Secure Communication System Protocol

Overview

This document describes the protocol used for secure communication between a client and server. The protocol ensures user authentication, message encryption, and integrity. It includes detailed steps for various commands and their expected responses.

startServer.sh which takes a port number as its only command-line parameter and attempts to start a server on that port (listening on all interfaces supported by the host system). If the server is unable to be started (perhaps because that port is already in use), the program will exit with an appropriate error message.

startClient.sh which takes a host name as its first command-line parameter and a port number as its second command-line parameter and attempts to connect to the server with the given host name and port number. If the client is unable to connect, it will exit with an appropriate error message.

Secure Server Setup

1. **SSL Configuration**:
    - The server uses SSL for secure communication.
    - The server certificate (**server.crt**) and key (**server.key**) are used for encryption.
2. **Database Configuration**:
    - SQLite is used to store user credentials and messages.
    - Tables:
        - **users**: Stores username, password hash, online status, and last online timestamp.
        - **messages**: Stores sender, receiver, send time, read time, message content, and notification status.
3. **Initialization**:
    - The server initializes the database tables if they do not exist using **create_db()**.

Commands and Responses

1. User Verification

Check Username

- **Command**: **CHECK <username>**
- **Response**:
    - **USERNAME ALREADY EXISTS**: If the username exists in the database.
    - **USERNAME NOT FOUND**: If the username does not exist.

2. User Authentication

Login

- **Command**: **LOGIN <username> <hashed_password>**
- **Response**:
    - On Success: JSON array of status, unread_messages and notifications
    - On Failure: **LOGIN FAILED: INCORRECT PASSWORD**

Create Account

- **Command**: **CREATE_ACCOUNT <username> <hashed_password>**
- **Response**:
    - **ACCOUNT CREATED**: On successful account creation.
    - **USERNAME ALREADY EXISTS**: If the username already exists.

3. Message Handling

Compose Message

- **Command**: **COMPOSE <sender> <receiver> <message>**
- **Response**:
    - **MESSAGE SENT SUCCESSFULLY.**: On successful message composition.
    - **There is some error in sending the message**: On failure.

Read Unread Messages

- **Command**: **READ_UNREAD <username>**
- **Response**:
    - On Success: JSON array of unread messages.
    - **No Messages Yet**: If no unread messages.

Read All Messages

- **Command**: **READ_ALL <username>**
- **Response**:
    - On Success: JSON array of all messages.
    - **No Messages Yet**: If no messages.

4. Notifications

View Notifications

- **Command**: **VIEW <username>**
- **Response**:
    - On Success: JSON array of notifications.
    - **No New Notifications**: If no new notifications.

5. Exit

Logout and Exit

- **Command**: **EXIT**
- **Response**:
    - None (Server closes the connection).

Client Operations

1. Login Process

1. Prompt user for username.
2. **Command**: **CHECK <username>**
3. **Response**:
    - If **USERNAME ALREADY EXISTS**:
        - Prompt for password.
        - Hash the password using SHA-256.
        - **Command**: **LOGIN <username> <hashed_password>**
        - Handle login response.
    - If **USERNAME NOT FOUND**:
        - Ask if the user wants to create a new account.
        - If yes, prompt for password, hash it, and send **CREATE_ACCOUNT** command.

2. Compose Message

1. **Command**: **COMPOSE <sender> <receiver> <message>**
2. **Response**: Handle the response from the server.

3. Read Messages

1. **Command**: **READ_ALL <username>** or **READ_UNREAD <username>**
2. **Response**: Display messages based on the server's response.

4. View Notifications

1. **Command**: **VIEW <username>**
2. **Response**: Display notifications based on the server's response.

5. Exit

1. **Command**: **EXIT**
2. The client logs out and exits the system.

Security Considerations

- **SSL Encryption**: Ensures secure communication between client and server.
- **Password Hashing**: SHA-256 is used to hash passwords before sending them over the network.
- **Database Security**: The server-side database ensures that only hashed passwords are stored.