Secure Password Storage and Validation Using Hashing

Ryan Nash

Department of Mathematics and Computer Science

High Point University

ABSTRACT

This project explores a practical application of hashing to solve a critical issue in software security: password protection. Insecure password storage methods, such as plaintext or unsalted hashes, pose severe risks when systems are breached. To address this, a secure password management system was developed using SHA-256 hashing with random salts in C++. The system supports secure user registration and login validation while implementing basic brute-force protection. This paper outlines the motivation, design, and implementation of the project, and evaluates the outcome from both a technical and conceptual standpoint.

INTRODUCTION

Password management is a cornerstone of cybersecurity. However, even in modern systems, passwords are frequently stored in plain text or with insufficient protection, leaving users vulnerable in the event of a breach. Hashing algorithms provide a solution by transforming user passwords into fixed-length, irreversible strings that can be securely stored. When coupled with additional techniques like salting and brute-force protection, password systems become significantly more secure.

This project proposes and implements a secure password registration and login system that utilizes hashing principles in a real-world context. The project leverages C++ and the OpenSSL library to implement SHA-256 hashing and uses hash tables for efficient data storage.

DESIGN & IMPLEMENTATION

The system was built in C++ using the OpenSSL cryptographic library. It features two main functionalities: registering new users and validating login attempts.

During registration, the system generates a unique 16-character alphanumeric salt for each user. This salt is concatenated with the user's password and hashed using SHA-256. The resulting hash and salt are then stored in an unordered_map with the username as the key.

To authenticate a user during login, the system retrieves the stored salt for that username, hashes the input password with that salt, and compares it to the stored hash. If they match, access is granted. Otherwise, the system allows up to three attempts before temporarily locking the account.

Key features of the implementation include:

- generate_salt(): Generates random alphanumeric salts.
- sha256(): Hashes strings using OpenSSL's SHA-256 function.
- register_user(): Handles user registration and stores the salted hash.
- login_user(): Verifies passwords using salted hashes and enforces lockouts.

RESULTS & EVALUATION

The resulting system meets its design goals. It securely stores user credentials and prevents login with incorrect passwords unless the correct salted hash is reproduced. The use of unordered_map provides O(1) average-time complexity for user lookups, ensuring efficiency.

Testing confirmed that:

- Salts are unique per user and effective against precomputed hash attacks (rainbow tables).
- Hashed passwords cannot be reversed.
- Brute-force attempts are limited by the lockout mechanism.

Although functional, the system can be further improved. For example, switching from SHA-256 to bcrypt or Argon2 would provide even stronger defense by including computational delay. Additionally, future versions could introduce password strength validation, account recovery features, or a GUI.

CONCLUSION

This project successfully applied hash-based techniques to a real-world security problem. By integrating SHA-256 hashing, random salting, and a basic lockout mechanism, the system offers a solid foundation for secure password handling. The experience provided valuable insights into both the theoretical and practical aspects of cybersecurity and hashing. Future development could enhance the system with stronger hash algorithms, password complexity checks, and user-friendly interface improvements.