# Peer Review for Hong Duc Vu

*By Menno Brandt (a1849852)*

## Manual Code Review:

### Adherence and Implementation of the Protocol:

This implementation was presented in a very compact, and structured manner. It consisted of three files (client.py, server.py, README.md), and their macOS compatible counterparts in another folder. The variable and method names were descriptive, and easy to understand. Comments were also clear, and helpful to gaining a deeper understanding.

Through my manual code review however, I do not think that this implementation adhered to the required Protocol.

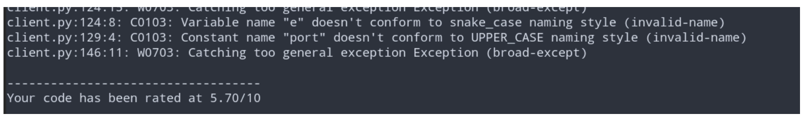
My reasons for this, are as follows:

- A ‘signed_data’ type and ‘hello’ are implemented in the code, but other required message types (such as ‘chat’, ‘public_chat’, ‘client_list_request’, etc) are not.
- Some important features such as file transfer, public chat, and client list request/response are missing from the implementation.
- The code uses RSA and can do message signing, but there is no use of AES to encrypt that message content and make the protocol’s end-to-end encryption work.
- The network topology and method of server discovery do not align with the protocol. It seems like a simplified client-server architecture that is not compatible with the neighbourhoods idea.

## Static Analysis:

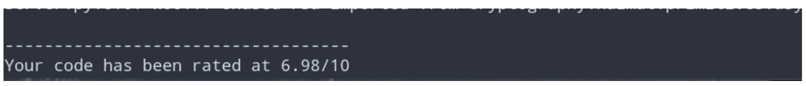
### Pylint

To do linting and identify potential errors in the code, I used Pylint. I first ran it directly from the terminal, on both of the files. The scores given for client.py (top) and server.py (bottom).



```
client.py:124:8: C0103: Variable name "e" doesn't conform to snake_case naming style (invalid-name)
client.py:129:4: C0103: Constant name "port" doesn't conform to UPPER_CASE naming style (invalid-name)
client.py:146:11: W0703: Catching too general exception Exception (broad-except)

-----------------------------------
Your code has been rated at 5.70/10
```



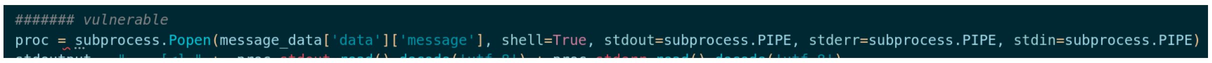
```
-----------------------------------
Your code has been rated at 6.98/10
```

Unsure how to interpret the scores, I used the Pylint VS Code extension. Between the two files, Pylint identified 0 Mistakes, 27 Warnings, and 41 Infos. The two most common warnings that I saw were “W0621:redefined-outer-name”, and “W0718:broad-exception-caught”. The first warning is basically caused when a person wants to use ‘variable_a’ in a function, but they also name the function’s argument ‘variable_a’. That difference redefinition and the difference in scope, could maybe cause unpredictable behaviour.

The second warning is about how exceptions are handled. My understanding is that you would have to try and catch more specific exceptions, to get rid of such warnings. The current code does this handling through a “try:”, and then a broad “except:”. But adding extra “”except:” conditions, and making them behave differently, would improve things.

### Bandit

To find potential security issues, I used the tool Bandit. It found two errors, within server.py. The first error was B602 (subprocess call with shell=True), and the line that causes this is shown in the screenshot below. The reason that this is a security issue, is because the user input is not being escaped. Therefore, command injection attacks, and arbitrary code execution are possible. To fix this, you could maybe escape the input with something like shlex.quote(), or use a list of arguments when calling those subprocess functions (as opposed to a string).



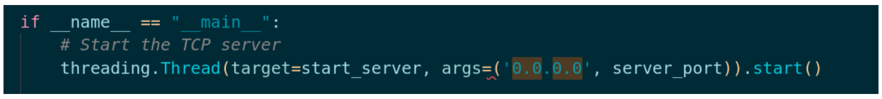
```
####### vulnerable
proc = subprocess.Popen(message_data['data']['message'], shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
```

(continued on next page)

# Peer Review for Hong Duc Vu

*By Menno Brandt (a1849852)*

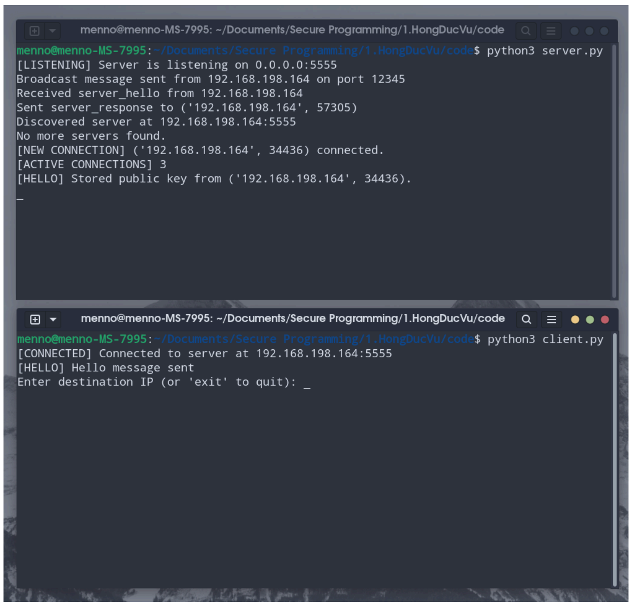
The second error identified by Bandit, was B104 (possible binding to all interfaces). This happened, because a socket was bound to all network interfaces, when 0.0.0.0 was used. This creates a security issue, because it exposes the app to networks it's not supposed to be accessible from. Below is a screenshot of the cause of that error:



```
if __name__ == "__main__":
    # Start the TCP server
    threading.Thread(target=start_server, args=('0.0.0.0', server_port)).start()
```

## Dynamic Analysis:

I began by installing the dependency "cryptography" library. Then I ran the server, and a single client.



menno@menno-MS-7995: ~/Documents/Secure Programming/1.HongDucVu/code

```
menno@menno-MS-7995:~/Documents/Secure Programming/1.HongDucVu/code$ python3 server.py
[LISTENING] Server is listening on 0.0.0.0:5555
Broadcast message sent from 192.168.198.164 on port 12345
Received server_hello from 192.168.198.164
Sent server_response to ('192.168.198.164', 57305)
Discovered server at 192.168.198.164:5555
No more servers found.
[NEW CONNECTION] ('192.168.198.164', 34436) connected.
[ACTIVE CONNECTIONS] 3
[HELLO] Stored public key from ('192.168.198.164', 34436).
_
```

menno@menno-MS-7995: ~/Documents/Secure Programming/1.HongDucVu/code

```
menno@menno-MS-7995:~/Documents/Secure Programming/1.HongDucVu/code$ python3 client.py
[CONNECTED] Connected to server at 192.168.198.164:5555
[HELLO] Hello message sent
Enter destination IP (or 'exit' to quit): _
```

After this, I ran multiple clients in different terminal windows. When it asked for the destination IP address, I simply supplied the same one and it would distinguish through their port numbers. I found that I was able to connect and send a message one way (Client 1 -> Client 2). But if an attempt was made for Client 2 -> Client 1, it would not work. Because the code uses threads, it could be a race condition. Or the client distinction mechanism does not work properly. But it seems that only unidirectional communication works.

## Recommendations and Conclusion:

The recommendations that I would make to the implementation, are the following:

- Try to increase compliance with the protocol.
- This can be done through: Making sure your underlying architecture is in alignment with the idea of the protocol and Neighbourhoods.
- And also adding missing message types, AES encryption, file transfer, public chat, client list, and other functionality.
- Another recommendation is to fix necessary Pylint warnings, and improve exception handling by making it respond in a more pointed way.