VPN Tasks 3-5

Client/Host U: 10.0.2.15

Server: 10.0.2.22

Host V: 192.168.60.101 (10.0.2.10)

Task 3: Encrypting the Tunnel

First step is to generate a self signed root CA. Then we generate a server certificate which is signed by root CA. We then added an entry for hostname rakshith2294.com as server's IP address in /etc/hosts file in both client and server.

```
[04/24/2020 00:03] Rakshith-10.0.2.22@VM:~/.../cert_server$openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
Generating a 1024 bit RSA private key
.....................................................................................................++++++
.....................++++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:SYR
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Syracuse
Organizational Unit Name (eg, section) []:VPN
Common Name (e.g. server FQDN or YOUR name) []:rakshith2294.com
Email Address []:rnallaha@syr.edu
[04/24/2020 00:07] Rakshith-10.0.2.22@VM:~/.../cert_server$
```

```
[04/24/2020 15:35] Rakshith-10.0.2.22@VM:~/.../cert_server$openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \-config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4099 (0x1003)
        Validity
            Not Before: Apr 24 19:36:05 2020 GMT
            Not After : Apr 24 19:36:05 2021 GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = NY
            localityName              = SYR
            organizationName          = VPN Server
            organizationalUnitName    = VPN
            commonName                = rakshith2294.com
            emailAddress              = rnallaha@syr.edu
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                5A:34:E7:FF:FA:27:90:25:38:FB:FF:D9:AE:E9:3D:DE:51:73:25:DA
            X509v3 Authority Key Identifier:
                keyid:D9:CD:1E:31:6F:A6:85:E8:A8:67:7B:A4:31:0A:5A:3E:1E:8F:BF:CB

Certificate is to be certified until Apr 24 19:36:05 2021 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
[04/24/2020 15:36] Rakshith-10.0.2.22@VM:~/.../cert_server$
```

In VPN tasks 1-3 we used UDP tunnel to encapsulate traffic, but our traffic was not encrypted because of which we cannot achieve the goal of data integrity and confidentiality. In order to encrypt the data we have make use of SSL /TLS. TLS is typically built on top of TCP hence in our code we replaced the UDP channel by TCP. We replaced the normal read and write function using ssl_read and ssl_write, latter function for encrypting the traffic before sending and former for reading the encrypted traffic. We load the root CA cert and key in our client program, we load server key and certificate in our server program,

once we run the program first a TCP handshake will happen between client and server, once the handshake is complete, the client will initiate a hello sending its supported ciphers, protocols etc, server will then respond with a hello and its supported parameters, server will send its certificate and public key which is used to create a pre master secret which encrypts data, and generates master secret which is then used to secure the connection between client and server.

Server Program:

When we setup TCP Server, we bind a TCP socket; it is set to accept connections from any clients, on port number 4433. In the next function we define what to do when the packet is on Tunnel interface and TCP socket.

```
int setupTCPServer()
{
    struct sockaddr_in sa_server;
    int listen_sock;
    listen_sock= socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    CHK_ERR(listen_sock, "socket");
    memset (&sa_server, '\0', sizeof(sa_server));
    sa_server.sin_family      = AF_INET;
    sa_server.sin_addr.s_addr = INADDR_ANY;
    sa_server.sin_port        = htons (4433);
    int err = bind(listen_sock, (struct sockaddr*)&sa_server, sizeof(sa_server));
    CHK_ERR(err, "bind");
    err = listen(listen_sock, 5);
    CHK_ERR(err, "listen");
    return listen_sock;
}
void processRequest(SSL* ssl, int sockfd, int tunfd)
{
    char buf[1024];
    int len = SSL_read (ssl, buf, sizeof(buf) - 1);
    buf[len] = '\0';
    while (1) {
    fd_set readFDSet;
    FD_ZERO(&readFDSet);
    FD_SET(sockfd, &readFDSet);
    FD_SET(tunfd, &readFDSet);
    select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);
    if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd,ssl);
    if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd,ssl);
  }
    SSL_shutdown(ssl);  SSL_free(ssl);
}
```

When packet is on tunnel interface we read the normal text, encrypt it using SSL, and send it to client.

When packet is on socket interface, we read encrypted data, decrypt it and write it to appropriate host.

```
struct sockaddr_in peerAddr;
int createTunDevice() {
  int tunfd;
  struct ifreq ifr;
  memset(&ifr, 0, sizeof(ifr));
  ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
  tunfd = open("/dev/net/tun", O_RDWR);
  ioctl(tunfd, TUNSETIFF, &ifr);
  return tunfd;
}
void tunSelected(int tunfd, int sockfd, SSL *ssl) {
  struct ip* tunsock;
  int  len;
  char buff[BUFF_SIZE];
  bzero(buff, BUFF_SIZE);
  len = read(tunfd, buff, BUFF_SIZE);
  buff[len] = '\0';
  SSL_write(ssl, buff, len);
}
void socketSelected(int tunfd, int sockfd, SSL *ssl) {
  struct ip* tcpsock;
  int  len;
  char buff[BUFF_SIZE];
  bzero(buff, BUFF_SIZE);
  len = SSL_read (ssl, buff, BUFF_SIZE);
  buff[len] = '\0';
  write(tunfd, buff, len);  |
}
```

```c
int main()
{
  SSL_METHOD *meth;
  SSL_CTX* ctx;
  SSL *ssl;
  int err;
  SSL_library_init();
  SSL_load_error_strings();
  meth = (SSL_METHOD *)TLSv1_2_method();
  ctx = SSL_CTX_new(meth);
  SSL_CTX_set_verify(ctx, SSL_VERIFY_NONE, NULL);
  SSL_CTX_use_certificate_file(ctx, "/home/seed/VPN-lab/cert_server/server_crt.pem", SSL_FILETYPE_PEM);
  SSL_CTX_use_PrivateKey_file(ctx, "/home/seed/VPN-lab/cert_server/server_key.pem", SSL_FILETYPE_PEM);
  ssl = SSL_new(ctx);
  int tunfd,sockfd;
  tunfd = createTunDevice();
  system("/home/seed/VPN-lab/vpn/config_server.sh");
  struct sockaddr_in sa_client;
  int client_len = sizeof(sa_client);
  int listen_sock = setupTCPServer();
  while(1){
    int sock = accept(listen_sock, (struct sockaddr*)&sa_client, &client_len);
    if (fork() == 0) {
       close (listen_sock);
       SSL_set_fd (ssl, sock);
       int err = SSL_accept (ssl);
       CHK_SSL(err);
       printf ("SSL connection established!\n");

       processRequest(ssl, sock, tunfd);
       close(sock);
       return 0;
    } else {
        close(sock);
```

Client Program: In Client program, we setup TLS client and TCP client, in TLS client we setup the location of root CA, and while setting up TCP client we bind a TCP socket to server's IP address and port number.

```c
SSL* setupTLSClient()
{
    SSL_library_init();
    SSL_load_error_strings();
    SSL_METHOD *meth = (SSL_METHOD *)TLSv1_2_method();
    SSL_CTX* ctx = SSL_CTX_new(meth);;
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
    if(SSL_CTX_load_verify_locations(ctx,NULL,"/home/seed/vpn/ca_client/") < 1){
   printf("Location verification failed\n");
   exit(0);}
    SSL* ssl = SSL_new (ctx);
    return ssl;
}

int setupTCPClient()
{
    struct sockaddr_in server_addr;
    int sockfd= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&server_addr, '\0', sizeof(server_addr));
    server_addr.sin_addr.s_addr=htonl(167772694);
    server_addr.sin_port    = htons (port);
    server_addr.sin_family = AF_INET;
    connect(sockfd, (struct sockaddr*) &server_addr,sizeof(server_addr));
    return sockfd;
}
```

First there will be a TCP handshake, after that TLS handshake, and then the tunnel will be encrypted by SSL.

```c
int main()
{
    int tunfd;
    tunfd = createTunDevice();
    system("/home/seed/vpn/config_client.sh");
    /*TLS initialization*/
    SSL *ssl   = setupTLSClient();
    /*TCP Handshake*/
    int sockfd = setupTCPClient();
    /*TLS handshake*/
    SSL_set_fd(ssl, sockfd);
    int err = SSL_connect(ssl);
    CHK_SSL(err);
    printf ("SSL connection using %s\n", SSL_get_cipher(ssl));
    printf("MiniVPN Connection successful\n");
    while (1) {
        fd_set readFDSet;
        FD_ZERO(&readFDSet);
        FD_SET(sockfd, &readFDSet);
        FD_SET(tunfd, &readFDSet);
        select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);
        if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd,ssl);
        if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd,ssl);
    }
}
```

```
[04/27/2020 02:16] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client
SSL connection using AES256-GCM-SHA384
MiniVPN Connection successful
```

```
[04/27/2020 02:16] Rakshith-10.0.2.22@VM:~/.../vpn$sudo ./tcp_server
[sudo] password for seed:
Enter PEM pass phrase:
SSL connection established!
```

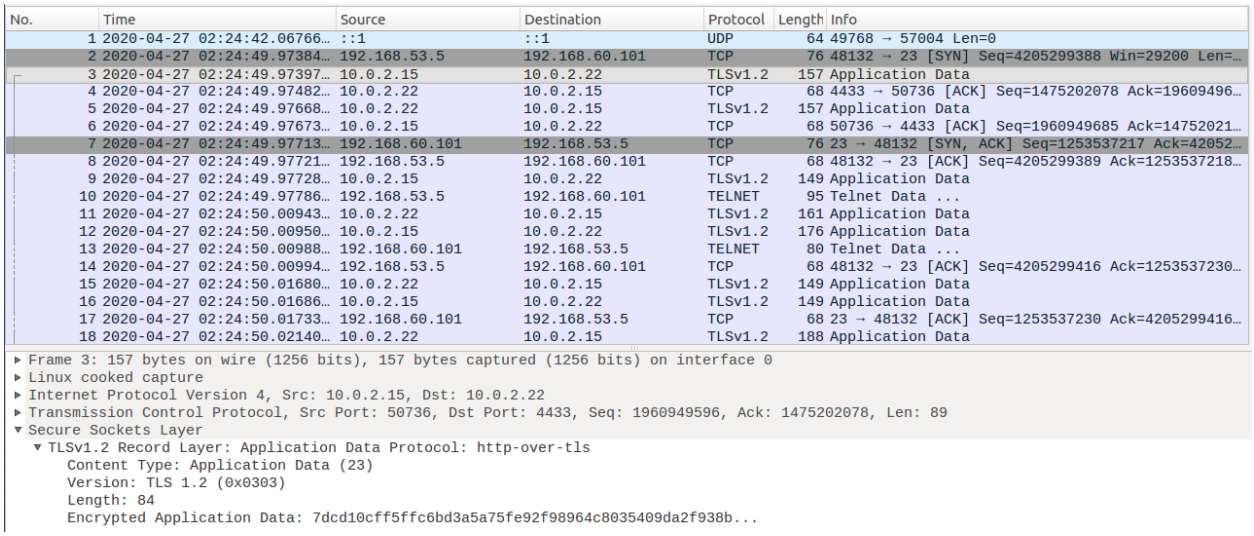
```
[04/27/2020 02:24] Rakshith-10.0.2.15@VM:~/vpn$telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Apr 26 16:19:04 EDT 2020 from 192.168.53.5 on pts/1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[04/27/2020 02:25] Rakshith-10.0.2.10@VM:~$
```

As you can see we are successfully able to telnet to host V from host U, and we can see that our data encapsulated by two ends of tunnels and data is encrypted as seen in the wireshark output.



No.	Time	Source	Destination	Protocol	Length	Info
1	2020-04-27 02:24:42.06766…	::1	::1	UDP	64	49768 → 57004 Len=0
2	2020-04-27 02:24:49.97384…	192.168.53.5	192.168.60.101	TCP	76	48132 → 23 [SYN] Seq=4205299388 Win=29200 Len=…
3	2020-04-27 02:24:49.97397…	10.0.2.15	10.0.2.22	TLSv1.2	157	Application Data
4	2020-04-27 02:24:49.97482…	10.0.2.22	10.0.2.15	TCP	68	4433 → 50736 [ACK] Seq=1475202078 Ack=19609496…
5	2020-04-27 02:24:49.97668…	10.0.2.22	10.0.2.15	TLSv1.2	157	Application Data
6	2020-04-27 02:24:49.97673…	10.0.2.15	10.0.2.22	TCP	68	50736 → 4433 [ACK] Seq=1960949685 Ack=14752021…
7	2020-04-27 02:24:49.97713…	192.168.60.101	192.168.53.5	TCP	76	23 → 48132 [SYN, ACK] Seq=1253537217 Ack=42052…
8	2020-04-27 02:24:49.97721…	192.168.53.5	192.168.60.101	TCP	68	48132 → 23 [ACK] Seq=4205299389 Ack=1253537218…
9	2020-04-27 02:24:49.97728…	10.0.2.15	10.0.2.22	TLSv1.2	149	Application Data
10	2020-04-27 02:24:49.97786…	192.168.53.5	192.168.60.101	TELNET	95	Telnet Data ...
11	2020-04-27 02:24:50.00943…	10.0.2.22	10.0.2.15	TLSv1.2	161	Application Data
12	2020-04-27 02:24:50.00950…	10.0.2.15	10.0.2.22	TLSv1.2	176	Application Data
13	2020-04-27 02:24:50.00988…	192.168.60.101	192.168.53.5	TELNET	80	Telnet Data ...
14	2020-04-27 02:24:50.00994…	192.168.53.5	192.168.60.101	TCP	68	48132 → 23 [ACK] Seq=4205299416 Ack=1253537230…
15	2020-04-27 02:24:50.01680…	10.0.2.22	10.0.2.15	TLSv1.2	149	Application Data
16	2020-04-27 02:24:50.01686…	10.0.2.15	10.0.2.22	TLSv1.2	149	Application Data
17	2020-04-27 02:24:50.01733…	192.168.60.101	192.168.53.5	TCP	68	23 → 48132 [ACK] Seq=1253537230 Ack=4205299416…
18	2020-04-27 02:24:50.02140…	10.0.2.22	10.0.2.15	TLSv1.2	188	Application Data

```
▸ Frame 3: 157 bytes on wire (1256 bits), 157 bytes captured (1256 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.22
▸ Transmission Control Protocol, Src Port: 50736, Dst Port: 4433, Seq: 1960949596, Ack: 1475202078, Len: 89
▾ Secure Sockets Layer
  ▾ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 84
      Encrypted Application Data: 7dcd10cff5ffc6bd3a5a75fe92f98964c8035409da2f938b...
```

Task 4: Authenticating the VPN Server:

Before a VPN is established, the VPN client must authenticate the VPN server, making sure that the server is not a fraudulent one. There are three important steps in server authentication:

Verifying that the server certificate is valid:

We write a callback function in the client to verify if the server certificate is valid. The client checks the server certificate and verifies the common name and whether the server certificate is signed; if the verification fails the TLS connection is terminated. This function is invoked by **ssl_CTX_set_verify(),** and this verifies the validity of certificate.

```
int verify_callback(int preverify_ok, X509_STORE_CTX *x509_ctx)
{
    char  buf[300];
    X509* cert = X509_STORE_CTX_get_current_cert(x509_ctx);
    X509_NAME_oneline(X509_get_subject_name(cert), buf, 300);
    printf("subject = %s\n", buf);
    if (preverify_ok == 1) {
       printf("Verification passed\n");
    } else {
       int err = X509_STORE_CTX_get_error(x509_ctx);
       printf("Verification failed: %s.\n",
     X509_verify_cert_error_string(err));
       exit(2);
    }
}
```

Verifying that the server is the owner of the certificate:

Client program verifies whether it is the owner of the server certificate, it sends a bunch of root CA's loaded at specific directory, it verifies if a particular CA owns any of the server certificate. This is done by the function call SSL_CTX_load_verify_locations.

Verifying that the server is the intended server:

We introduce hostname check during SetupTLSclient, we make use of **SSL_get0_param** which checks if the hostname matches with the peer certificate and **X509_VERIFY_PARAM_set1_host** sets corresponding bit to 1 if success and 0 if failed. This verifies if the hostname is indeed real. So, our modified client codes looks like the below.

```
SSL* setupTLSClient(const char* hostname)
{
    SSL_library_init();
    SSL_load_error_strings();
    SSL_METHOD *meth = (SSL_METHOD *)TLSv1_2_method();
    SSL_CTX* ctx = SSL_CTX_new(meth);;
    SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, verify_callback);
    if(SSL_CTX_load_verify_locations(ctx,NULL,"/home/seed/vpn/ca_client/") < 1){
   printf("Location verification failed\n");
   exit(0);}
    SSL* ssl = SSL_new (ctx);
    X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
    X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);
    return ssl;
}

int setupTCPClient(const char* hostname, int port)
{
    struct sockaddr_in server_addr;
    struct hostent *hp = gethostbyname(hostname);
    struct sockaddr* ip = (struct sockaddr *) result->ai_addr;
    int sockfd= socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&server_addr, '\0', sizeof(server_addr));
    memcpy(&(server_addr.sin_addr.s_addr),hp->h_addr,hp->h_length);
    server_addr.sin_port    = htons (port);
    server_addr.sin_family  = AF_INET;
    connect(sockfd, (struct sockaddr*) &server_addr,sizeof(server_addr));
    return sockfd;
}
```

```
[04/27/2020 03:54] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client
Incorrect parameters please provide hostname and port no
[04/27/2020 03:54] Rakshith-10.0.2.15@VM:~/vpn$
```

Server is not the intended server, DNS is poisoned but the hostname verification is successful

```
[04/27/2020 03:55] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client facebook.com 4433
Connecting to server facebook.com...
Verification failed: Hostname mismatch.
Closing connection
[04/27/2020 03:55] Rakshith-10.0.2.15@VM:~/vpn$
```

Intended Server

```
[04/27/2020 03:55] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client rakshith2294.com 4433
Connecting to server rakshith2294.com...
Verification passed.
Verification passed.
SSL connection using AES256-GCM-SHA384
MiniVPN Connection successful
```

Task 5: Authenticating the VPN Client

Accessing the machines inside a private network is a privilege that is only granted to authorized users. Therefore, only authorized users are allowed to establish a VPN tunnel with the VPN server. In this task, authorized users are those who have a valid account on the VPN server. We will therefore use the standard password authentication to authenticate users.

We write a function client_auth in our vpn client code, we use normal scanf to obtain username, and write it to the tunnel, the server will receive the username data and invoke getspnam() function to search the shadow file entry for the provided username, if entry exists it will write 1 to the tunnel else it will write 0. If client receives 1, it will ask user for password using getpass() function through which user can type password without displaying it, we then write the password to the tunnel which will be read by server. Shadow file contains the password as hash value for each username, but is not displayed directly, instead it maintains a database entry to the encrypted password, getspnam() provides the pointer to the structure which contains entry for the username in the shadow file, we then use crypt function to generate a hash for the password received by the tunnel from user and compare it using strcmp with the hash provided by getspnam pointer, if it returns 0 then passwords match, else passwords are not same and user is not authenticated.

Server modified code:

```
void processRequest(SSL* ssl, int sockfd, int tunfd)
{
    char buf[1024];
    int len = SSL_read (ssl, buf, sizeof(buf) - 1);
    buf[len] = '\0';
    printf("Connection request from client %s \n",buf);
    struct spwd *pass;
    char *hash;
    pass = getspnam(buf);
  if (pass == NULL) {
   printf("Request Terminated: User not found\n");
   SSL_write(ssl, "0", 1);
   return;
  }
    SSL_write(ssl, "1", 1);
    len = SSL_read (ssl, buf, sizeof(buf) - 1);
    buf[len] = '\0';
    hash = crypt(buf, pass->sp_pwdp);
    if (strcmp(hash, pass->sp_pwdp)) {
  printf("Incorrect password\n");
  SSL_write(ssl, "0", 1);
  return;
  }
    printf("Successfully Authenticated\n");
    SSL_write(ssl, "1", 1);
```

Client Code:

```
int client_auth(SSL *ssl, char* hostname)
{
    char buf[6000];
    char username[90];
    printf("Username : ");
    scanf("%s",username);
    SSL_write(ssl, username, strlen(username));
    int len;
    len = SSL_read (ssl, buf, sizeof(buf) - 1);
    if(buf[0] == '0') {
  printf("User account does not exist\n");
  exit(0);
    }
    char* pwd = getpass("Password: ");
    SSL_write(ssl, pwd, strlen(pwd));
    len = SSL_read (ssl, buf, sizeof(buf) - 1);
    if(buf[0] == '0') {
  printf("Incorrect password!\n");
  exit(0);
    }
}
```

Authorized user with correct password

```
[04/27/2020 04:59] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client rakshith2294.com 4433
[sudo] password for seed:
Connecting to server rakshith2294.com...
Verification passed.
Verification passed.
SSL connection using AES256-GCM-SHA384
Please enter your username and password
Username : seed
Password:
You are now connected to the VPN
MiniVPN Connection successful
```

```
[04/27/2020 04:58] Rakshith-10.0.2.22@VM:~/.../vpn$sudo ./tcp_server
Enter PEM pass phrase:
SSL connection established!
Connection request from client seed
Successfully Authenticated
```

Unauthorized user

```
[04/27/2020 05:02] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client rakshith2294.com 4433
Connecting to server rakshith2294.com...
Verification passed.
Verification passed.
SSL connection using AES256-GCM-SHA384
Please enter your username and password
Username : rak2294
User account does not exist
[04/27/2020 05:02] Rakshith-10.0.2.15@VM:~/vpn$
```

```
[04/27/2020 05:02] Rakshith-10.0.2.22@VM:~/.../vpn$sudo ./tcp_server
Enter PEM pass phrase:
SSL connection established!
Connection request from client rak2294
Request Terminated: User not found
```

Authorized user but incorrect password

```
[04/27/2020 05:03] Rakshith-10.0.2.15@VM:~/vpn$sudo ./tcp_client rakshith2294.com 4433
Connecting to server rakshith2294.com...
Verification passed.
Verification passed.
SSL connection using AES256-GCM-SHA384
Please enter your username and password
Username : seed
Password:
Incorrect password!
[04/27/2020 05:03] Rakshith-10.0.2.15@VM:~/vpn$
```

```
Connection request from client seed
Incorrect password
```