|  |  |  |
| --- | --- | --- |
| Name: Adarsh Kumar | SRN No: PES2UG20CS016 | Assignment No: 09 |
| Section: B | Date: 11/11/2022 |

|  |  |
| --- | --- |
| **Task 1:** | **Network Setup** |
| Screenshot | Checking packet sniffing using tcpdump command    we can see that tcpdump is active and listening at interface eth0.  Testing  Pinging to server-router for client 10.9.0.5    The connection is successfully established between server-router and client.  Pinging to Host V 192.168.60.5 from server-router    As we can see that VPN Server can successfully established a connection to Host V (private network)  Pinging to Host V 192.168.60.5 from Host U 10.9.0.5    All our sent packets are lost and we are not able to establish a connection between Host U & Host V.  **Sniffing the packet on network**  Pinging to VPN Server-router from Client 10.9.0.5    Running tcpdump sniffer command on the VPN server-router.    We can see that when client 10.9.0.5 is pinging to server-router then we are able to sniff the packet. |
| **Task 2** | **Create and Configure TUN Interface** |
| Task 2.a: | Name of the Interface |
| Screenshot | On Client - 10.9.0.5 we are running the tun.py program    we can see that a new interface is created by the name of tun0 and presently it is in down state.  Killing the established tunnel    Now we are changing the name of interface    We can see that the interface name is CS0160 now SRN+0 |
| Task 2.b: | Set up the TUN Interface |
| Screenshot | The Our interface in UP state |
| Task 2.c: | Read from the TUN Interface |
| Screenshot | Now we are trying to read data passing through tun interface.    On Host U, ping a host in the 192.168.53.0/24 network. What is printed out by the tun.py program? What has happened? Why?  Ans: This is because another end of the tunnel is not setup yet, the packet found a route for 192.168.53.0/24 via tun0, so the packet pass top this interface and the application able to get the packet. For 192.168.53.99, the ping command success, because it is the address of local adapter, so the ICMP message goes into loopback interface.    On Host U, ping a host in the internal network 192.168.60.0/24, Does tun.py print out anything? Why?  Ans: A ping request to any address under 192.168.60.0/24 is unresponsive on our tunnel interface as no data is being sent or received from that interface. It happened because which command ping execute to IP address 192.168.60.1, the OS lookup the routing table and found the packet should pass via the physical interface ens33, so the packet did not pass through tun0 interface and the application cannot capture the packet.  It is worth noting that when ping192.168.53.99 is our pipe interface, it will not receive packets on this interface, but the terminal that sends the ping request can receive packets. The guess is because the system found that it was sent to its own address, so it was replaced with a loopback address. Use tcpdump to monitor the loopback address, as expected: |
| Task 2.d: | Write to the TUN Interface |
| Screenshot | 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.    It can be seen tun successfully received the message and returned the corresponding ICMP message.  Since any IP starting point and ending point is the local machine, the message will be received by the kernel (known by task 4), and the message that does not meet the requirements of the network segment will also be rejected, so the messages sent and received here are only used to show the code |
| **Task 3:** | **Send the IP Packet to VPN Server Through a Tunnel** |
|  | Listing to Server-router of VPN    As you can see, the VPN-SERVER has successfully received and is ready to forward.  Establishing a tunnel from client side HOST U    the client will automatically transfer 192.168.53.0/24 to the sun0 interface we set before, but it is not a real (virtual) interface and cannot really send packets. But our program can read it and package it and send it to the server we set. The server unpacks it and reads that message which we sent actually.    There is output for packet send to the 192.168.53.5 & 192.168.60.5. But ping test fail. The reason behind is that there is only one way traffic from HOST U to VPN Server the tunnel works in one way and no IP address assigned as 192.168.53.100 at VPN Server, and you can observe the ping packet is encapsulated inside UDP packet.  We want the packets going to HOST-V to go through tun, so we need to configure the routing table: |
| Task 4: | Set Up the VPN Server |
|  | From the output we can see that the code runs correctly and sends the ICMP packets sent in the tunnel.  Ping the Private network (192.168.60.5) from Client 10.9.0.5    On Host 192.168.60.5 running the tcpdump to capture packet    It can be seen that although there is no return function at present, the message has been correctly sent to HOST-V.  Killing the earlier Tunnel Process - |
| Task 5: | Handling Traffic in Both Directions |
|  | The client Host-U has been able to ping to Host-V normally:  Server is listening via Tun interface    Client Is listening via Tun interface |
| Wireshark Screenshot | It can be seen that ping can be roughly divided into four processes:  HOST-U sent to VPN-SERVER  VPN-SERVER sends ping request to HOST-V  HOST-V replies to VPN-SERVER's ping request  VPN-SERVER sends reply back to HOST-U |
|  | Screenshot for telnet test success    At the same time, remote login can also be completed  Client    Server |
|  | Packet capture at HOST V of the telnet test    Description  In ping test, the icmp packet at Host U is target to 192.168.60.5, by the static route setup, the packet will route to interface tun0 and next hop address is 192.168.59.100. the packet will delivery to tun 0 and capture by the application tun\_client.py. Then tun\_client.py encapsulate the icmp packet with a udp packet and deliver to 10.9.0.11 with destination port 9090. As the application tun\_server.py is started and the udp socket is listening at port 9090. The UDP packet to 10.9.0.11:9090 will captured by the tun\_server.py and the application will decapsulate the UDP packet and extract its payload to become the IP packet. After that the application will pass the packet to tun0 right away. By the IP forward function is enabled at VPN server’s kernel. The decapsulated packet will forward to 192.168.60.5 according to the destination address at ip header.  Vice versa, the HOST V server receive the ICMP request and response with ICMP reply to VPN server. According to the routing table at VPN Server, the VPN Server receive an icmp reply packet with src ip 192.168.60.5 and destination 192.168.53.99. the VPN server will forward this packet to tun0 interface and capture by tun\_server.py application. The application will encapsulate the packet with a UDP packet with destination ip 10.9.0.11 and destination port 9090 Via ens33 interface at VPN server. At next step, the ens33 interface at Host U will receive the UDP packet and the application has a udp socket that is listening at 9090 port. The application will receive the packet and decapsulate to the icmp reply packet. Finally, it passes back to tun0 interface and OS will pass to ping application with success result. |
| Task 6: | Tunnel-Breaking Experiment |
| Screenshot | Telnet to client to establish internet connection    Do you see what you type? What happens to the TCP connection? Is the connection broken?   * We found that no matter what was entered, nothing was displayed. The TCP connection is not broken. the connection can resume if the disconnection time is not too long. The character typed after tunnel breaks can resume and send to the telent session. And telnet session is resumed without issue. * From my understanding telnet use TCP as protocol, the packet send without ack from remote end will resend within certain time windows. That makes the connection persistent and recoverable even the UDP tunnel breaks. However, the disconnect time beyond the Retransmission timeout RTO. In RFC 1122, the recommendation is at least 100 seconds for the timeout, which corresponds to a value of at least 8, ubuntu default at 15. Another recommendation is at least 3 retransmissions, which is the default at ubuntu.   Once the tunnel is re-established, what is going to happen to the telnet connection? Please describe and explain your observations?   * While keeping the remote login online, the same situation occurs when the tunnel service of the client or server is broken, that is, no text can be entered in the remote login interface, and there will be no new output. When the service is reconnected for a short period of time, the backlogged packet buffers in the TUN file will be released one by one. As shown in the figure below:   On VPN Server-Router    On the private network we can see that packets are also receiving |

**THE END**