Secure Chat Project Documentation

1. PROJECT OBJECTIVES

From what I understand from the directions, these are the main things that the project wants us to accomplish.

Step 1: Implement the Handshake Protocol

- The handshake establishes a secure connection between the client and server using the Diffie-Hellman key exchange. This allows both parties to generate a shared secret that is used as the basis for secure communication.

Step 2: Set Up Encryption and Message Integrity

- Messages exchanged between the client and server must be encrypted to maintain confidentiality. This involves using the shared secret to encrypt the message data. Additionally, messages should have integrity checks in place, typically using HMAC, to ensure that they haven't been tampered with.

Step 3: Implement Replay Attack Prevention

- Prevent replay attacks by implementing mechanisms like nonces or sequence numbers to ensure that intercepted messages can't be resent by attackers.

(Semi) Step 4: Be Aware of Other Security Issues

- Protect the program against common software vulnerabilities, such as buffer overflows, by carefully managing memory and implementing secure coding practices.

The project follows a structure similar to the SSH protocol to provide secure and authenticated communication.

WHAT IS COMPLETED ALREADY

A. Step 1: Handshake and Key Exchange

- I implemented the Diffie-Hellman Key Exchange, allowing the server and client to generate both long-term and ephemeral key pairs. These keys are exchanged between both parties, and a shared secret is derived using the key pairs. The shared secret forms the basis for the subsequent encryption and decryption of messages.
    - **You can find these changes in the following files/functions:** `initClientNet()` and `initServerNet()` functions in `chat.c`.
    - **Testing:** I added code to display the computed shared secret for both the client and server to ensure they match, confirming that both sides derive the same shared key. This code is still in the project but is just commented out.

B. Step 2: Encryption and Decryption

- I implemented encryption using AES in CTR mode, where the shared secret is used as the encryption key, and each message is accompanied by a randomly generated initialization vector (IV).
- I also implemented decryption for incoming messages, ensuring that they can be correctly decrypted on the receiving end.
    - **You can find these changes in the following files/functions:** `encryptAndSendMessage()` and `decryptMessage()` functions in `chat.c`.
    - **Testing:** I added print statements to output the original plaintext, the encrypted ciphertext (in hexadecimal), and the decrypted plaintext to verify the encryption and decryption process.

C. Step 2 (Partial): Message Integrity Check

- I added HMAC verification for each received message. This ensures that messages have not been altered during transmission by verifying the integrity of the message using HMAC.
    - **You can find these changes in the following files/functions:** `decryptMessage()` function in `chat.c` for HMAC verification.
    - **Testing:** I included print statements to display the generated HMAC, the received HMAC, and the computed HMAC to check if they match. (If they match, then we know an attacker didn't tamper with them.)

REMAINING TASKS

A. Step 2 (Unfinished): Complete Message Integrity Setup

- **Description:** I didn't actually completely finish step 2 of the project. There are a few additional improvements needed for the encryption and integrity setup:
    - **Secure Key Derivation:** Instead of directly using the shared secret for both encryption and HMAC, it's better to derive separate keys for these purposes using a Key Derivation Function (KDF).
        - **From what I understand, the "suggested Approach" is to:** Use a KDF like HKDF to derive separate keys from the shared secret—one for HMAC and one for AES encryption.
        - **Relevant files/functions:** I believe this should be implemented after computing the shared secret in `initClientNet()` and `initServerNet()` functions in the `chat.c` file.
    - **Message Structure Consistency:** Ensure that the structure for sending messages is clearly defined and followed throughout the code. This includes having a consistent order for IV, ciphertext, and HMAC when sending and receiving messages.

B. Step 3: Implement Replay Attack Prevention

- **Description:** Replay attacks occur when an intercepted message is resent to trick the recipient. To prevent this, we need a mechanism to detect and reject replayed messages.
    - **From what I understand, the suggested Approach is:** Use nonces (randomly generated values that are never reused) or sequence numbers for each message. The receiver should track these values to detect and reject duplicate messages.
    - **Relevant files/functions:** I'm pretty sure this is supposed to be implemented in the `encryptAndSendMessage()` and `decryptMessage()` functions in `chat.c`.

C. (Semi) Step 4: Be Aware of Buffer Overflow Protection

- **Ensure that the code is secure from buffer overflows and other memory-related issues.**
    - **Suggested Approach:** Review all memory allocations and string operations to ensure proper bounds checking and handling of dynamic memory.
    - **If there is a problem with these types of issues, I'm pretty sure it would be in these functions at the moment:** `recvMsg()`, `encryptAndSendMessage()`, and `decryptMessage()` in the `chat.c` file.

NOTES

- All test code is already put in the correct spots if you want to test them yourself to see what happens, you just have to uncomment them back in.
- The text chat might be a little buggy at the moment. Sometimes the 2nd and 3rd messages display weirdly.
- I already removed the character limit of the text chat.
- We didn't use Triple Diffie-Hellman, or at least not yet.
- We don't have implicit or deniable authentication yet.
- I tried to handle the problems of partial reads and writes by using the functions the professor gave in `util.h`. But I'm not sure if that completely solved it.