Workflow Description

The program implements a secure multi-threaded client-server forum system. The client uses UDP to transfer control information (such as logins, thread actions, and user commands) and TCP to transfer files (uploads and downloads). When TLS_1 mode is enabled, a secure handshake is initiated before normal communication begins. When Retransmission mode is enabled, UDP transmission control messages are transmitted reliably.

The handshake follows a simplified protocol similar to TLS 1.3. The client first sends a ClientHello message containing a temporary Diffie-Hellman public key. The server responds with its own temporary DH public key in the ServerHello message, along with a digital signature for the key and a chain of certificates (server, intermediate certificate, and root). The client then verifies the signatures of each certificate in the certificate chain, ensures that the root certificate is trusted by comparing it to the local CA pool, and confirms that the DH key signed by the server matches its certificate.

Once the certificate chain and signatures are verified, both parties compute the same shared key from the DH exchange. This key can then be used for simple symmetric encryption (e.g., XOR) to enable encrypted UDP message communication.

Once the security context is established, the client is authenticated using a username/password login sequence and can then issue commands such as create thread (CRT), post message (MSG), read (RDT), edit (EDT), and so on. File uploads (UPD) and downloads (DWN) are handled over TCP and coordinated by UDP messages that prepare and acknowledge the transfer.

Each client is processed in a separate thread on the server, and thread-safe locks are used to avoid contention conditions when accessing shared resources such as forum files and user credentials. The system supports concurrent clients and uses queues to separate UDP message reception from processing logic.

Unfinished Programs

The first is that TLS achieves forward security using the short-lived DH public key, but there is no way to achieve protection against replay attacks, again adding ACK sequence numbers and timestamps to the message solves this problem.

The second place is not add keep-alive, when the client does not use XIT exit (abnormal exit), the server can not detect the client has exited, the login will show the current user is online, so you can add keep-alive, the client regularly sends a light “heartbeat” message to the server The client sends a light “heartbeat” message to the server periodically to indicate that “I'm still alive”.

issues with the program

In Retransmission mode, the client and server use UDP to communicate and do not implement ACK determination. Retransmission after packet loss causes the server to repeat the client's commands, but the program can run reliably, and the client will respond correctly even though the client's commands may be different because of packet loss.

For example, when the client sends a CRT thread_1, the server did not reply due to packet loss, the client will send again, at this time, thread_1 has been created, so the server will respond to the thread_1 already exists, although different, but the purpose of the thread_1 reached, the solution is to add the ACK sequence number authentication.

Program Design

TLS_1

1. Server Preparation: DH public parameters: p=7919, g=2, Server Generated Certificate Chain: 1. RootCA (self-signed) 2. IntermediateCA (signed by RootCA) 3. MyServer (signed by IntermediateCA)

2. Client → Server: ClientHello | client_dh_pub

3. Server → Client: The server signs ServerHello + server_dh_pub with the RSA private key and returns the signature + certificate chain together.

4. Client authentication of server identity and public key

5. Both parties calculate the shared key

Login Phase (UDP)

1. Client sends username

2. Server returns "OK" (exists) or "New user"

3. Client sends password or new password

4. Server verifies and returns: Login successful or " Invalid password" / "already logged in" and so on.

Control commands（UDP）

1. Client uses parse_user_input() to construct command, calls udp_send() to send

2. If Retransmission = True is enabled, use timeout + retransmission mechanism to ensure server responds

3. Server maintains one thread per client, UDPThread, that parses commands and manipulates the file system

File transfer (UDP+TCP)

Upload (UPD) process:

1. Client UPD <thread> <file> send to UDP port

2. Server logs pending upload, returns "READY"

3. Client initiates TCP connection, sends file block by block

4. Server logs and sends sendto() to notify upload is successful

Download (DWN) process:

1. Client DWN <thread> sends to UDP server verifies existence, returns "READY" Client connects to TCP, downloads file, saves it locally. thread> <file> sends to UDP

2. server verifies existence, returns "READY"

3. client connects to TCP, downloads file, saves it locally.

Safety design

1. Multi-client support Independent, UDPThread per client

2. Message distribution client, queues[addr] = Queue()

3. File uploads/downloads Per connection, TCPThread handling, pending_downloads and pending_uploads

4. Data consistency Post locking: get_thread_lock(name)

5. Logon status online_users, online_addrs Manage

6. online_user_zeroed prompt Use online_zero_printed to control printing only once