Matt Conflitti

CIS457

Project 4 Doc

This project went much better than previous projects. I was able to establish a non-secured chatroom quite quickly using previous concepts such as a channel selector and threading. I used a selector in the server because I knew there was going to have to be communication between threads if I used threads and that is not much fun from what I saw online. So since there was going to be a couple shared maps between SocketChannels that contained the AES keys and also just storing the socket information so that data could be sent back out to any connected clients, it was best to use a single threaded method such as a selector. On the client side, it made since to use threads because only one connection needed to be made and it is nice to split up the sending and receiving concerns. By doing so it allows each thread to focus on one task and make the structure of the code much more coherent. The only shared info between the two threads is the AES key, but that can be pushed down into each thread through the constructor since that info will not change for the duration of the connection. The overall structure of the client is quite simple. When the receiving thread gets an encrypted message it will split the buffer up to get the initialization vector, the length of the message, and the message itself. Then the message trimmed to the proper length is decrypted with the cryptotest class provided and displayed to the user. The sending thread does the opposite of this. It creates a new initialization vector, gets the length of the command, and gets the message itself. It packs these up and sends it through the SocketChannel open. The server is bit more complex since it has to handle all of these connections. When the client first connects to the server, the socket will be added to a map and then the AES key is sent to the server to be stored. The key is sent using RSA encryption. Once the server has this information the user is essentially logged in and can begin sending commands to the server. When a command is sent in, the server will check to make sure the client’s socket info is stored and that an AES key exists for that user. If all is well, the server will decrypt the command and run that command through some logic to determine what to do. I found a great tutorial online that explains the Java Selector and how you can re-register channels to the Selector to break up the logic into stages. The stages include acceptance, reading, and writing. I liked this because it allows me to logically split up the work and improve code readability. This way, I can have specific logic that occurs when a socket is opened and connected, when information is received, and when information needs to go out. If the client was requesting information it will send the correct info. If the client is sending a message to someone it will look up that user in the map and if it exists, it will take that socket info and send the encrypted message over encrypted with that user’s AES key. When a user is kicked, it is removed from the map and a quit message is sent to the client being kicked. This quit message tells the client to shut down. The only pitfalls I had were getting the encryption logic correct. If encryption is not done in the right order or if the message to be decrypted is not pulled from the buffer properly it would not work. It was more of a ByteBuffer issue, but it was easily avoidable. This was a very good project. I enjoyed making a secure chatroom.