0 interface, 192.168.53.1.

Step 2: Run VPN Client 

Observation: Before starting the VPN client, the IP for the server needs to be changed to our server.

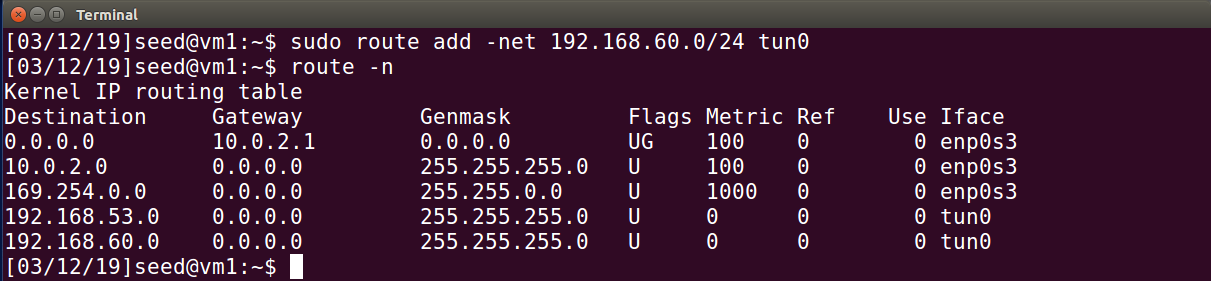


Observation: On VPN Client, we first run the client program. Configure the tun0 interface. I use 192.168.53.5/24 as IP for the TUN interface for the VPN Client. The above command assign the IP address to the tun0, bring it up and the corresponding route was added to routing table automatically. The VPN tunnel is established.

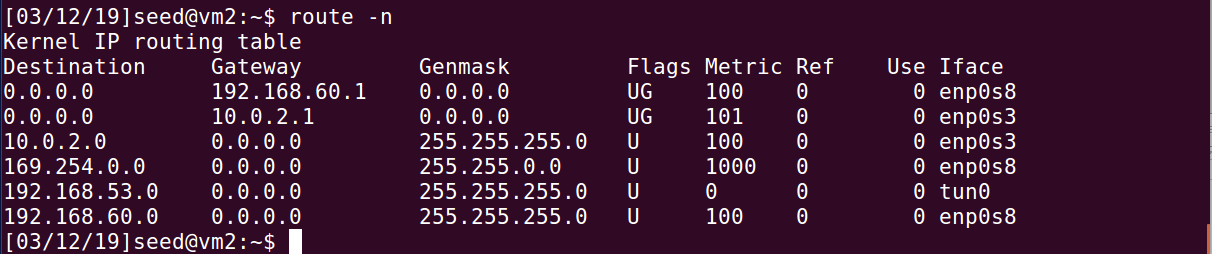


Observation: The server program creates a TUN interface (tun0) and waits for a tunnel connection request from a client. The client program creates a TUN interface (tun0) and sends a “hello” message to VPN Server to establish a VPN tunnel, as you can see above. From tun0 interface the packets will be sent through the VPN tunnel.

Step 3: Set Up Routing on Client and Server VMs

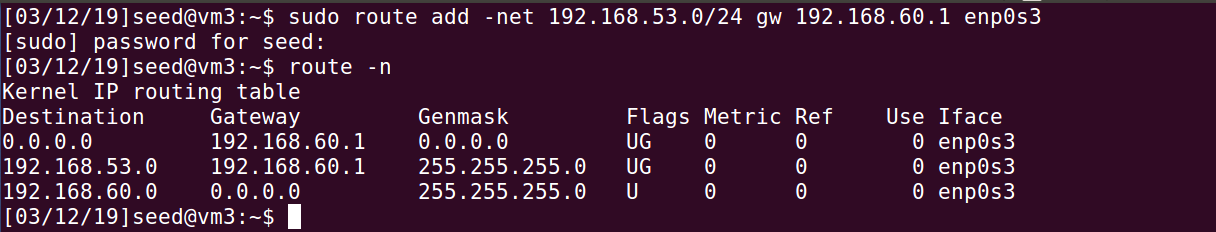


Observation: I add a route above, so that all the packets for 192.168.60.0/24 are routed to the tun0 interface. This tells the VPN Client that all the traffic to private network 192.168.60.0/24 should go through the tunnel.



Observation: On both client and server machines, we needed to set up a routing entry so all the traffic going to the 192.168.53.0/24 network are directed to the tun0 interface. This entry was added automatically when we assigned 192.169.53.X to the tun0 interface.

Step 4: Set Up Routing on Host V

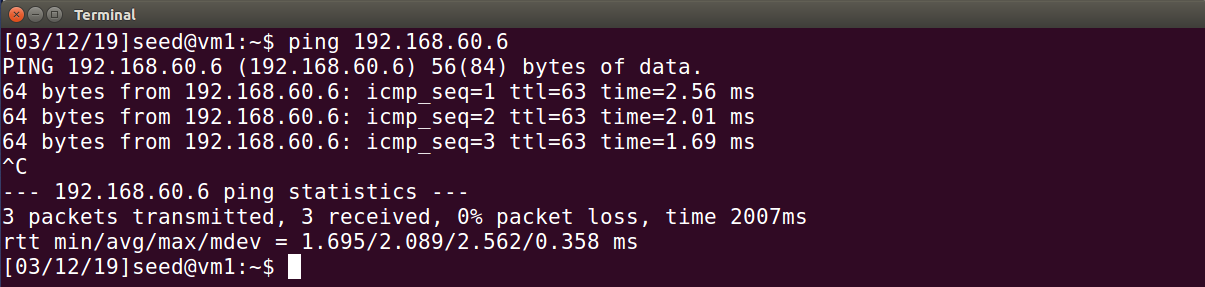


Observation: The reply packets should go back via the same VPN tunnel, so that they are protected. To ensure that, routed all packets for the 192.168.53.0/24 network toward the tunnel. For Host V, adding the above routing entry routes such packets to VPN Server. Host V connects to the 192.168.53.0/24 network via the enp0s3 network interface.

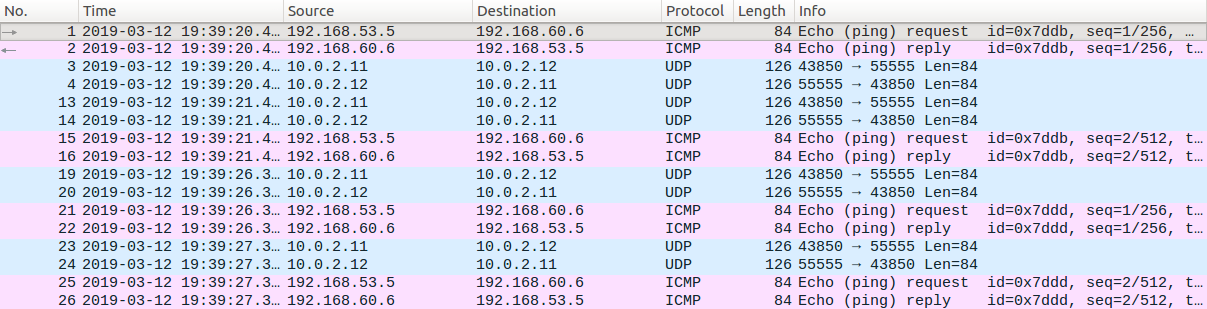
Step 5: Test the VPN Tunnel

Before the VPN is established there would be no response from Host V because it is not reachable from Host U. After the VPN is setup and starts working, Host U can access Host V.

All the network traffic between Host U and Host V will go through a tunnel between VPN Client and VPN Server.



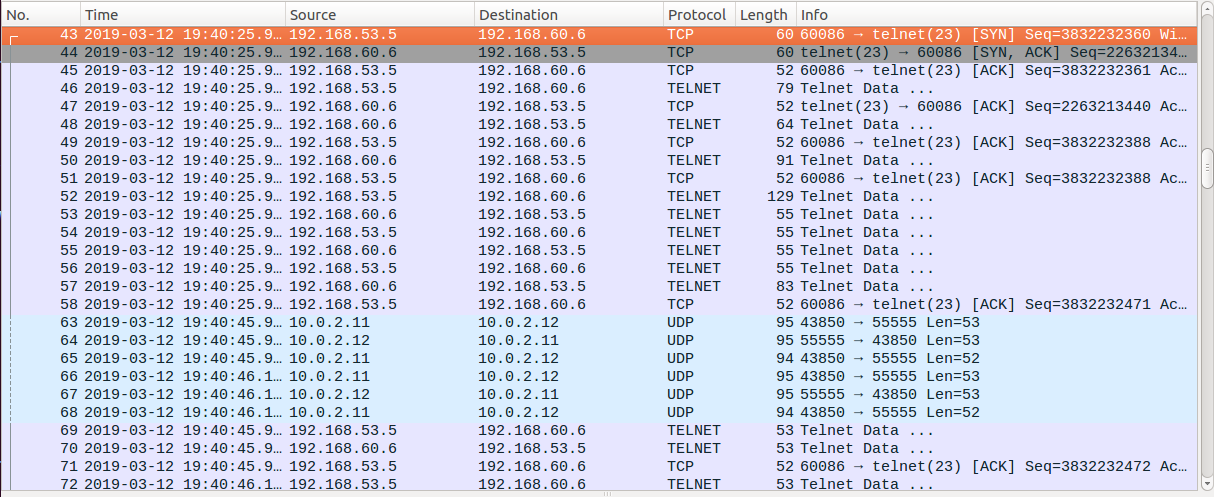
Observation: Ping Host V from Host U and we see the above results.



Observation: The above figure shows the packets generated when we ping Host V (192.168.60.6). The above packets are generated by the ping request. Due to the routing setup, the ICMP packet is routed to the TUN interface (the source IP is 192.168.53.5->assigned to tun0). The tunnel application gets the ICMP packet, and then feeds it into its tunnel, puts it inside a UDP packet towards the VPN Server (10.0.2.12). The return UDP packet from VPN Server, inside it is an encapsulated ICMP echo reply packet from 192.168.60.6. The tunnel application on VPN Client gets this UDP packet, and takes out the encapsulated ICMP packet, and gives it to kernel via the tun0 interface. Then it’s all triggered again by another ICMP echo request message.

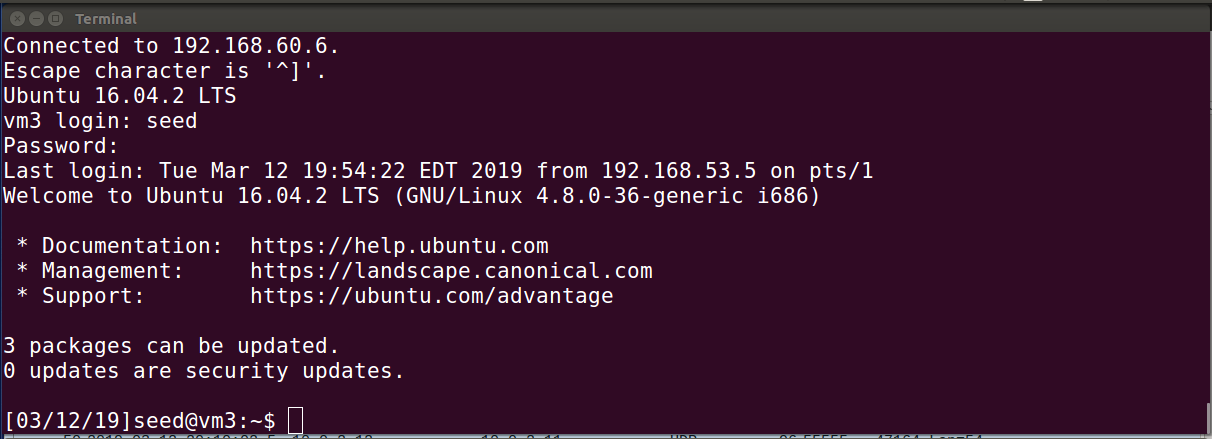


Observation: The above result shows the successful connection to the telnet server on Host V inside the private network from Host U through the VPN tunnel.

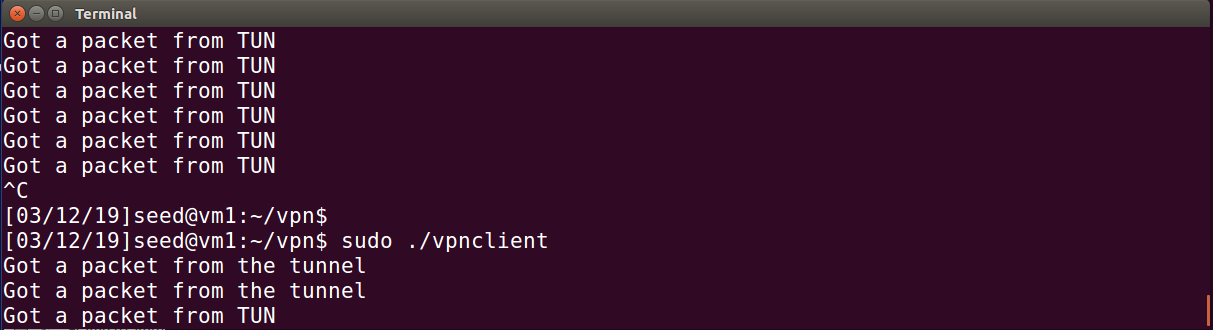


**Observation:** The above is the wire shark packet capture from successful connection to the telnet server on Host V inside the private network from Host U through the VPN tunnel.

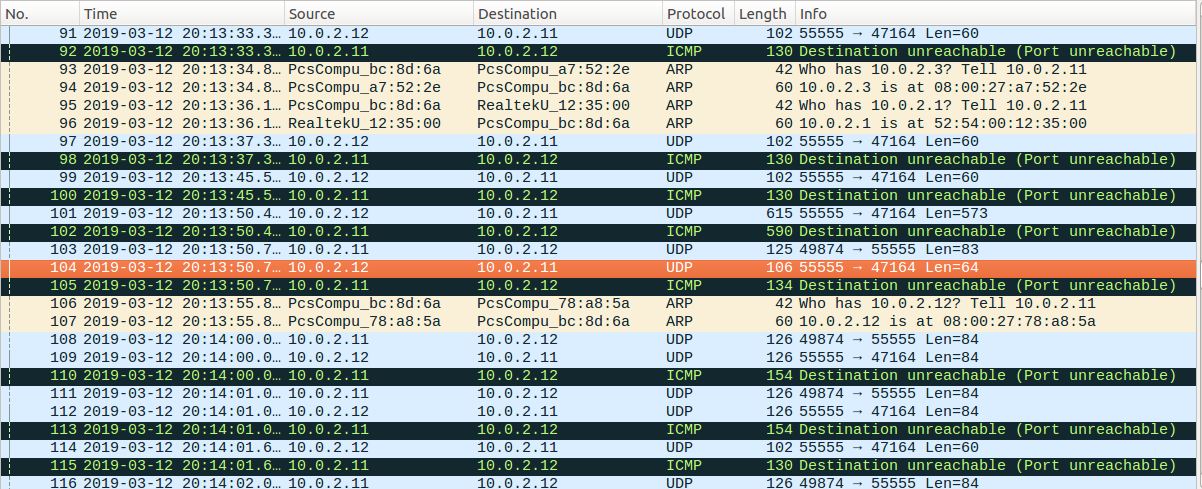
Step 6: Tunnel-Breaking Test.



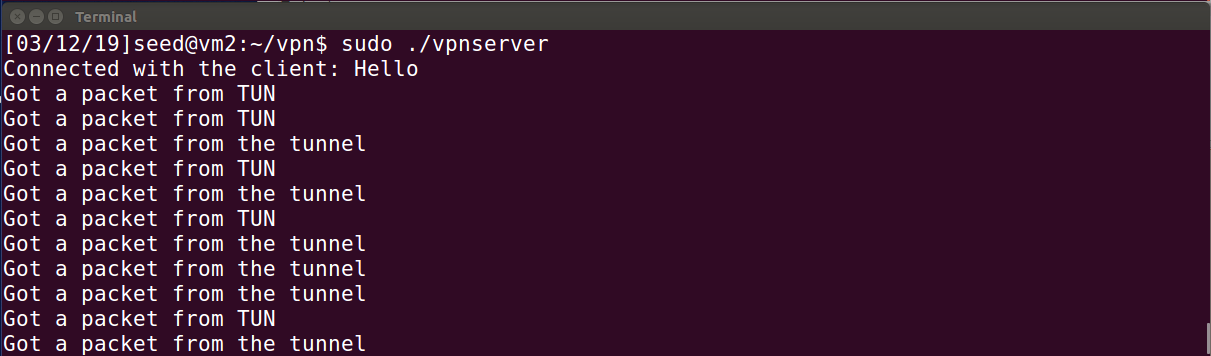
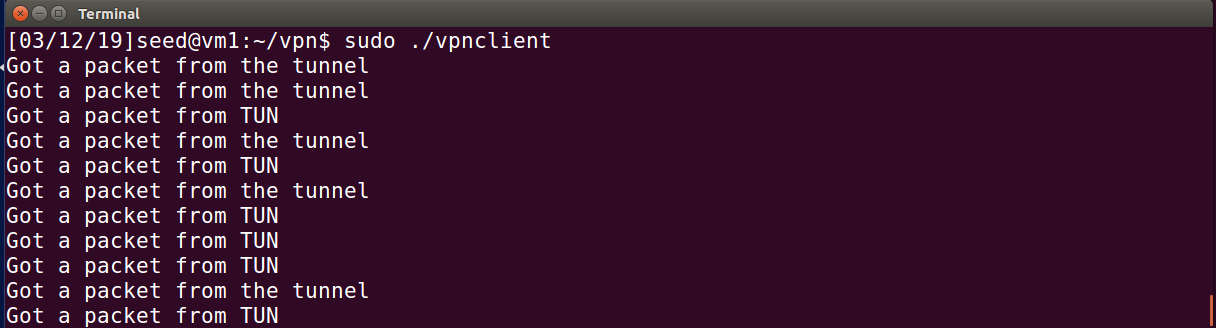
Observation: Connected to Host V from Host U over telnet to set up for tunnel breaking test.



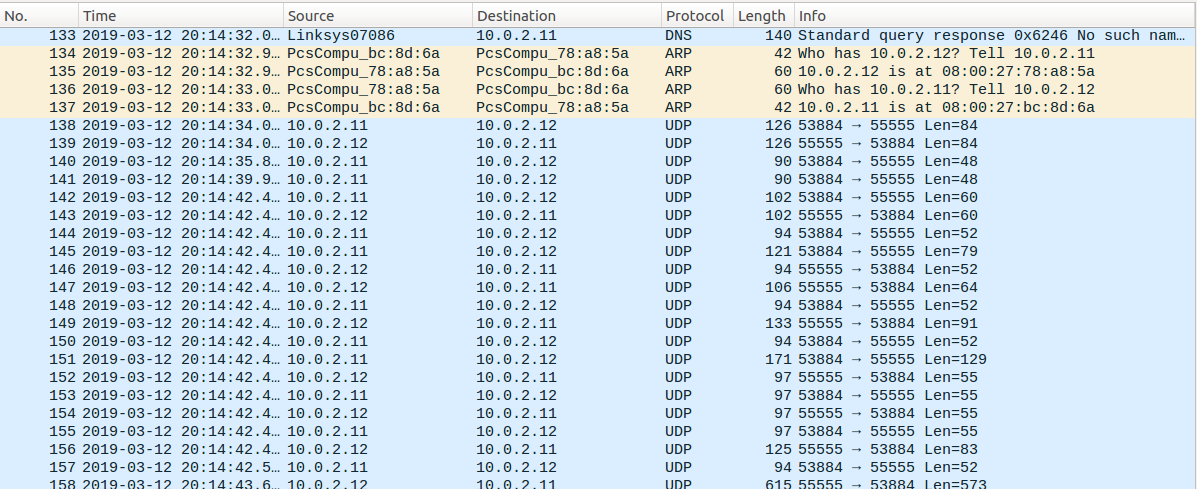
Observation: The tunnels are broken. Telnet is still working but since the packets it sends out via the broken VPN tunnel goes nowhere, TCP will keep resending the packets.



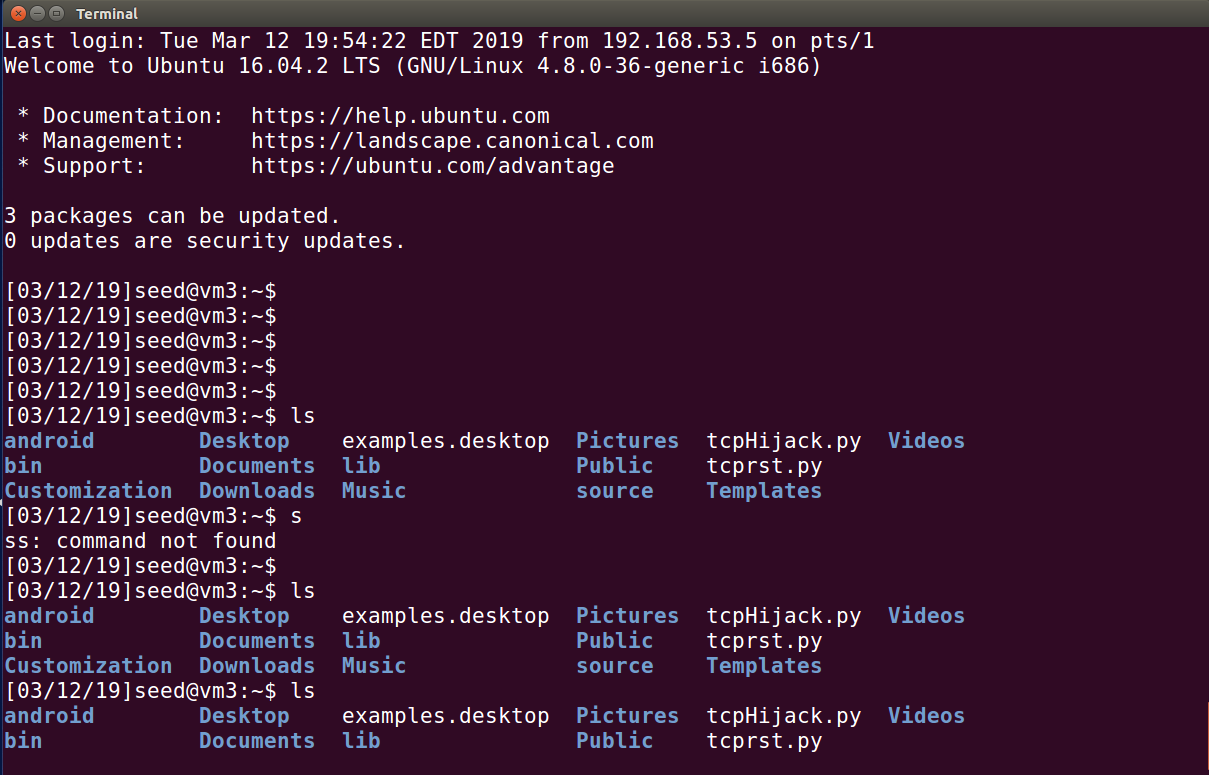
**Observation:** The telnet connection is not broken. TCP will keep resending packets, but they cannot be delivered because the tunnel is broken. Whatever was typed in telnet will be buffered by TCP, not lost, but can’t be seen. As soon as we reconnect the tunnel, everything that was typed will show up.

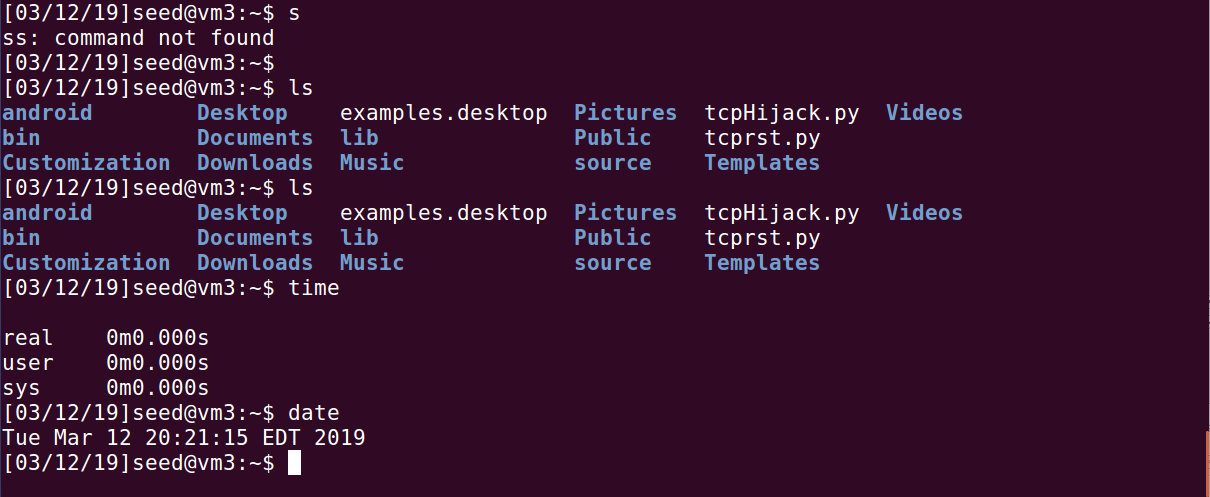
 

**Observation:** The tunnels are reconnected. I restarted the VPN server and client. Then on another terminal assigned the IP addresses 192.168.53.xx to the tun0 interfaces for both.



**Observation:** I packets that were in the buffer are sent through.





**Observation**: The tunnel was successfully reconnected and you can see whatever was typed into the telnet was actually not lost, the commands were buffered, waiting to be sent to the telnet server. When the server receives a character, it echo's the character back to the telnet client, which will print them to the terminal. Due to TCP transmission, when VPN tunnel was reconnected, all the blindly typed characters will show up on the client side.