Jevin Modi

March 3rd, 2024

CS 528 – HW2

Virtual Private Network (VPN) Lab

Answers to the questions-

1) It is better to use UDP for the tunnel than TCP. The way VPN works is that the raw packet from my computer (source) is wrapped around with another packet with the source and destination address being that of the tunnel. When it reaches the destination tunnel, the raw packet is extracted and sent to the destination, which may be a TCP connection. So, at the final destination if packets are missing or arrived out of order it will send back a reply through the tunnel to the source destination. Hence, there is no need for the tunnel to send and receive reply or keep track of missing/ out of order packets. The tunnel should be just fast in delivering the packets. And hence, it is better to use UDP for the tunnel.
2) It is not recommended to use our own encryption/ decryption, HMAC or any other cryptographic algorithms because our own implementations are not tested thoroughly and could be vulnerable to cryptographic or even side channel attacks. The algorithms already implemented are tested very well by researchers and engineers and implemented in ways to prevent attacks on it.
3) It is important for the server to release resources when a connection is broken because it will help the server manage resources effectively. It can provide better services to the connected clients by freeing up unwanted resources. If the server does not free up the resources upon closed connections, there are chances that the server might become overwhelmed, and cases of denial-of-services could arise.

Explanation about my implementation-

In my implementation of MiniVPN, I have created two sockets bound to two different ports. One of the sockets is used for UDP and other for TCP. UDP is used for data transmission whereas TCP is used for key exchange and authentication. Authentication is done through certificates and key pairs. OpenSSL library is used for the implementation of SSL connection.

Firstly, I converted the code which used TCP to UDP. After that I implemented aes encryption for confidentiality and HMAC SHA 256 for integrity. This works in the following way.

To begin with, our code sits in the middle of the TUN/TAP and the network/tunnel. So, we read the plaintext from TUN/TAP. After that we encrypt the plaintext and then calculate HMAC using encrypt-then-HMAC technique. I used this technique as this does not expose our plain text at all. We can verify the HMAC first and only when it is successful, we decrypt it. So, if something is tampered with then we do not need to decrypt as the HMAC itself will fail and we can drop the packet. This may also save up computational resources. After

calculating the HMAC we write the encrypted text and HMAC to the network. Similarly, when we receive something from the network, we read it then, as discussed earlier, we verify the HMAC first. Upon successful verification, we decrypt it and then write it to TUN/TAP. I tested this task by hard coding the iv and key. In the next task we authenticate the client and server and exchange the keys and iv. For encryption, decryption and HMAC I have used functions provided by OpenSSL.

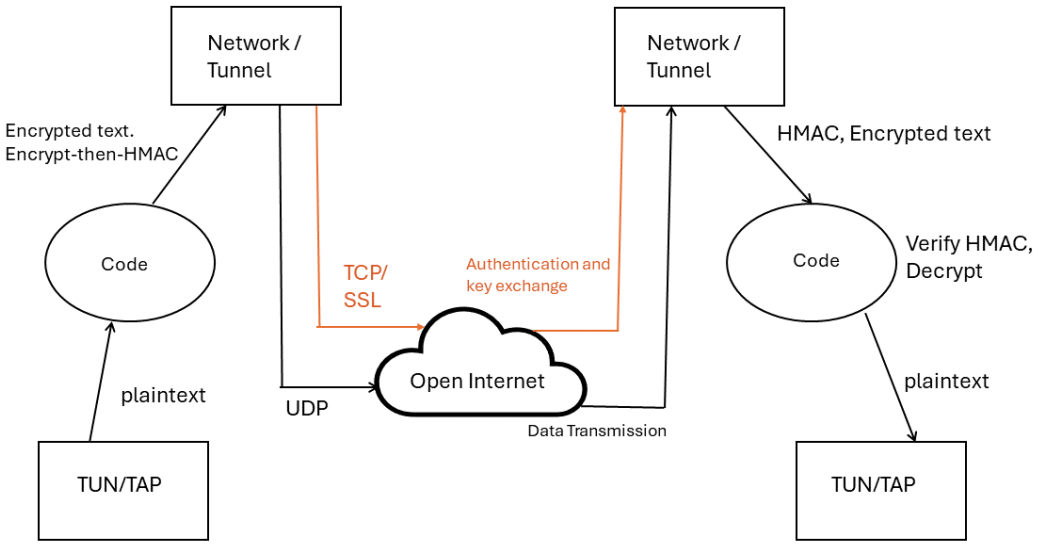
For the authentication and key exchange part, I created an SSL connection between the client and server. For that I first created a root CA and created a certificate signed by itself. After that I generated public/private key pairs for the server and client. And ultimately the CA issued a certificate using the key pair for the client and the server. At this point, I have two code files – simpletun.c (server) and simpletunClient.c (client) since I have hard coded the names of the certificate file and the key file.

Let me first talk about the server-side implementation. I have defined another port namely SERVER_PORT at 55111. I have first initialized the OpenSSL. Then created a context with the server_method. After that I load my certificate and key files into that context. Then I load my CA certificate to verify the server certificate. After that I create a TCP socket and bind it to SERVER_PORT. After that the socket starts listening for incoming connections on that port. I then bind that socket to ssl object. So then when there is a new connection request, I use SSL_accecpt to accept that request and then the SSL_connection is established. OpenSSL takes care of verifying the certificates and authenticating the other party. Once we have the SSL connection set up, we can generate random key and iv and using ssl_write send it to the client. The client side implementation pretty much works in the same way. The only difference is that our socket will try to connect to the SERVER_PORT and then create an SSL connection using SSL_connect. Once we have the SSL connection established we can read the key and iv send by the server through this channel using ssl_read. To shut down the ssl connection we use SSL_shutdown and we free the bio object and the context.

To support multiple clients, we create a thread on the server for each client connection request. This will make sure that each client connection is isolated from the other and upon closing the connection, we can just delete the thread and this will free up the resources. But we might still need to careful about the security implications, as one thread might unknowingly reveal information that might lead to vulnerabilities and possible attacks.

One thing to note is that client and server concepts only make sense during the initial setup of the connection. After the connection is established, both are just two ends of the tunnel.

Below is a figure showing a high level idea of the working of MiniVPN-



These are the screenshots of execution-

These are for the server-

```
[03/03/2024 19:33] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ nano simpletun.c
[03/03/2024 19:35] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ gcc -o simpletun simpletun.c -lssl -lcrypto
[03/03/2024 19:37] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ ser
[sudo] password for cs528user:
Successfully connected to interface tun0
Enter PEM pass phrase:
Server listening on port 55111
SSL connection established.
Key sent to client successfully.
IV sent to client successfully.
Key and IV sent to client successfully.
SERVER: Waiting for packets...
NET2TAP 1: Read 96 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 96 bytes to the network
NET2TAP 2: Read 96 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 96 bytes to the network
NET2TAP 3: Read 96 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 96 bytes to the network
NET2TAP 4: Read 96 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 96 bytes to the network
```

```
[03/03/2024 19:36] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ sr
[sudo] password for cs528user:
[03/03/2024 19:37] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ pt
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_req=1 ttl=64 time=0.038 ms
64 bytes from 10.0.1.1: icmp_req=2 ttl=64 time=0.044 ms
64 bytes from 10.0.1.1: icmp_req=3 ttl=64 time=0.069 ms
64 bytes from 10.0.1.1: icmp_req=4 ttl=64 time=0.074 ms
64 bytes from 10.0.1.1: icmp_req=5 ttl=64 time=0.068 ms
64 bytes from 10.0.1.1: icmp_req=6 ttl=64 time=0.065 ms
64 bytes from 10.0.1.1: icmp_req=7 ttl=64 time=0.061 ms
64 bytes from 10.0.1.1: icmp_req=8 ttl=64 time=0.067 ms
64 bytes from 10.0.1.1: icmp_req=9 ttl=64 time=0.057 ms
64 bytes from 10.0.1.1: icmp_req=10 ttl=64 time=0.058 ms
64 bytes from 10.0.1.1: icmp_req=11 ttl=64 time=0.057 ms
64 bytes from 10.0.1.1: icmp_req=12 ttl=64 time=0.079 ms
64 bytes from 10.0.1.1: icmp_req=13 ttl=64 time=0.063 ms
64 bytes from 10.0.1.1: icmp_req=14 ttl=64 time=0.052 ms
64 bytes from 10.0.1.1: icmp_req=15 ttl=64 time=0.050 ms
64 bytes from 10.0.1.1: icmp_req=16 ttl=64 time=0.050 ms
64 bytes from 10.0.1.1: icmp_req=17 ttl=64 time=0.067 ms
^C
--- 10.0.1.1 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16027ms
rtt min/avg/max/mdev = 0.038/0.059/0.079/0.014 ms
[03/03/2024 19:38] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$
```

These are for the client-

```
[03/03/2024 19:33] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ gcc -o simpletunClient simpletunClient.c -lssl -lcrypto
[03/03/2024 19:37] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ sudo ./simpletunClient -i tun0 -c 192.168.15.8 -d
[sudo] password for cs528user:
Successfully connected to interface tun0
Enter PEM pass phrase:
SSL connection established.
CLIENT: Connected to server 192.168.15.8
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 96 bytes to the network
NET2TAP 1: Read 96 bytes from the network
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 96 bytes to the network
NET2TAP 2: Read 96 bytes from the network
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 96 bytes to the network
NET2TAP 3: Read 96 bytes from the network
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 96 bytes to the network
NET2TAP 4: Read 96 bytes from the network
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 5: Read 84 bytes from the tap interface
TAP2NET 5: Written 96 bytes to the network
```

```
[03/03/2024 19:36] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ cr
[sudo] password for cs528user:
[03/03/2024 19:37] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$ pt
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_req=1 ttl=64 time=1.24 ms
64 bytes from 10.0.1.1: icmp_req=2 ttl=64 time=1.21 ms
64 bytes from 10.0.1.1: icmp_req=3 ttl=64 time=1.29 ms
64 bytes from 10.0.1.1: icmp_req=4 ttl=64 time=1.07 ms
64 bytes from 10.0.1.1: icmp_req=5 ttl=64 time=1.03 ms
64 bytes from 10.0.1.1: icmp_req=6 ttl=64 time=1.11 ms
64 bytes from 10.0.1.1: icmp_req=7 ttl=64 time=1.16 ms
64 bytes from 10.0.1.1: icmp_req=8 ttl=64 time=1.16 ms
64 bytes from 10.0.1.1: icmp_req=9 ttl=64 time=1.12 ms
64 bytes from 10.0.1.1: icmp_req=10 ttl=64 time=1.19 ms
64 bytes from 10.0.1.1: icmp_req=11 ttl=64 time=1.10 ms
64 bytes from 10.0.1.1: icmp_req=12 ttl=64 time=1.10 ms
^C
--- 10.0.1.1 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11049ms
rtt min/avg/max/mdev = 1.031/1.152/1.292/0.083 ms
[03/03/2024 19:38] cs528user@cs528vm:~/Desktop/vpn_code/CS528-lab2-vpn-lab/CS528-lab2-vpn-lab$
```