Peer Review for Yong Yue Beh

By Menno Brandt (a1849852)

Manual Code Review:

Adherence and Implementation of the Protocol:

After conducting a manual review, I would say that this protocol mostly adheres to the protocol and its requirements.
Key areas that are satisfied are as follows:
- The implementation uses a WebSocket, client-server model. The same architecture that's detailed in the protocol.
- It does file transfer, through HTTP.
- It incorporates the different message types, such as 'hello', 'public_chat', 'private_chat', etc.
- RSA is used for asymmetric encryption, and signing. AES is used for symmetric encryption of messages. Cryptography is correct.
- There's a counter for messages, to prevent replay attacks.

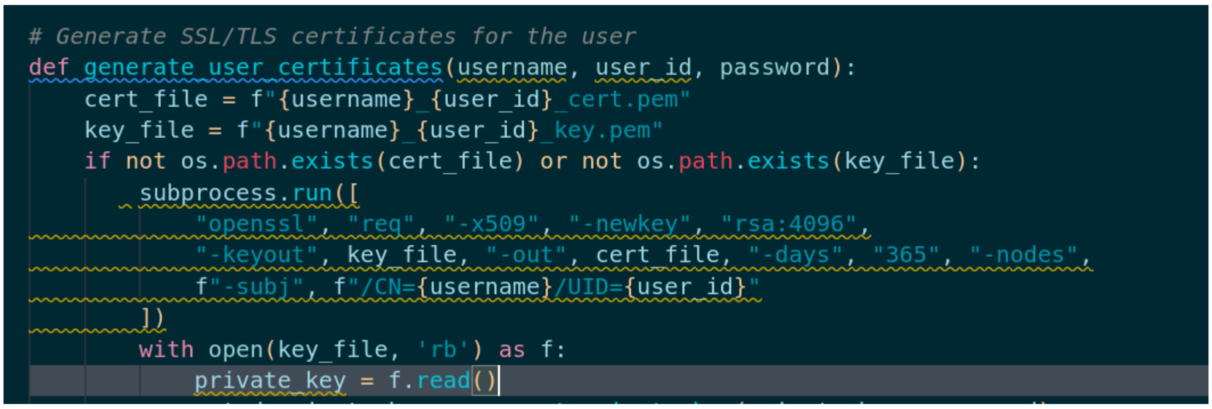
Although it's close, there are still some deviations. For example:
- The 'client_list_request' being called 'list_members'.
- The neighbourhood idea isn't entirely implemented. I don't think that the server can connect to others.

Static Analysis:

Pylint

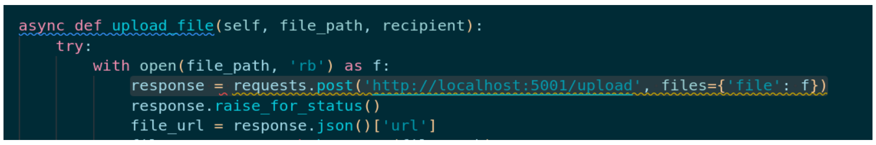
After linting all of the Python files in the implementation with Pylint, I tried identifying some of the most common "Warnings" and "Infos". They are as follows:
- W0621 (redefined outer name). This happens, in essence, when you try and pass a variable into a function by using the same. For example, passing in variable_a into a function as variable_a. This difference in scope and the redefinition of a name can (potentially) cause unexpected behaviour.
- W0603 (global statement). This one was a result of the "global counter", being used throughout the code. Although global variables aren't standard practice, and Pylint warns you against it, I myself struggle to think of an alternative solution to implementing that counter.
- W0718 (broad exception caught). This happens when the exception handling is not able to handle different kinds of exceptions. Instead, it treats them all as the same.

Aside from these common warnings, there were also some outliers.
- For example, a W1510 (subprocess run check) within client.py. This is the result, of subprocess.run being used, without specifying the value of "check".



```
# Generate SSL/TLS certificates for the user
def generate_user_certificates(username, user_id, password):
    cert_file = f"{username}_{user_id}_cert.pem"
    key_file = f"{username}_{user_id}_key.pem"
    if not os.path.exists(cert_file) or not os.path.exists(key_file):
        subprocess.run([
            "openssl", "req", "-x509", "-newkey", "rsa:4096",
            "-keyout", key_file, "-out", cert_file, "-days", "365", "-nodes",
            f"-subj", f"/CN={username}/UID={user_id}"
        ])
        with open(key_file, 'rb') as f:
            private_key = f.read()
```

- A timeout is also missing within this HTTP request. Resulting in a W3101.



```
async def upload_file(self, file_path, recipient):
    try:
        with open(file_path, 'rb') as f:
            response = requests.post('http://localhost:5001/upload', files={'file': f})
            response.raise_for_status()
            file_url = response.json()['url']
```

Peer Review for Yong Yue Beh

By Menno Brandt (a1849852)

Bandit

Along with Pylint, I also used Bandit. This was for the sake of scanning for vulnerabilities and security flaws. These came up less frequently, and the following were found:

- B113 (requests calls without timeout). This was in both the receive_file and upload_file functions.
- B607 (starting a process with a partial executable path). This was in the generate_ssl_certificates function.
- B603 (subprocess call, check for execution of untrusted input). This vulnerability was also in the generate_ssl_certificates function.

(Potential) Backdoors and Vulnerabilities

1. The server generates self-signed certificates, and certificate verification is disabled on the client side. I think this can lead to a man-in-the-middle attack. It wasn't something I could figure out how to tangibly exploit.
2. Input sanitisation is done with a sanitize_input function. I feel it's likely that it's not enough, to prevent injection attacks. Along with this, some user input is also not being thoroughly validated before being used.
3. For the SSL Key encryption, a hardcoded password is used. "generate_ssl_certificates("your_password_here")".
4. There are no HTTP timeouts, so you could perhaps exhaust all the resources.

Recommendations:

My recommendations for improving your implementation, are the following:

- Enhance input sanitisation and validation. Through making sure that central function is air tight, and that everything passes through it.
- Add timeouts to all HTTP requests.
- Use full paths for executables in subprocess calls and sanitize any user input.
- Add more error handling, and more specificity to exception handling.
- Implement the complete neighbourhood idea, for connecting servers with eachother.