

UNIVERSITY OF MIAMI

ECE376 - CYBERSECURITY

SPRING 2024

Final Project Report

Secure User Authentication System

GEORGE CHIEFFI

May 3, 2024



ECE376 SECURE USER AUTHENTICATION SYSTEM

George Chieffi, University of Miami

May 3, 2024

Contents

Introduction	2
Client Application Flow	2
Home Menu	2
Signup	2
Login	3
Secret Action	3
Technical Implementation	3
Client	4
Server	4
Database	5
Protocol	5
Design Choices	6
SSL	6
Argon2	6
Muulti-Threading	7
Conclusion	7

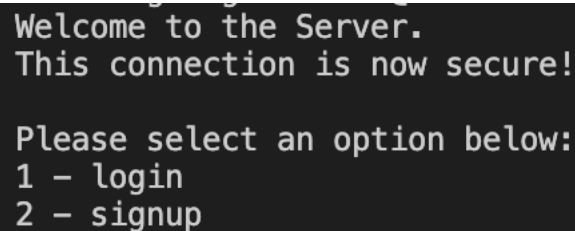
Introduction

Authentication has always been at the forefront of defensive cybersecurity. Systems, businesses and organizations rely on authentication to verify a users identity. By practicing strong authentication practices, organizations defend better against several different attack vectors such as phishing attacks, man-in-the-middle attacks, etc. In this report I will describe the technical implementation and specific design choices for the secure user authentication system I have built.

Client Application Flow

Below will be a series of screenshots showing the flow of events that the user will follow when interacting with the application.

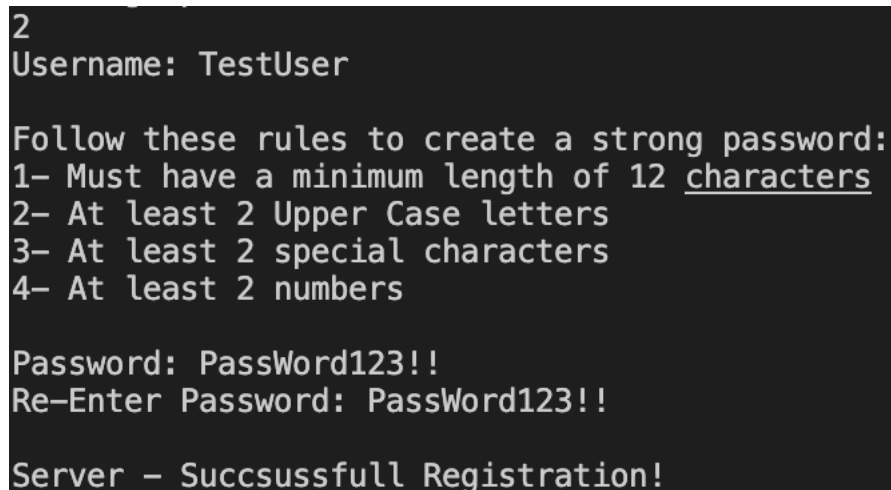
Home Menu

A terminal window with a black background and white text. The text reads: "Welcome to the Server." followed by "This connection is now secure!". Then, "Please select an option below:" followed by a list: "1 - login" and "2 - signup". A white cursor is positioned at the end of the "2 - signup" line.

```
Welcome to the Server.  
This connection is now secure!  
  
Please select an option below:  
1 - login  
2 - signup  
█
```

This is the clients initial state. The user will be faced with 2 options: login to the server, signup and create and account

Signup

A terminal window with a black background and white text. The text shows the user has selected option 2, then entered "TestUser" as the username. It then lists four rules for a strong password: minimum length of 12 characters, at least 2 upper case letters, at least 2 special characters, and at least 2 numbers. The user entered "PassWord123!!" for the password and "PassWord123!!" for the re-enter password. The final message is "Server - Succsussfull Registration!".

```
2  
Username: TestUser  
  
Follow these rules to create a strong password:  
1- Must have a minimum length of 12 characters  
2- At least 2 Upper Case letters  
3- At least 2 special characters  
4- At least 2 numbers  
  
Password: PassWord123!!  
Re-Enter Password: PassWord123!!  
  
Server - Succsussfull Registration!
```

Since the client presumably does not have a pre-existing login, they select option 2 which is signup. They are prompted to enter a username and password. Before entering a password

the user is shown 4 rules to follow to ensure the creation of a strong password. The client must also enter the password twice to make sure their original password does not contain a typo. After entering this information the server will send a conformation message informing the client their registration was successful.

Login

```
Username: TestUser
Password: PassWord123!!

Server - You are now logged in!
Server - Take the secret flag: CTF{C0ngr@tZ_on_$ecur3ly_4uthent1c@t1ng_Y0urs3lf}

9 - Secret Flag
█
```

After creating an account the user is showed the home menu once again. The above screenshot shows what happens after selecting the login option. The user is prompted to enter a username and password. If the credentials they entered are validated against the database, the server sends the above messages. Otherwise, the server sends the client an error message and has them send over a new login request.

Secret Action

```
9 - Secret Flag
9
Enter the flag: CTF{C0ngr@tZ_on_$ecur3ly_4uthent1c@t1ng_Y0urs3lf}

Server - We are now closing your connection. You have succsessfully:
    Created an account
    Logged into your account
    Performed an action on the server reserved for logged in users
Server - Thank you!
Server - CLOSED
```

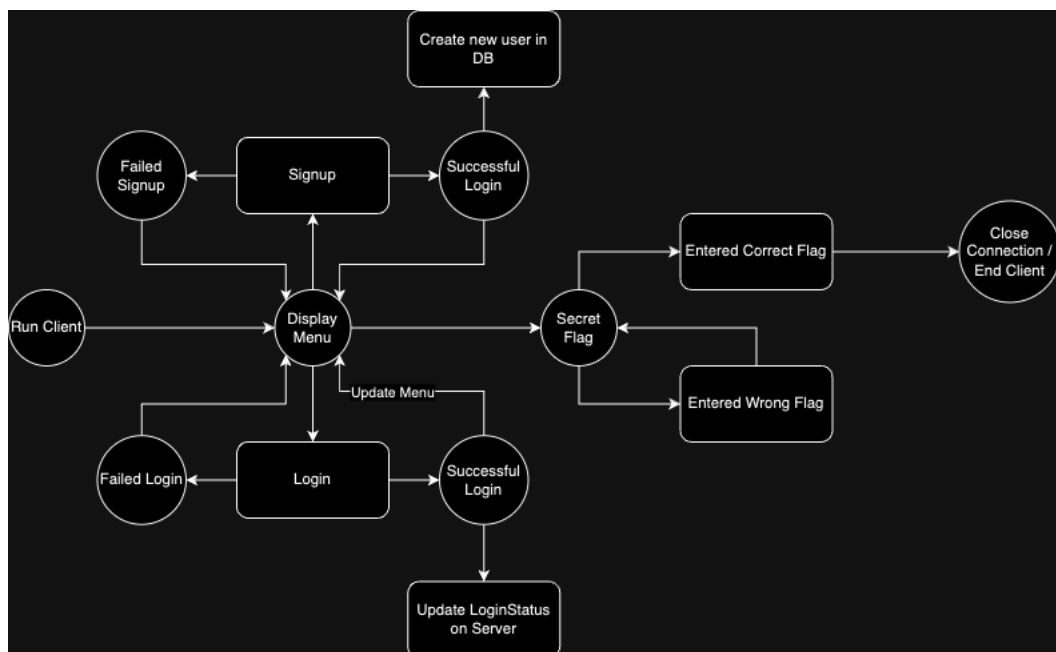
After successfully logging into the server, the clients menu changes to only display '9 - Secret Flag'. When selecting this command the client is prompted to enter the flag they received in the initial login response. If the correct flag is entered the server sends three messages to the users summarizing their experience. The connection to the server is then closed on both ends.

Technical Implementation

A slight overview of the project before I describe each component in detail. The project was built following a Client-Server architecture connected through sockets communicating using the TCP protocol. All communication is encrypted via the implemented Secure Socket Layer.

Client

The client has a very basic implementation. From a very high level, the client creates a socket, wraps it in an SSL context using the servers certificate (already has access to this), then attempts to connect to the server on port 50000. If the connection is successful, the client will go through a loop reading messages from the server and sending messages the user creates. The user creates these messages from following a prompt already defined on the client. The users input is joined together with other data and send as one message (I will go over this protocol more in the protocol subsection). If the server does not accept the connection for any reason, the client will catch the connection error and print an error message to the user the gracefully shutdown.



Server

The server has a more complex implementation but overall isn't too confusing. There are a few steps the server must complete before it is ready to authenticate any users. First the server makes sure it has a database to access for later, if not it will call a function to create and configure one. Then the server will create an SSL context using its CA key and certificate. After the context is created, a socket will be created, binded to localhost port 50000, and start listening to a max of 5 simultaneous connections. After a client initiates a connection the server will accept the client, wrap the connection with its SSL context, and pass the handling of this client to a new process thread, printing log messages on the server the entire time.

After a new thread is created and a client is passed to this thread a few things will happen. First a user object will be created for the user. The user object simply contains two variables, the current clients address and a boolean value pertaining to the loggedInStatus of the corresponding client. Second, a welcome message is created and sent to the user. Now the server will start a 'read loop' that will read incoming messages from the client and either pass the message to another function for handling or send the client an error message because the command they are trying to use is not recognized.

There are 3 commands the client can send to the server: login, sign up, or a secret command only available to signed in users. Login will take the data from the clients message and expects to successfully extract 2 variables from it, username and password. A DB connection is established and a select query is executed to receive the stored hashed password corresponding to the user. The password from the client is then verified against the stored password. If the passwords are a match, the user object loggedInStatus will be changed to true and the client will receive a message telling them their logged in attempt was a success.

Sign-up also attempts to extract 2 variables from the clients message, a username and password. The password is checked on the client to ensure it follows standard strong password rules and entered twice so the client doesn't submit a password with a typo. After receiving a strong password and username the server will hash the password, and insert the new user into the database.

Database

This project implements an SQLite3 database in order to contain the users. If the application was larger the SQLite3 db would be more than capable to create new tables associated with the user table in order to support and additional business logic added in the future. Currently, since the system is strictly for user authentication, the only table the db holds is 'users'. Below is a picture describing the tables schema.

users
ROWID : INT NOT NULL AUTO INCREMENT PK
username : TEXT NOT NULL
password : TEXT NOT NULL
Timestamp : DATETIME DEFAULT CURRENT_TIMESTAMP

Protocol

Although the base communication protocol followed between the client and server is TCP, there is also a custom protocol built on top. This custom protocol actually differs slightly between the client and server. To explain this concept, we will look at the protocol from perspective of the client sending and receiving messages from the server. When reading messages, the client breaks server messages down into two parts: the data length and the data. Data length simply describes the length of the data, and the data is whatever information the server wants the client to know. A predefined rule of datalen is that it will always be 10 bytes long. so if the length of data is 22 characters the data len will look like the following 22——. In practice the dashedlines are empty characters but the principle of trailing spaces applies. After reading the first 10 bytes from the message the client will decode the datalen (because it is utf-8 encoded before transmission) and know exactly how many bytes to allocate to read from the connection to get data.

When sending a message to the server, the message needs to contain one more piece of information in its header, the command. The server needs to know if the client wants to login, signup, or perform the secret action. When creating a message for the server, the client allocates the first 2 bytes to the command. the rest of the message is the same, datalen and data. When sending variables through the data portion of the message, each variable is separated by the delimiter "&".

To recap the message structure from server to client is "DATALEN + DATA", and client to server is "COMMAND + DATALEN + DATA". At this point the message is one large concatenated string. In order to send this message across TCP it needs to be encoded into bytes using utf-8. When the message reaches its final destination each section of the message is read and decoded separately and handled appropriately.

Design Choices

The following subsections discuss the design choices made during this project to ensure security and efficiency of the system.

SSL

As stated in the technical implementation section, the first thing the client and server does is create an SSL context to later wrap the TCP connection in. This was a conscious effort to ensure the confidentiality and integrity of the data communication layer. SSL will verify the sender of the data is who they say they are and the receiver was the intended target of this data. This prevents against man-in-the-middle attacks or any other interference attack from happening. In addition, the data sent across the communications layer is encrypted at all stages of delivery. So even if someone was sniffing the line they would only see encrypted data.

Argon2

Selecting a password hashing algorithm is very difficult. The most important factor when choosing a hashing algorithm is the computational expense of performing the hash and the salt. The computational expense should be high enough where as a brute force attack is not worth it and will take too long, and salt protects against dictionary attacks.

After doing research I discovered Argon 2 was currently the most secure password hashing algorithm, specifically Argon2id. This password hashing algorithm implements a memory cost for each hash, providing the computational expense needed. In addition, by default a truly randomly generated salt is created with each new hash. The algorithm takes a raw password and generates a long string of characters containing the variant of Argon2 used, the version used at time of hashing, the salt, and finally the hashed password. This string is stored under password in the user table.

Multi-Threading

Multi-Threading was chosen in this project in order to ensure availability of the system. For every connection made to the server, a new thread is spun up and started to handle each user. This not only allows each user to interact with the server quicker after already establishing a connection, but also doesn't block the server from accepting connections on the main process thread.

Conclusion

In conclusion, the system is not a perfect user authentication system and does have a few future features that must be added to follow current guidelines, but is a good representation of how confidentiality, integrity, and authenticity can be achieved for the purpose of user authentication over a TCP socket connection. The most important future feature of this project would be the addition of MFA, Multi-Factor Authentication. In a study done by Google, NYU, and UCSD, MFA blocked 100% of automated bots, 99% of bulk phishing attacks, and 66% of targeted attacks on google users accounts. Clearly, implementing MFA in any authentication system would greatly increase its resistance to any type of cyber attack.