LOVELY
PROFESSIONAL
UNIVERSITY

Transforming Education Transforming India

Topic: - Secure Authentication Module for Operating Systems

Name: Ch. Sri ram

Registration No.: 12309600

Section: K23BE

Roll No.: 15

Submitted to:

Mrs. Pushpendra Kumar pateriya

1. Project Overview

Objectives:

- Develop a strong authentication system that offers secure login procedures.
- Implement role-based access control, multi-factor authentication (MFA), and password hashing.
- Safeguard against security threats such as session hijacking, brute force attacks, and unauthorized access.

Expected Outcomes:

- A secure and multi-layered method of confirming identity.
- Prevention of frequent authentication errors.
- Protection against common authentication vulnerabilities.

Scope: The aim is to have user authentication mechanisms in place. Securely store credentials and session management. Include access control rules and encryption to prevent unauthorized access. Future upgrades will incorporate adaptive security techniques and biometric authentication.

2. Module-Wise Breakdown

Module 1: User Authentication & Access Control

Objective: Grant operating system access only to authorized users. Function:

- Implement password hashing to make user registration secure.
- Two-factor authentication (biometric or OTP) is being implemented.
- To control system privileges, employ role-based access control (Admin, User, Guest).

Module 2: Credential Security & Encryption

Objective: Safe storage and transmission of authentication data. Function:

- To prevent credential theft, password hashing algorithms such as Argon2 are employed.
- Utilize RSA or AES encryption to secure stored credentials.
- Token-based automatic session expiration and secure session management with tokens.

Module 3: Threat Detection & Prevention

Objective: To discover and reduce security threats to authentication. Function:

• Implement CAPTCHA authentication and account lockout policies to provide brute-force protection.
• Implement secure cookies and encrypted session tokens to avoid session hijacking.
• Monitoring and logging authentication attempts for the purpose of detecting suspicious activity.

3. **Functionalities**

Module 1: Authentication System

• **Login System:** Has multi-factor authentication via biometric or OTP.
• **User Registration:** Utilizes password hashing to securely save user information.
• **Session Management:** Prevents unauthorized access using secure tokens that automatically expire.

Module 2: Credential Storage and Encryption

• **Password Hashing:** This stores passwords securely via Argon2.
• **Encryption:** Uses RSA or AES to encrypt data related to authentication.
• **Access Logs:** For use in security verification, we log user login attempts**.**

Module 3: Threat Detection & Prevention

• **Brute Force Protection:** Requires CAPTCHA login protection and rate limiting.
• **Session Security:** Prevents session hijacking through the implementation of encrypted tokens and secure cookies.
• **Logging & Monitoring:** Tracks attempts to log in and detects abnormal activity for investigation.

4. **Suggestions for Technology**

Language of Programming:

• **C:** For optimal memory management and low-level OS authentication methods.
• **C++:** For secure interaction with systems and creating modular authentication.

Libraries:

• **Password Hashing:** Argon2 is employed to hash passwords and save them securely.
• **Encryption:** For the protection of credentials, employ RSA and AES (Advanced Encryption Standard).
• **2FA Implementation:** Generation of OTP using the C++ random library or the Google Authenticator API**.**

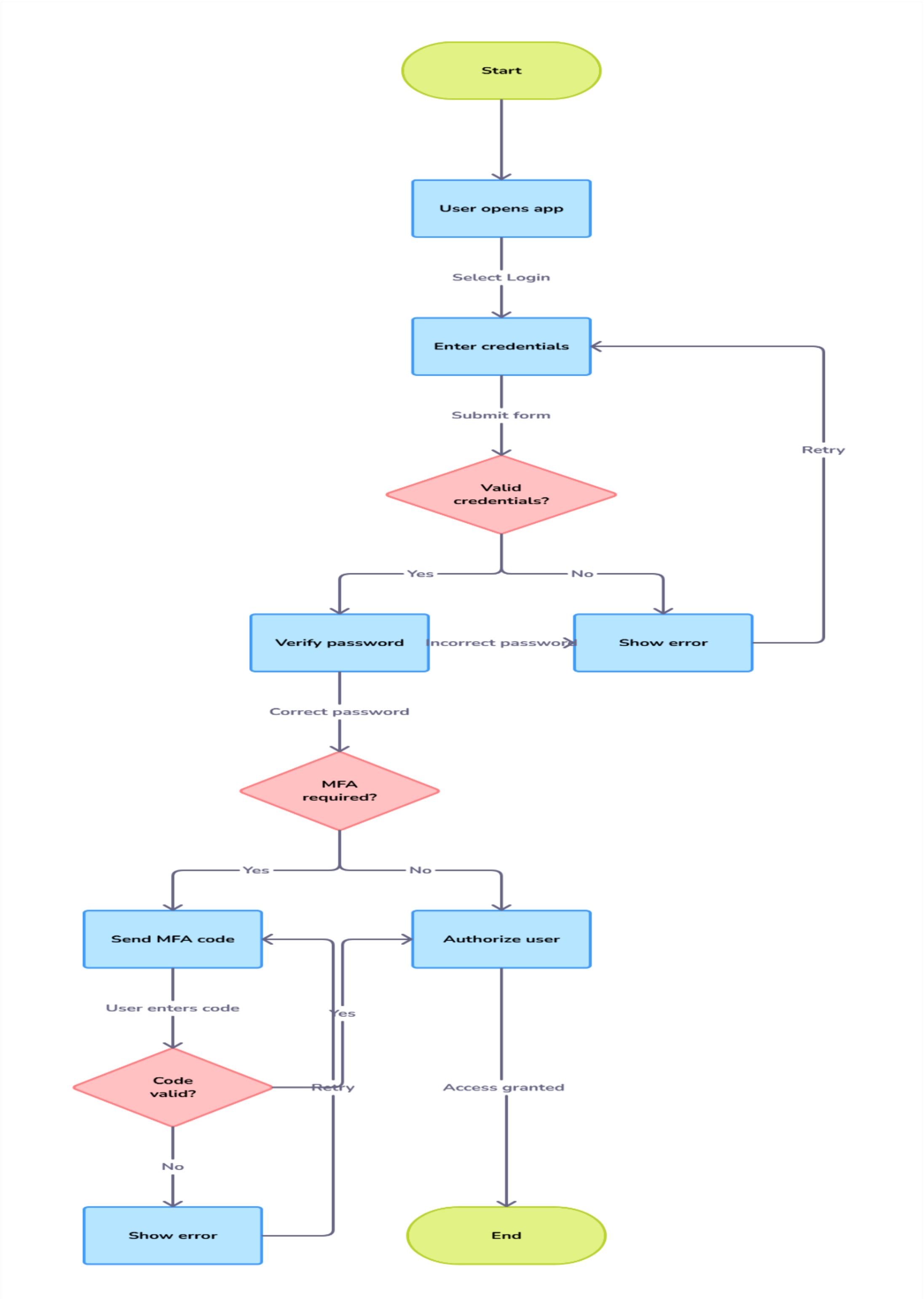
Tools for Development:

- GCC or MinGW (for C and C++) is the compiler.

- IDE: Visual Studio Code or Code::Blocks for an intuitive user interface.

Database: SQLite or flat-file credential storage for authentication records.

5. Flow Diagram:



6. Revision Tracking on GitHub

- Repository Name: Secure-Authentication-Module-
- GitHub Link: https://github.com/Demon-sunny/Secure-Authentication-Module-

7. Conclusion and Future Scope

Conclusion: In summary, the secure authentication module significantly enhances system security through real-time threat detection, encrypted storage of credentials, and strong authentication mechanisms. The system ensures data integrity, confidentiality, and availability through multi-factor authentication, secure session management, and encryption measures. It provides a scalable and reliable operating system authentication subsystem, lowering the likelihood of security attacks like session hijacking and brute force attacks.

Future Scope:

- **Biometric Authentication:** To provide more security, include facial and fingerprint recognition.
- **Adaptive Security:** Leverage AI-driven threat detection to scan authentication anomalies in real-time.
- **Cloud Integration:** Develop federated identity management-compatible secure authentication systems.
- **Scalability:** Add user management and strong authentication for large enterprises to the module.

8. References:

- To complete the project and expand your understanding, you can utilize the following resources:

  Books:

- "Cryptography and Network Security" by William Stallings
- "The C++ Programming Language" by Bjarne Stroustrup

  Web Resources:

- Official OpenSSL Documentation – For encryption and secure authentication mechanisms.
- C++ Reference – Documentation for C++ functions and libraries.
- GeeksforGeeks for secure coding example and basic threat identification.

Tools:

- OpenSSL Documentation: https://docs.openssl.org/master/
- SQLite Tutorial: https://sqlite.org/docs.html
- Research Papers and Articles:
- Explore IEEE and ACM publications on secure authentication techniques and encryption methods.

Appendix

B. Problem Statement:

How can a secure authentication module be developed for operating systems that effectively prevents brute force attacks, credential leaks, and phishing by integrating additional security measures such as CAPTCHA verification, passkeys, OTP (one time password) validation and secure password?

C. Solution/Code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <ctime>

using namespace std;

//Generate a simple hash
string simpleHash(string input) {
   unsigned long hash = 5381;
   for (char c : input) {
      hash = ((hash << 5) + hash) + c;
   }
```

```
    return to_string(hash);
}

//To craete CAPTCHA
int generateCaptcha() {
    int num1 = rand() % 10;
    int num2 = rand() % 10;
    cout << "CAPTCHA: " << num1 << " + " << num2 << " = ? ";
    return num1 + num2;
}

//Generate a 6-digit OTP
int generateOTP() {
    return 100000 + rand() % 900000; // Generates a 6-digit OTP
}

//register
void registerUser() {
    string username, password, passkey, hashedPassword;
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    cin >> password;
    cout << "Enter passkey (set your unique passkey): ";
    cin >> passkey;
    // Hash the password before storing
    hashedPassword = simpleHash(password);

    // Store credentials in a file
    ofstream outFile("users.db", ios::app);
    outFile << username << " " << hashedPassword << " " << passkey << endl;
    outFile.close();

    cout << "Registration successful!\n";
}

//login
void loginUser() {
    string username, password, passkey, storedUsername, storedPassword, storedPasskey;
    cout << "Enter username: ";
    cin >> username;
    cout << "Enter password: ";
    cin >> password;
    cout << "Enter passkey: ";
    cin >> passkey;

    //Generate and validate CAPTCHA
    srand(time(0));
    int captchaResult = generateCaptcha();
    int userCaptchaInput;
    cin >> userCaptchaInput;
    if (userCaptchaInput != captchaResult) {
        cout << "CAPTCHA verification failed! Login denied.\n";
        return;
    }

    ifstream inputFile("users.db");
    bool loginSuccess = false;

    while (inputFile >> storedUsername >> storedPassword >> storedPasskey) {
        if (username == storedUsername && simpleHash(password) == storedPassword && passkey == storedPasskey) {
            loginSuccess = true;
            break;
        }
    }
    inputFile.close();

    if (loginSuccess) {
        //Generate OTP
        int generatedOTP = generateOTP();
        cout << "Your OTP is: " << generatedOTP << endl;

        int enteredOTP;
        cout << "Enter the OTP: ";
        cin >> enteredOTP;
        if (enteredOTP == generatedOTP) {
            cout << "Login successful! Welcome, " << username << "!\n";
        } else {
            cout << "Invalid OTP! Access denied.\n";
        }
    } else {
        cout << "Invalid credentials! Access denied.\n";
    }
}

int main() {
    int choice;
    while (true) {
        cout << "\nSecure Authentication System\n";
        cout << "1. Register\n";
        cout << "2. Login\n";
        cout << "3. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                registerUser();
                break;
            case 2:
                loginUser();
                break;
            case 3:
                cout << "Exiting...\n";
                return 0;
            default:
                cout << "Invalid choice! Try again.\n";
        }
    }
}
```