Introduction

Transmitting IP packets through the public internet is a bad idea due to bad guys who might be waiting to grab our confidential data in IP packet payloads. One solution for this would be building our own wired network infrastructure with our own cables, routers and stuff which will be physically protected against wiretapping. However unfortunately, this is not practical. Therefore, the next choice we have is encrypting our IP packets as a whole or partially which can be sent as a payload of another IP packet through the Internet. This is how some flavors of IPSec work (i.e. ESP).

In this assignment, your task is to implement a small system using which two hosts can communicate with each other securely. It is important to make sure that our IP packet encryption and decryption functionalities are transparent to the application layer. You are not required to implement something completely compatible to IPSec following the standard. It is good enough to implement a your own mechanism to encrypt IP packets and putting them in the payload of another IP packet.

Architecture of the system

Usually when we ping from a host to another host, ICMP packets go through the first host to the second host which will bounce back from the second host to the first host. In ordinary case, we are using the real networking interfaces available on the host computers such as eth0 and wlan0. But, when you send packets through such interfaces, your packets go out as plain text.

The diagram shown in Figure-1 illustrates the architecture of the system we are expecting to have. In this case, we are using TUN networking interfaces instead of real networking interfaces. A TUN interface is a virtual network interface which we can create on Linux. We can set an IP address to it and use any network related application program with the TUN interface just like the real interface. However, when you give packets to it, it cannot send those packets out since there's no any real network card associated with the TUN network interface. Therefore, it provides a file descriptor which we can read from and write to exchange the packets with another control program written by us.

As shown in the diagram, we should create a virtual TUN interface called asa0 on two host computers and then should run a control program to deal with the TUN interface on each host. In this case, we have named the two programs as tun-client.c and tun-server.c. We can assign an IP address for the interfaces using ip command. From one computer if you ping to the IP address of the other TUN interface in the other host, you will not receive any response because these TUN interfaces are not connected using any real physical media. So, what we have done is creating a TCP socket connection between the two control programs to relay any data between the two TUN interfaces on two hosts. This connection is marked as connection X in the diagram.

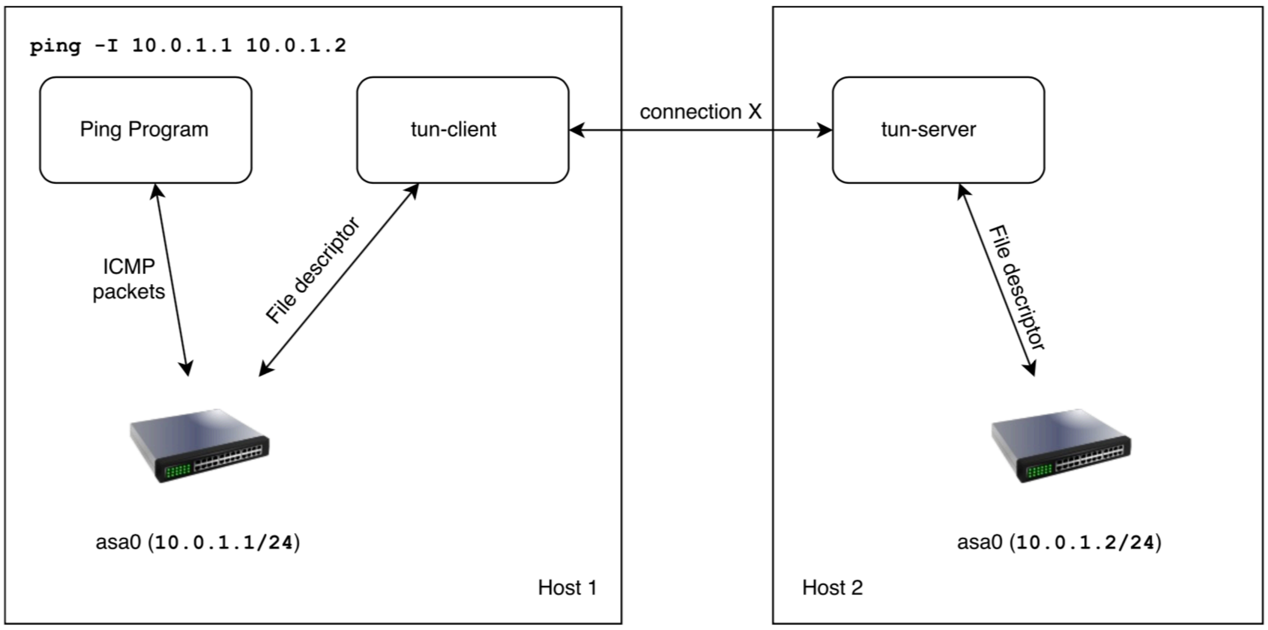

Figure 1: High-level overview of the system.

In this interesting setup, we are delivering our IP packets inside a TCP tunnel but of course still in plain text. Source code of this setup is provided to you and this is how we run it.

1. In the first computer, run following commands to setup a TUN interface called asa0,
   ```
   sudo ip tuntap add dev asa0 mode tun
   sudo ip addr add 10.0.1.1/24 dev asa0
   sudo ip link set dev asa0 up
   ip addr show
   ```

2. In the second computer, run following commands to setup a TUN interface called asa0,
   ```
   sudo ip tuntap add dev asa0 mode tun
   sudo ip addr add 10.0.1.2/24 dev asa0
   sudo ip link set dev asa0 up
   ip addr show
   ```

3. Compile and run TUN controller program which is also a TCP server on host 2,
   ```
   ./tun-server
   ```

4. Compile and run TUN controller program which is also a TCP client on host 1,
   ```
   ./tun-client
   ```

5. Ping from host 1 to host 2 where our ping packets will be delivered through the TCP client and TCP server in the TCP socket connection (connection x),
   ```
   ping -I 10.0.1.1 10.0.1.2
   ```

Your task

If we intercept the packets going between the two hosts using Wireshark, in current case, we will be able to see the IP packets with a TCP payload. The TCP payload is the encapsulated IP packet in plain text. You need to improve the security of this connection x and the amount of marks you get is proportional to how close your implementation is to IPSec.
If you improve the tun-client.c and tun-server.c programs to encrypt/decrypt the IP payload before sending through the TCP tunnel, you get 60% which is the minimum acceptable work. When I run Wireshark, I should be able to still see the outer IP header and TCP header but the payload must be encrypted.

If you switch the connection x to a raw socket connection and implement something like IPSec ESP transport mode, you will get 80%. When I run Wireshark, I should be able to see the original IP header of the packet came from the ping program but anything beyond should be encrypted. (your packet structure for the ESP header does not have to be same as the original IPSec specification.)

If you switch the connection x to a raw socket connection and implement something like IPSec ESP tunnel mode, you will get 100%. When I run Wireshark, I should be able to see only the new IP header which is not what came from the ping program. Our full original IP packet should be encrypted and placed in the payload of the outer IP packet. (your packet structure for the ESP header does not have to be same as the original IPSec specification.)

Note: You have freedom to decide the encryption algorithms you use. Additionally, the keys are manually copied to the two hosts before running our setup.