A project report on password strength analyzer

A Web-Based Security Tool

Under the supervision of

Dr. Kakali Chatterjee

Assistant Professor

Computer Science and Engineering

Submitted By

Raman Kumar - 2206180

Rahul Kumar - 2206165

Ayushmaan - 2206269

Contents

0.1 Introduction

Weak passwords remain a critical security risk, often targeted by brute-force and dictionary attacks. The Password Strength Analyzer is a web-based tool designed to evaluate password security in real-time, helping users create stronger passwords. Built using HTML, JavaScript, Vite, and Tailwind CSS, it assesses passwords based on complexity criteria and checks against a list of 10,000 common passwords. The system assigns a 0–100 score, labeled as Weak (`<=50`), Moderate (`51-80`), or Strong (`>80`), and computes SHA-256 hashes to demonstrate secure password handling. This report details the project's objectives, technical design, implementation, results, and future potential.

0.2 Objective

The project aims to:

- Provide instant feedback on password strength to guide users toward secure choices.
- Enforce password complexity policies, including minimum length and character diversity.
- Educate users on cryptographic hashing through client-side SHA-256 computation.
- Deliver a responsive, user-friendly interface using modern web technologies.

By combining scoring, common password checks, and hashing, the analyzer seeks to enhance user awareness and promote best practices in password security.

0.3 Technical Overview

The analyzer operates entirely client-side, leveraging:

- **HTML**: Structures the input form and result display.
- **JavaScript**: Powers the `PasswordAnalyzer` class for logic.
- **Vite**: Streamlines development and bundling.
- **Tailwind CSS**: Styles the UI with responsive design.
- **Web Crypto API**: Computes SHA-256 hashes.

Passwords are evaluated for complexity (length $\geq$ 8, uppercase, lowercase, digits, special characters) using regular expressions. A `Set` of 10,000 common passwords (loaded from `common_passwords.txt`) ensures vulnerable inputs are flagged. The scoring system awards 20 points per complexity criterion, deducts 50 points for common passwords, and adds 10 points for passwords longer than 12 characters, capped at 100. Results are labeled with (Weak, Moderate, and Strong) and colored (red, yellow, green) for clarity.

0.4 Implementation

The project is structured as a single-page web application. Key components include:

- **Setup**: Initialized with npm, Vite, and Tailwind CSS. The `common_passwords.txt` file resides in `public/`.
- **Logic**: The `PasswordAnalyzer` class checks the password's complexity. This means it looks at the password to see if it's built in a way that makes it hard to guess. It checks five things:
  - **Length**: Is the password at least 8 characters long? For example, "sunshine" has 8 characters, so it passes this check.
  - **Uppercase letters**: Does it have at least one capital letter, like A, B, or C? In "Sunshine", the "S" is uppercase, so it counts.
  - **Lowercase letters**: Are there small letters, like a, b, or c? "sunshine" has lots of these.
  - **Numbers**: Does it include digits, like 1, 2, or 3? For example, "sunshine123" has numbers.
  - **Special characters**: Are there symbols like !, @, #, or $? "sunshine!" has an exclamation mark, which is a special character.

  To do these checks, the class uses something called regular expressions, which are like patterns that search for specific types of characters. For instance, it uses `[A-Z]` to find any uppercase letter and `[0-9]` to find numbers. If a password passes a check, the class notes it down, like ticking a box on a checklist.
- **UI**: A Tailwind-styled interface displays the score, label, emoji, SHA-256 hash (with `break-all`), and recommendations.

The system uses `fetch` to load common passwords into a `Set`, ensuring efficient lookups. Error handling manages network failures gracefully.

0.5 Result

Testing validated the analyzer's functionality:

- `password`: Scored 0 (Weak, red) due to common password penalty.
- `MyPass2023`: Scored 80 (Moderate, yellow), meeting four complexity criteria.
- `MyVeryLongPass$2023`: Scored 100 (Strong, green), exceeding all criteria with length bonus.

Performance is robust, with hashing under 10ms and real-time UI updates. The responsive design adapts to various screen sizes. Limitations include client-side hashing (not production-ready) and static scoring weights, which may oversimplify strength assessment.

0.6 Conclusion

The Password Strength Analyzer successfully delivers real-time password evaluation, enforces complexity policies, and educates users on hashing. Its intuitive interface, powered by Tailwind CSS and Vite, ensures accessibility and ease of use. By integrating common password checks and a clear scoring system, it promotes stronger password practices, addressing a key cybersecurity need.

0.7 Future Work

Potential enhancements include:

- **Entropy Scoring**: Incorporate Shannon entropy for better randomness assessment.
- **Server-Side Hashing**: Move hashing to a backend for production security.
- **Larger Password Lists**: Expand to millions of known passwords for improved detection.
- **UI Features**: Add a progress bar for visual score representation.

These additions would strengthen the analyzer's accuracy and applicability.