Javier Nieves & Nicholas Redmond

Professor Zhang

COMP 3825

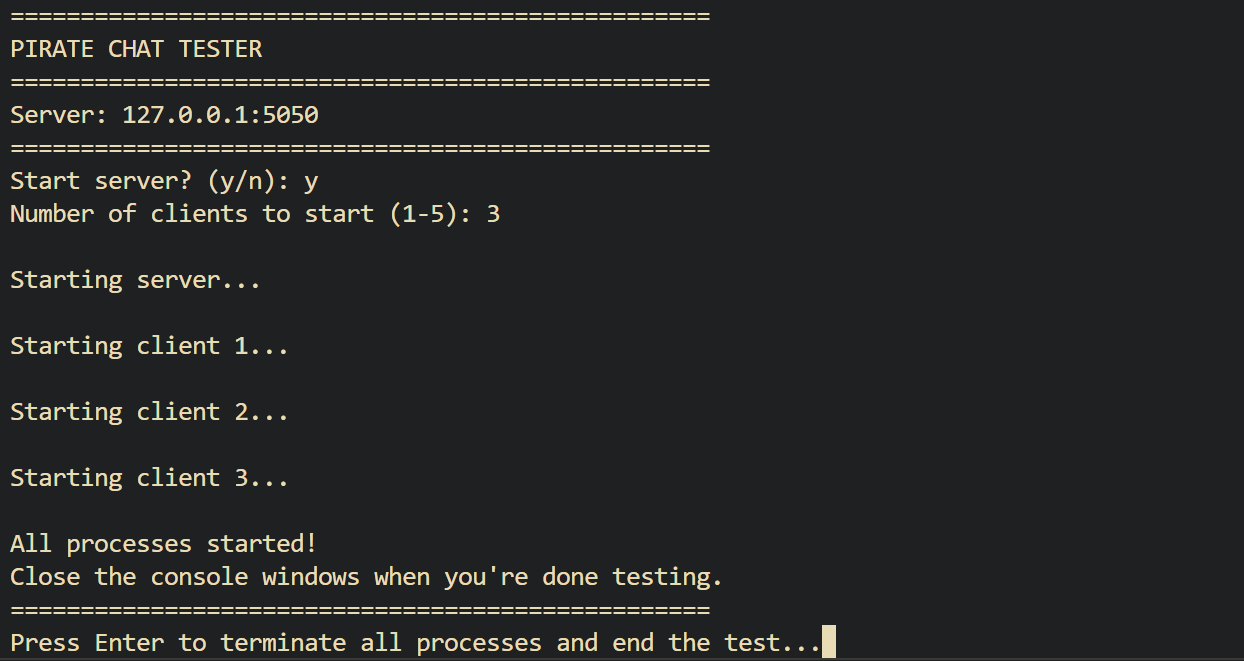
24 April 2025

A skull and crossbones on a black background with Skull and Bones in the background

AI-generated content may be incorrect.

**Pirate Chat**

Pirate Chat is a secure, real-time messaging platform developed in the Python programming language, and it allows multi-client support, allowing various users to communicate in a private chat room environment. The application offers encrypted message exchange and concurrent client handling by leveraging the power of Python's socket, SSL (TLS/SSL wrapper for socket object), and threading libraries. Adding a playful twist with the pirate theme adds a thematic charm, while the implementation demonstrates proficiency in secure programming. This report will outline the design, functionality, and evaluation of the Pirate Chat system, offering an insight into its architecture, communication model, installation process, and performance.



The design of Pirate Chat emphasizes security. On the server side of the application, the application would listen for a client’s connection using an SSL wrapped socket, which is secured with a self-signed certificate and private key (server.crt and server.key). Once a connection is established, the server will be able to spawn a new thread to manage each client independently. This threading model ensures the server can manage multiple users simultaneously. Each client is tracked in a shared list that is safeguarded using threading locks, enabling synchronized access during message broadcasting. While on the client side, the users are connected by entering a unique pirate name, after which they receive a welcome message and a Jolly Roger ASCII banner. Clients are then able to send and receive messages in real time chat. The application also supports custom commands such as “.list”, which provides a list of active users, and “.exit”, which allows for a graceful disconnection. Messages sent by one client are relayed by the server to all others, excluding the sender, using a broadcast mechanism.

A screenshot of a computer

AI-generated content may be incorrect.

Communication between clients is routed entirely through the server. Each message received from a client is handled in its dedicated thread and then broadcast to other clients. Special commands are interpreted by the client application itself. For instance, when a user types “.list”, the client requests the list of currently connected pirate names from the server. If a user enters “.exit”, the client notifies the server of the disconnect, terminates its session, and the server updates the client list accordingly. This message flow illustrates the seamless communication and graceful exit strategy implemented within the chat environment. Following along to how you would be able to run Pirate Chat, users must ensure their system meets a few basic requirements. Python 3.x and OpenSSL must be installed to enable secure connections. The SSL certificate can be generated using the command:

openssl req -new -x509 -days 365 -nodes -out server.crt -keyout server.key.

A screenshot of a computer

AI-generated content may be incorrect.

Finally, the users will have the certificates ready, and the users can launch the server by running “pirate\_server.py”. Clients can then join by executing “pirate\_client.py”. Both files must be in the appropriate directories, and the server’s certificate must be accessible to the client for trust validation. Some of the challenges we faced lay mostly in the SSL encryption process. It was very easy to verify the communication from socket to socket. Verifying that we had implemented the SSL encryption correctly was a bit trickier. Ultimately we used a packet-sniffer called Wireshark to capture the data packets send between client and server, to verify encryption. It indeed encrypted the data but also appeared to use an updated TCP protocol titled TCP 1.2 for transmission.

A screenshot of a computer

AI-generated content may be incorrect.

To illustrate its functionality, the application provides clear visual outputs in the terminal. The server window displays connection logs, SSL handshake messages, and broadcast logs. Meanwhile, the client terminal shows the welcome banner, message exchanges, and responses to commands such as “.list” and “.exit”. These screenshots, available in the accompanying appendix, visually confirm the smooth operation and encryption features of the application. From an evaluative standpoint, Pirate Chat excels in several areas. It achieves secure, real-time messaging with SSL encryption and concurrent processing through threads. The user interface, while basic, is clear and responsive, and command handling adds a layer of interactivity. Nonetheless, there are areas for future improvement. Introducing a graphical user interface (GUI) would enhance usability, while better certificate management, such as using a trusted Certificate Authority (CA), could improve accessibility. Additionally, integrating user authentication would help validate identities and further protect the chat environment. In conclusion, Pirate Chat is a well-rounded, educational application that demonstrates essential concepts in network security, multithreading, and socket programming. By creating a secure space for users to chat using SSL encryption and synchronized threads, the application effectively meets its design goals. Its simplicity in setup and robustness in functionality make it a strong foundation for further development into a comprehensive, secure messaging platform.

**Please open the README file included to find complete installation instructions.**

<https://github.com/BelaBartok39/Pirate-Chat-Project>

# References

"Network Programming in Python." n.d. March 2025. <https://bogotobogo.com/python/python\_network\_programming\_server\_client.php>.

"TLS/SSL wrapper for socket object." n.d. March 2025. <, https://docs.python.org/2/library/ssl.html>.

Tripathi, Satyam. "Guide to Python Socket Programming." n.d. March 2025. <https://builtin.com/data-science/python-socket>.

Ubah, Kingsley. "Implementing TLS/SSL in Python." n.d. March 2025. <https://snyk.io/blog/implementing-tls-ssl-python/>.

"Wireshark Tutorial and Cheat Sheet." n.d. April 2025. <https://hackertarget.com/wireshark-tutorial-and-cheat-sheet/>.