Report

I have developed a basic messaging program in python. It supports 1-1 messaging, basic authentication, and encrypted connection between client and server.

Architecture

A diagram of a server

Description automatically generatedThis program has a client-server architecture that allows for clients to register and log in, and for clients to send messages to each other. The user can interact with the client in two ways: logging in/registering or through user input after logging in. User input is either interpreted as a command if it begins with ‘/’ or it is interpreted as a message. Commands are used to either close the client or set the intended recipient for the messages which is passed to the server along with each message. All communication with the server first passes through the TLS 1.2 protocol, encrypting the data before it is sent through the socket connected to the server and decrypting data received through it. Received messages are displayed to the user.

The server works without human input, only allowing the administrator to view connected clients and shut down the server. The login/registration module records registered users and ensures that only one client can be signed in per user. The client management module records connected clients and their corresponding output sockets. This is used by the message routing module to verify that the recipient of an incoming message exists and to route the message to the correct client. All messages passed from a client to the server are encoded using base64 to prevent any form of injection attack as ‘:’ is used as a separator, which may lead to errors if included in a user input.

Encryption

To encrypt the TCP messages sent between the server and clients, I chose to use TLS 1.2 protocol using a 128-bit AES cipher and SHA hash function. The certificate I used for this is a self-signed certificate.

A computer code with text

Description automatically generated

This protocol wraps the socket, encrypting all outgoing messages and decrypting all incoming messages.

A close up of a logo

Description automatically generated

On the client side, only the protocol and the cipher are defined as the certificate is sent over from the sever side.

A close-up of a computer code

Description automatically generated

TLS 1.2 works in the following steps: The client sends the server a ClientHello message containing TLS versions and ciphers supported and a random number.

A screenshot of a computer

Description automatically generated

The server then responds with a ServerHello message that contains the chosen version and cipher, along with another random number. The server also sends its certificate, which contains a public key for the client to use. This certificate, in practice, should be signed by a trusted certificate authority, but in this case, I’m using a self-signed certificate.

A screenshot of a computer

Description automatically generated

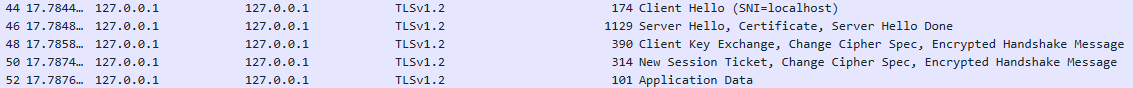
Once the client receives the certificate, it generates a random pre-master secret and encrypts it using the server’s public key, ensuring that only the server can decrypt this. This pre-master secret is then sent to the server and is decrypted using its private key.

A screenshot of a computer

Description automatically generated

Using this pre-master secret and the two random numbers exchanged in the Hello messages, both the server and client derive a master secret. This master secret is used to generate symmetrical session keys that will be used for all communication during this session before they are discarded.

In my case, as I chose to use AES-128 and SHA, two symmetrical keys will be used. One is used for AES or Advanced Encryption Standard, with the 128 bits representing the size of the key and the size of the chunks of data encrypted at a time. This encrypts the data to ensure that without the key, intercepted messages cannot be read. The other key is used for SHA hashing. This is used on the data to produce a string of fixed size or hash that is to be sent with the data. Once the message arrives on the other side, the key can be used on the received data to confirm that the data sent and data received is identical, ensuring integrity.



A close up of a message

Description automatically generated

As shown in the Wireshark screenshots, the results of this protocol are TCP messages containing encrypted data that cannot be read without deciphering using the symmetrical keys.