Sam - Group 5 Peer Review

This implementation stands out with its impressive adherence to the OLAF/Neighbourhood protocol and the incorporation of a functional graphical user interface, which greatly enhances user experience. Core features such as messaging, file transfer, and client list management are well-executed, demonstrating a thorough understanding of the protocol's requirements and effective application of its specifications.

However, a (small) critical security vulnerability has been identified concerning access control, where clients can impersonate servers by sending unauthorized `server_hello` messages. This flaw poses a significant risk to the application's integrity and security. Addressing this vulnerability, along with improving input validation and cryptographic practices, will be essential in fortifying the system against potential attacks. The following sections provide a comprehensive analysis and actionable recommendations to help mitigate these issues.

Reviewed By:

- Samuel Chau (a1799298)

1. Manual Code Review

Architecture and Design

Aspect	Status	Comments
Protocol Implementation Adherence	Mostly Adhered	The code attempts to implement the OLAF/Neighbourhood protocol and handles specified message types such as `hello`, `chat`, `public_chat`, `client_list_request`, and `client_update`. However, there is a discrepancy in the `client_update` message format. The code sends the `clients` field as a list of base64-encoded PEM strings, whereas the protocol specifies it should be a list of PEM-formatted public keys without base64 encoding. This inconsistency may lead to interoperability issues with other implementations strictly following the protocol. Further verification during dynamic analysis will be conducted to verify. However, this is a minor issue and adherence is otherwise exemplary.

Code Quality

Aspect	Status	Comments
Readability and Organization	Needs Refactoring	The code is generally organized into functions and classes, making it somewhat readable. However, there are areas where refactoring could improve clarity and maintainability. Variable names are sometimes non-descriptive, and there is a lack of inline comments explaining complex sections, which may hinder understanding for new developers.

Aspect	Status	Comments
Error Handling and Logging	Well integrated	Error handling is present and correctly captures most communication across servers and clients. Most logs seem to be development focused - i.e. message arrived here, sending here, etc. While this is generally sufficient, a more robust and verbose logging scheme could be beneficial for production. Moreover, exceptions are occasionally caught and printed, but not always handled appropriately or propagated.

Security-specific Checks

Aspect	Status	Comments
Input Validation	Potential Vulnerability	Input validation is minimal throughout the code. In functions like `process_message`, the code assumes that incoming data is well-formed without thorough validation of message structures or contents. This lack of validation could lead to the processing of malformed or malicious data, potentially causing unexpected behavior or security vulnerabilities.
Access Control	Potential Backdoor	The `is_server` variable in the `ClientSession` class poses a security risk. Any client can send a `server_hello` message, causing their session to be marked as `is_server = True`, which bypasses signature verification in subsequent messages. This could allow an attacker to send unsigned or malicious messages without proper authentication. The code lacks robust mechanisms to authenticate and verify the identities of servers versus clients, compromising access control.
Cryptographic Implementations	Incorrect Implementation	The cryptographic implementations do not align with the protocol specifications. For signing, the code uses RSA with PKCS1v15 padding instead of the required RSA-PSS with SHA-256. For symmetric encryption, AES in CBC mode with PKCS7 padding is used, whereas the protocol specifies AES in GCM mode.
Secure Data Storage and Transmission	Insecure Transmission	Data transmission is not secured at the transport layer. The server communicates over plain TCP sockets without TLS encryption, and the Flask application for file uploads/downloads operates over HTTP rather than HTTPS. This exposes sensitive data to potential interception and eavesdropping. The lack of secure channels undermines the application's security measures.

Note: While this review focuses on areas for improvement from a security perspective, it's important to acknowledge the overall quality of this implementation. The team has successfully created a functional system that adheres to most aspects of the OLAF/Neighbourhood protocol. The identified issues provide opportunities for enhancement, but they should not overshadow the considerable effort demonstrated so far!

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:

server.py

Issue ID	Severity	Confidence	Location	Description	Recommendation	More Info
B201	High	Medium	server.py:429	A Flask app appears to be run with `debug=True`, which exposes the Werkzeug debugger and allows the execution of arbitrary code.	Disable debug mode in production by setting `debug=False`. Ensure that the Flask app is not run with debug mode enabled in any deployment or production environment.	Link

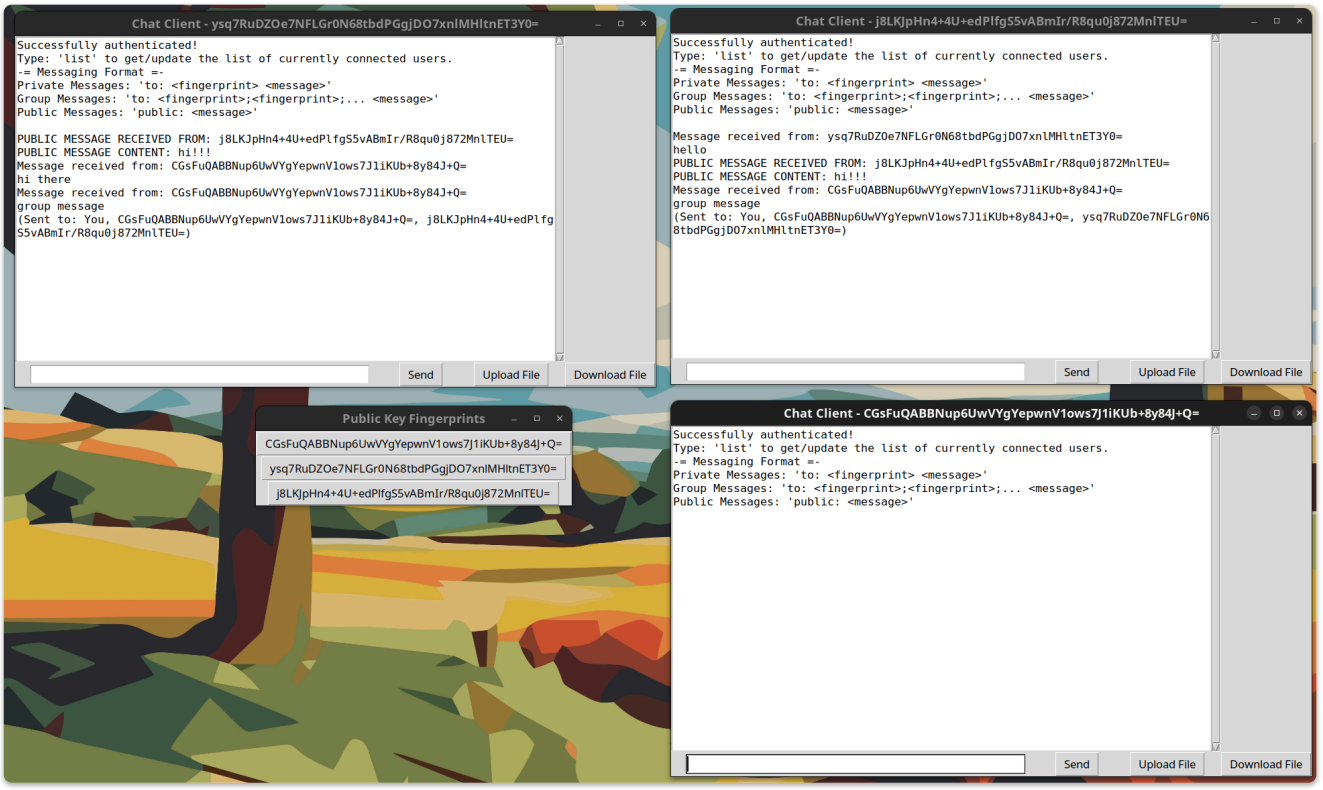
client.py

No issues identified.

3. Dynamic Analysis

Functional Testing

Aspect	Observation	Comments
GUI	Excellent	The Ttinker GUI is superb. Instructions are clear, and the overall interface looks very professional.
Client List	Room for improvement	The client list could be integrated into the main window for ease of use.
Messaging	Functional	Private, group, and public messages appear to be working as intended.
Protocol Adherence	Deviation noted	Logs indicate that client lists are being encoded in base64 rather than raw PEM keys before being sent, which deviates from the protocol specification.



Sample Log Output


```
Received message: {'type': 'client_list_request'}
client_list_request
I HAVE RECEIVED A REQUEST FROM A CLIENT FOR A LIST
{"type": "client_list", "servers": [{"address": "127.0.0.1:12346", "clients":
["LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSU
lCQ2dLQ0FRRUExLzhCNHBXVWc1NSt6cllKL1Ruagp6cG5uNkRtRER4enkzMk9uOGEydCtoeHV0ekY1bVYrKz
hkSnMwVWxlVzZyS0M4cHRBcEFqSmRKNDUxSDRGakhjCm9RRWVQaHNYNEcwU2lXVnV4d2cwdEY5Y3UzYzlMUk
VYU3ZkN1JsSGVUNEdydjVEYUt2UFlFQ3c5VFluTW5EdS8KdUZHREVETjBwYzB4OHR3MHNjY0VYd0k3OHJ5Um
Z0S1lFYlE3RURSTER2eVczQlZIdCtwczhqN0R3L250dk9abQpWS3lNU2lycytaSkoyRFEvM1FjL2RIK2dtWG
5EVitwNmpUd0tLVHZCYlBZNTFraEhUa1BqQ2MrL0F5Y2pTb1dBCkMzOEJaeW9DbmVLTVBsQVZoeWd3ZUs1Zk
9aZ1NEVExPZFRlYklCLy9JdTBWVlMvdEZoT2RZeUFDOHk2QjJpdjUKZVFJREFRQUIKLS0tLS1FTkQgUFVCTE
lDIEtFWS0tLS0tCg==
"]}, {"address": "127.0.0.1:12345", "clients":
["LS0tLS1CRUdJTiBQVUJMSUMgS0VZLS0tLS0KTUlJQklqQU5CZ2txaGtpRzl3MEJBUUVGQUFPQ0FROEFNSU
lCQ2dLQ0FRRUFscWRqMGFFTVNvZkloeW0zeWpyRQp6bTlydUY2Qis2QmMzRWNjdElUUEQzRUlsblZsa3Z5M3
JqcEcvandNUWpPdWpnUTBlcFI4WHZXeHltNkhEUXJyClBxR2RXZzhnbGZ4YnBvN0g5MWVQL29ud3FCQVVUWT
IyTVh4OWdkSU1XL3BrOEJpMU1PNkkwQlU2Zm45enIyRjQKV1h5R1FDU2VuYnV2dWtGbnVINDcrNkdtOUQ2V3
ZBVUFVR1lLOURWN09IQnJqclVQMW1iTk1seHZsVFZhQmV5RgplZ1IvSFV1dVJjRDFLMG51VGtKZWNKTXpRSm
8vUTVBVmszNFdQT3FXYVV1UWlBNUxONWpoZm9JK0xFeVdBS05iCmszVXdlbUVhNHRhbHc5REFGdytqL1V3b1
hMU01oOUgyNXlwb2JzN2MwUkttaFBwTnk5ZGxDeEJBbGRWSTcrSHUKb1FJREFRQUIKLS0tLS1FTkQgUFVCTE
lDIEtFWS0tLS0tCg==
"]}]}
```

Security Testing

Test	Method	Result	Implications
Access Control Bypass	Netcat	Successful	Critical vulnerability allowing unauthorized access and message broadcasting

Penetration Testing Steps:

1. Setting `is_server` to `True`:

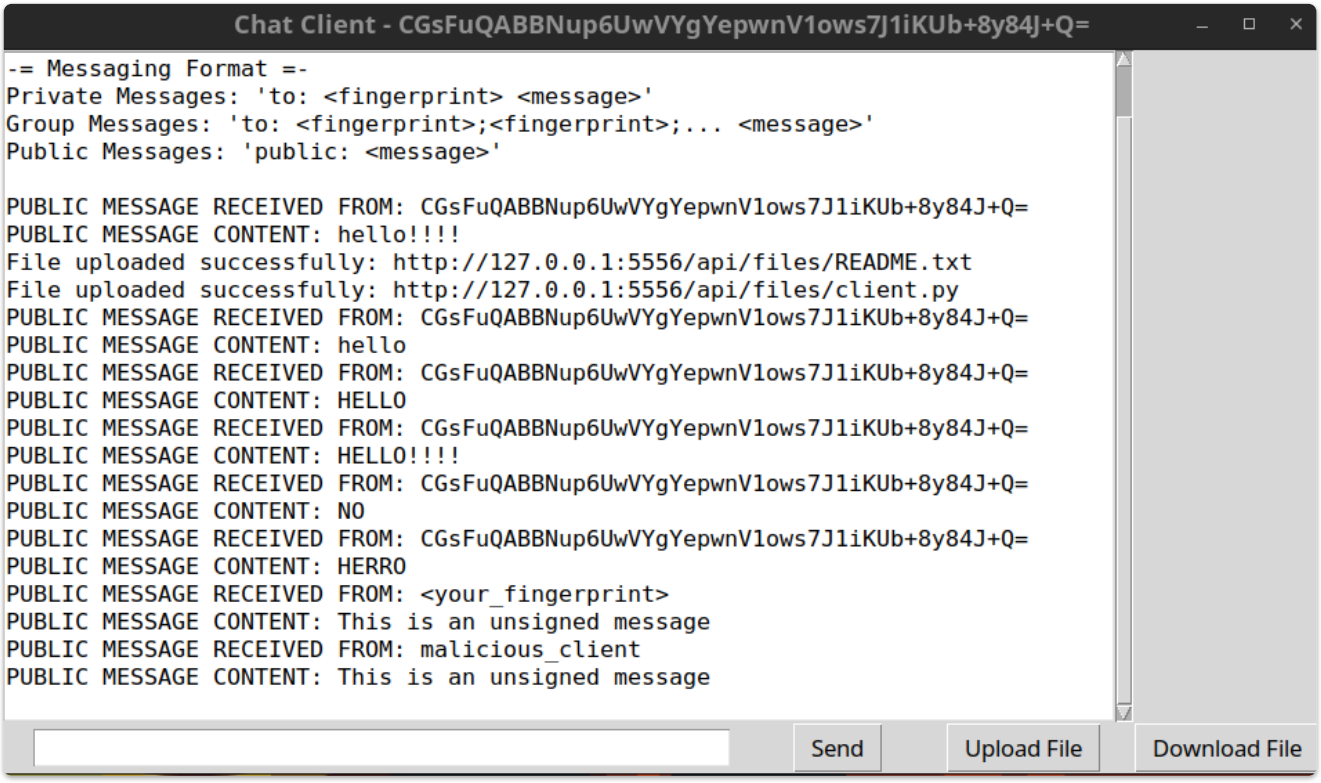
```
echo '{"type":"server_hello","sender":"127.0.0.1:12347"}' | nc 127.0.0.1 12345
```

2. Sending an unsigned `public_chat` message:

```
echo '{"type":"signed_data","data":
{"type":"public_chat","sender":"malicious_client","message":"This is an unsigned
message"},"counter":1,"signature":""}' | nc 127.0.0.1 12345
```

Vulnerability Impact:

The server accepted and broadcast the unsigned message to other clients without verifying the signature, even when no client application was open. This demonstrates a critical vulnerability where a malicious actor can impersonate a server and send unauthorized messages.



4. Backdoor/Vulnerability Assessment

Identified Intentional Backdoors:

Backdoor	Description	Impact
Server Spoofing via `server_hello` Messages	Any client can impersonate a server by sending a `server_hello` message. This sets the `is_server` flag to `True` in their session, allowing them to bypass signature verification for subsequent messages.	Attackers can send unsigned or malicious messages, impersonate servers, and disrupt secure communication channels.

Additional Observations:

After an exhaustive review of the provided code, no further intentional backdoors could be found.

5. Results Summary

Strengths:

Aspect	Details
Functional Implementation	- The application implements core functionalities like messaging, file transfer, and a graphical user interface. - Adherence to almost all aspects of the OLAF/Neighbourhood protocol
User Interface	- The Tkinter GUI is user-friendly and provides clear instructions, enhancing the overall user experience.

Areas for Improvement:

Aspect	Details
Security Measures	- Inadequate input validation and error handling.
Code Quality	- The code could benefit from better organization and more descriptive variable names. - Additional comments and documentation would improve readability and maintainability.

Security Enhancements:

Recommendation	Details
1. Implement Strict Authentication and Authorization	- For Servers: - Review the use of `is_server` flag via messages.
2. Enhance Cryptographic Practices	- Use RSA-PSS with SHA-256 for digital signatures, as specified by the protocol. - Switch to AES in GCM mode for symmetric encryption to provide both confidentiality and integrity. - Ensure that all cryptographic keys and operations follow best practices to prevent vulnerabilities.
3. Secure the Flask Application	- Disable debug mode by setting `debug=False` - Implement authentication for file upload and download endpoints to prevent unauthorized access.
4. Improve Input Validation and Error Handling	- Validate all incoming data on both client and server sides to ensure it conforms to expected formats. - Implement comprehensive error handling to manage exceptions gracefully without exposing sensitive information.

Code Quality Improvements:

Recommendation	Details
Refactor Code for Clarity and Maintainability	- Use descriptive variable and function names to enhance readability. - Organize code into modules or classes where appropriate. - Add comments and documentation to explain complex sections.
Implement Robust Logging Mechanisms	- Utilize Python's `logging` library to log events with appropriate severity levels. - Ensure that logs do not contain sensitive information that could be exploited.
Enhance Exception Handling	- Catch and handle specific exceptions rather than using broad exception handlers. - Provide meaningful error messages to users and log technical details for developers.