FSCT 8561 – Lab 1: Socket Programming – Stateful Client-Server Chat Application

By: Jose Bangate, A01271709

Part 5 – Reflection Questions

1. If the server crashes while the client is still connected, the client side will return an error message if a user sends a message.
2. The server can handle multiple clients if there is threading in the server itself.
3. Yes, recv(1024) can split messages unexpectedly. This is because TCP is a stream-oriented protocol.
4. This can be implemented by requiring a username and password, which the server verifies before accepting chat messages.

Part 6 – Security Analysis

Because of its unique protocol design, session state management, and dependence on raw TCP sockets, the implemented chat application presents a number of security risks. The code exposes several attack surfaces that would be problematic in a real-world deployment, even though it is suitable for educational purposes.

The application-level message protocol (***HELLO|***, ***MSG|***, ***EXIT|***) is a major attack surface. Any client can pretend to be another user by just selecting the same username because the server trusts the client to self-identify using ***HELLO|username***. There is no authentication method to confirm identification, so the server stores the username in session state (***username = “”***) until ***HELLO*** is received. Additionally, an attacker can continually login, select any username, and send messages without any restrictions because the server just verifies if ***HELLO*** was sent initially.

Abuse scenarios are also introduced by the permanent connection concept. Because each client connection is managed by a dedicated thread, an attacker might open numerous connections at once and use up all of the server’s resources. The server is vulnerable to connection flooding and message spamming because there is no rate limitation, timeout, or maximum connection count. Additionally, since ***recv(1024)*** reads from a TCP stream, the server’s buffering logic may be stressed, or unexpected behaviour may be triggered by incorrect or intentionally fragmented data.

TCP itself provides no security guarantees beyond reliable, ordered delivery. The application sends all data in plain text, meaning usernames and chat messages can be intercepted or modified by anyone with network access. TCP does not provide encryption, authentication, or integrity protection. Therefore, attackers could eavesdrop on conversations, inject messages, or hijack sessions using man-in-the-middle techniques.

For a real deployment, several mitigations are necessary. First, authentication should be enforced using credentials (username and password) verified server-side, preferably with hashed passwords. Second, encryption should be added using TLS to protect confidentiality and integrity of messages in transit. Third, the server should implement rate limiting, connection limits, and inactivity timeouts to reduce denial-of-service risks. Finally, stronger input validation and structured error handling would help prevent malformed input from destabilizing the server.

Overall, while the code correctly demonstrates protocol enforcement and session state, it lacks essential security controls required for safe real-world use.