Sam - Group 4 Peer Review

Group 4 has made a solid start in developing the core structure of the chat application, successfully setting up the WebSocket server and implemented a frontend GUI. These elements are crucial for facilitating basic communication between clients and servers.

Despite these foundational strengths, the current implementation appears to be incomplete and exhibits several critical security shortcomings. Key features such as robust message signing, comprehensive encryption, and proper authentication mechanisms are either missing or inadequately addressed. Enhancing these areas will significantly improve the application's security posture and ensure adherence to the OLAF/Neighbourhood protocol. Detailed observations and suggestions are provided in the subsequent sections to guide your development efforts.

Reviewed By:

- Samuel Chau (a1799298)

1. Manual Code Review

Architecture and Design

Aspect	Status	Comments
Protocol Implementation Adherence	Not Adhered	The provided code does not implement the OLAF/Neighbourhood protocol as specified. Key protocol features such as message signing, encryption, fingerprint verification, and proper routing through neighborhood servers are missing or improperly implemented. For example, the `handle_chat` function does not handle end-to-end encryption as outlined in the protocol. Additionally, the client-server communication does not follow the defined JSON structures for different message types.
Security Measures	Partially Implemented	Some basic security measures are present, such as the use of WebSockets for communication. However, critical security features like proper authentication and encryption are either missing or incorrectly implemented. The `vulnerable_authentication` function is a placeholder and does not follow secure practices. Moreover, the use of plaintext messages in public chats contradicts the protocol's emphasis on securing communications.

Code Quality

Aspect	Status	Comments
Readability and Organization	Poor Organization	The code lacks proper structure and modularization. Functions are scattered without clear separation between different components of the system. Additionally, comments indicate placeholder implementations, such as in `encrypt_message_cpp` and `decrypt_message_cpp`, suggesting that the code may have been heavily AI-generated without proper modification.

Aspect	Status	Comments
Error Handling and Logging	Inadequate	Error handling is minimal and inconsistent. For instance, the `handle_chat` function sends a generic error message if the recipient is not connected but does not handle other potential errors gracefully. Logging is limited to basic `print` statements, which are insufficient for debugging and monitoring in a production environment. Comprehensive logging mechanisms are absent, making it difficult to trace issues or monitor system behavior effectively.

Security-specific Checks

Aspect	Status	Comments
Input Validation	Insufficient	There is minimal input validation throughout the code. For example, the `vulnerable_authentication` function does not properly sanitize or validate user inputs, making it susceptible to injection attacks. Additionally, message handling functions do not thoroughly validate the structure and content of incoming messages, increasing the risk of processing malformed or malicious data.
Access Control	Missing	The code does not implement robust access control mechanisms. Users are not properly authenticated or authorized to perform actions, allowing potential unauthorized access. The `vulnerable_authentication` function uses hardcoded credentials, which can be easily bypassed, and there are no role-based access controls to restrict functionalities based on user privileges.
Cryptographic Implementations	Incorrect Implementation	The cryptographic functions `encrypt_message_cpp` and `decrypt_message_cpp` are placeholders and do not implement the specified encryption protocols (RSA, AES-GCM). Additionally, the use of subprocess calls for encryption is insecure and opens avenues for exploitation. The protocol specifies detailed encryption and signing mechanisms which are not adhered to in the code.
Secure Data Storage and Transmission	Insecure Transmission	While WebSockets are used for communication, there is no implementation of secure WebSockets (`wss://`). Data transmitted is not adequately encrypted, especially in public chats where messages are sent in plaintext. The protocol emphasizes end-to-end encryption and secure transmission channels, which are not implemented in the current codebase.

2. Static Analysis

The static analysis was performed using **Bandit**, a security-oriented static analysis tool for Python. Below are the findings from the analysis of `chat_server.py`:

Issue	Severity	Confidence	Location	Description	Recommendation	More Info
B404	Low	High	chat_server.py:4	Usage of the `subprocess` module can lead to security vulnerabilities if not handled properly.	Avoid using subprocess with untrusted input or ensure that all inputs are properly sanitized.	Link
B603	Low	High	chat_server.py:71	Subprocess calls without `shell=True` can be exploited if untrusted input is passed.	Avoid using `subprocess.run` with untrusted input or use safer alternatives. If `shell=True` is necessary, ensure that the input is thoroughly sanitized.	Link
B603	Low	High	chat_server.py:76	Same as above.	Same as above.	Link

3. Dynamic Analysis

The provided code is unfinished and poorly documented, so dynamic analysis was not suitable for this review. The reviewer did attempt to get the application running and send messages, however, most of the protocol remains in a placeholder state so nothing substantial can be said about the application.

4. Backdoor/Vulnerability Assessment

Note that the following vulnerabilities are only *theoretically* possible because the application seems mostly unfinished and doesn't actually implement the protocol. The reviewer has written this section under the implication that these functions exist and operate in a finished implementation.

Vulnerability	Location	Description	Example
Subprocess Command Execution	`encrypt_message_cpp` and `decrypt_message_cpp` functions.	These functions utilize `subprocess.run` to execute external encryption/decryption programs. This allows for the execution of arbitrary commands if user input is not properly sanitized.	A malicious user could send a specially crafted message that includes shell commands, potentially leading to remote code execution. For instance, sending a message like `"; rm -rf / #` could exploit the subprocess call to delete critical server files.
Vulnerable Authentication Mechanism	`vulnerable_authentication` function.	The authentication function uses hardcoded credentials (`username == "admin" && password == "password"`), making it trivial for attackers to gain unauthorized access.	An attacker can easily bypass authentication by using the known credentials, gaining access to administrative functionalities or sensitive data. This could *potentially* allow the attacker to send fraudulent messages, manipulate user lists, or disrupt the chat service.

5. Results Summary

Category	Details
Strengths	- **Basic Functionality:** The code includes initial setups like a WebSocket server and basic message handling functions.
Areas for Improvement	- **Protocol Non-Adherence:** Does not implement the OLAF/Neighbourhood protocol, missing essential features like message signing and proper encryption. - **Security Shortcomings:** Lacks proper authentication, encryption, and input validation; uses hardcoded credentials and insecure subprocess calls. - **Code Organization:** Poor structure with scattered functions and placeholder implementations; lacks modularity and clear separation of concerns. - **Error Handling:** Minimal and inconsistent error handling; insufficient logging for debugging and monitoring.
Critical Issues	- **Insecure Authentication:** Hardcoded credentials in `vulnerable_authentication` function pose a high security risk. - **Unsafe Subprocess Usage:** `encrypt_message_cpp` and `decrypt_message_cpp` functions use subprocess calls that can be exploited for code injection.

6. Recommendations

Recommendation Category	Actions
Implement Protocol Specifications	- Fully adhere to the OLAF/Neighbourhood protocol, including message signing, end-to-end encryption with RSA and AES-GCM, and correct message routing.
Enhance Security Measures	- Replace hardcoded credentials with a secure authentication system using hashed and salted passwords. - Sanitize and validate all user inputs to prevent injection attacks. - Secure subprocess calls or replace them with safe, integrated cryptographic functions. - Use secure WebSocket connections (`wss://`) to encrypt data in transit.
Improve Code Quality	- Refactor the codebase for better organization, separating concerns into modules or classes. - Remove placeholder functions and implement fully functional features. - Introduce robust error handling and comprehensive logging mechanisms.