Password Strength Tester
Design and Architecture

Aymen MSADDAK
Moona SAADAOUI
Houssine KHLIF
Aya ARFAOUI

May 4, 2025

1 System Design Overview

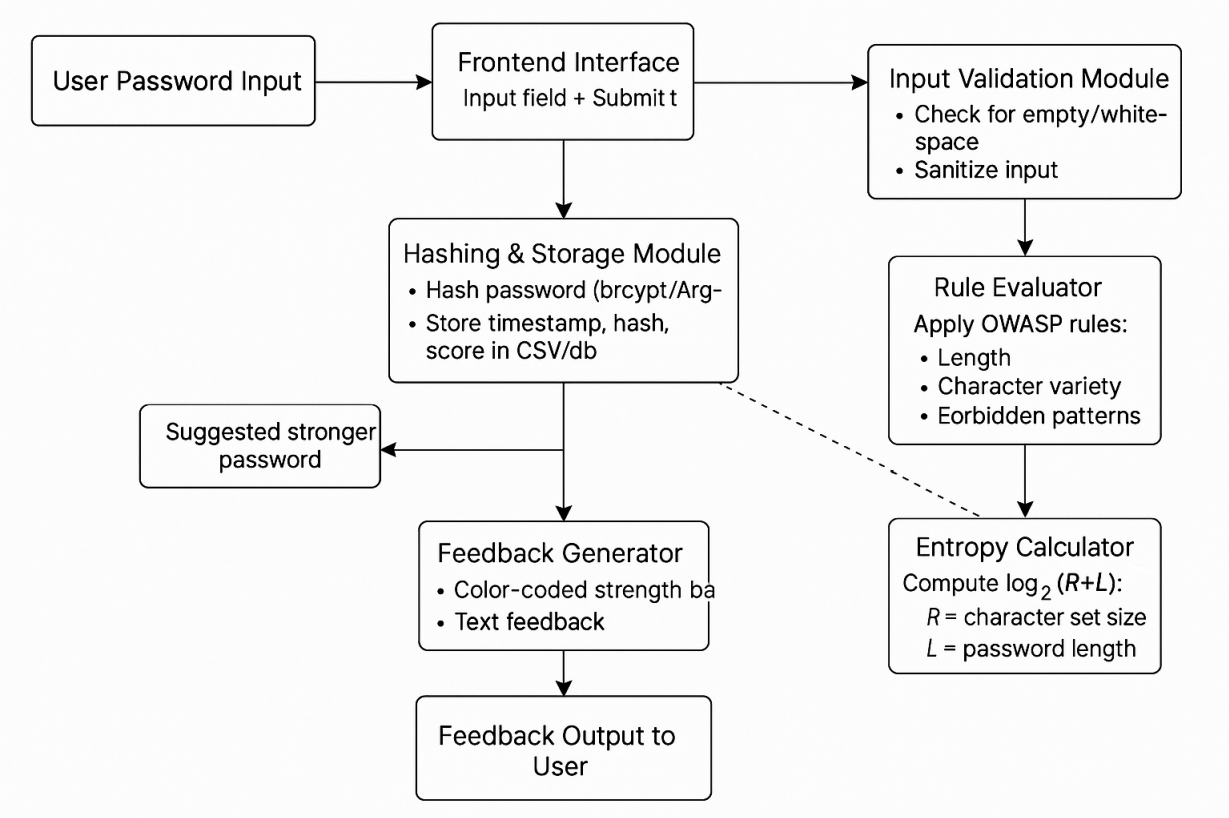

Figure 1: System Design

2 System Components

The proposed Password Strength Tester is composed of modular and interactive components designed to enhance usability while adhering to cybersecurity standards. The architecture is divided into frontend and backend layers, with secure handling and processing mechanisms at each stage.

2.1 Frontend Interface

This component serves as the direct interaction point for users and includes:

- User Input Field: A simple and secure text input box for users to enter their desired password.
- Strength Indicator Bar: A dynamic, color-coded visual element that reflects password strength in real time (e.g., red = weak, green = strong).
- Feedback Section: A textual display that provides constructive feedback and improvement suggestions (e.g., "Add special characters", "Too short", "Avoid using common phrases").

2.2 Password Analyzer

This backend module is responsible for assessing the input password against OWASP-inspired security rules:

- Validation checks:
- Regex-based evaluation
- Entropy calculation

2.3 Suggestion Engine

When weaknesses are identified, this engine provides intelligent recommendations:

- Password improvements
- Strong password recommendation

2.4 Feedback Generator

This module compiles results from the analysis phase and translates them into user-readable feedback:

- Color-coded strength result
- Textual suggestions

2.5 Hashed Password Storage Module

To mimic real-world password processing and support future analysis, this module securely handles password data:

- Secure Hashing
- Storage file

3 Exchanged Messages/Data

Effective communication between system components ensures a secure and efficient password evaluation process. Below is a breakdown of the data exchanged throughout the workflow:

- Input: User-entered password
- Internal Data: Password Features Extracted
- Output:
  - Strength score
  - Warnings
  - Improvement tips
  - Suggested password
- Stored Data:
  - Timestamp: Date and time of password submission
  - Hashed password: Hashed version using secure algorithm
  - Length: Number of characters in the password
  - Entropy Score: Computed entropy in bits
  - Strength Category: Classification (e.g., Weak, Moderate, Strong)
  - Feedback Tags: List of failed rules or notable issues (e.g., "No symbols")

4 Functional Steps

Setting accurate and clear working flows ensures a robust and secure mechanism for assessing password strength, providing user feedback, and securely storing relevant data.

4.1 Password Submission

The process begins when a user enters a password into the input field on the web interface. The password is captured by the frontend and sent to the backend for processing.

4.2 Input Validation and Pre-processing

Upon receiving the input, the system conducts initial checks:

- Ensure the input is not empty or purely whitespace.
- Sanitize input to prevent code injection (if submitted via HTTP).
- Normalize formatting if necessary (e.g., trimming leading/trailing spaces).

4.3 Pattern Matching and Rule Evaluation

The password is evaluated against a predefined set of security rules inspired by OWASP guidelines:

- Length check (e.g., < 8 = weak; > 16 = strong).
- Use of uppercase and lowercase letters.
- Inclusion of numbers and special characters.
- Avoidance of forbidden patterns (e.g., common dictionary words, names, dates, keyboard sequences).

Regex-based string pattern matching is applied to detect violations or weaknesses.

4.4 Entropy Calculation

To measure the unpredictability of the password, entropy is calculated using a formula based on character pool variety and length. The entropy is given by Entropy $= \log_2(R^L)$ where:

- R = size of the character set used
- L = length of the password

The resulting entropy score gives a quantitative indication of strength, which is categorized into levels such as:

- Very Weak (¡ 28 bits)
- Weak (28–35 bits)
- Moderate (36–59 bits)
- Strong (60–127 bits)
- Very Strong (128 bits)

4.5 Feedback Generation

Based on rule evaluation and entropy score, the system generates:

- A color-coded strength indicator (e.g., red/yellow/green bar)
- Descriptive feedback highlighting the password's strengths and weaknesses
- Tips for improvement (e.g., "Add more characters", "Include symbols", etc.)
- A suggested strong password generated using random passphrase logic

4.6 Secure Hashing and Storage

The system hashes the password using a strong cryptographic hashing function (e.g., bcrypt or Argon2). Plaintext passwords are never stored. The data is stored in a .csv file lightweight database for future analysis while maintaining user privacy.

4.7 Logging and Analysis (Future Feature)

Stored password metadata may later be used for analytical purposes such as:

- Detecting common weak patterns (while keeping user input private)
- Training a machine learning model to classify passwords
- Identifying password reuse trends (by comparing hashes)

5 Roles / Users

The Password Strength Tester system is designed with two distinct types of users, each with specific responsibilities and interactions within the system.

5.1 End User

The End User represents any individual who wishes to test the strength of a password. Responsibilities and Interactions:

- Accesses the web-based interface.
- Inputs a password for strength evaluation.
- Receives immediate feedback.
- No account or login is required.
- No personal information is collected or stored.

5.2 Developer/ Administrator

The Developer or Administrator is responsible for building, maintaining, and improving the Password Strength Tester system. This role typically involves a higher level of access and control over the system's components. Its responsibilities include:

- Implements and maintains the frontend and backend components.
- Defines password evaluation rules (e.g., regex patterns, entropy thresholds).
- Integrates secure hashing algorithms (e.g., bcrypt, Argon2) for safe password handling.
- Manages the storage of hashed password data and metadata in a csv file or database.
- Analyzes stored metadata for trends and system improvement.
- Ensures compliance with security standards and best practices (e.g., OWASP guidelines).

6 Tools and Technologies

- Frontend: HTML, CSS, JavaScript
- Backend: Python/Node.js for entropy and scoring logic
- Security Libraries: OWASP ZAP, bcrypt, Argon2
- Hashing Libraries: bcrypt, argon2-cffi, passlib
- Storage: CSV module
- Future Analysis: Power BI

7 Development Phases

The development process is broken down into practical, goal-oriented phases. Each phase builds on the previous one, ensuring modularity and testability.

7.1 Phase 1: UI Design and Basic Input Handling

- Design a clean, minimal interface (HTML + CSS).
- Implement a password input field, strength meter placeholder, and feedback section using JavaScript.
- Ensure input is captured correctly and sanitized before submission.

7.2 Phase 2: Implement Password Validation Rules

- Code the rule-based evaluator using regular expressions.
- Develop a rule-checking engine that returns flags for violations.
- Test this module with both weak and strong password examples.

8 Phase 3: Integrate Entropy and Scoring

- Implement entropy calculation logic.
- Translate raw entropy values into categories (based on bit ranges).
- Connect entropy to the scoring system already in place.

8.0.1 Phase 4: Feedback Generation

- Map feedback to failed rules (e.g., missing numbers, too short).
- Add dynamic color-coded strength indicators to the UI.
- Link textual feedback with visual elements (like warning icons).
- Suggest an auto-generated strong password (can also be a list of different passwords).

8.1 Phase 5: Add Password Hashing and Secure Storage

- Use bcrypt (via passlib or native Python module) to hash passwords.
- Structure the CSV to log essential metadata (not the raw password).
- Handle file locking and exception handling for safe writes.

8.2 Phase 6: Optional Analytics (e.g., entropy trends)

- Use Power BI tools to analyze collected metadata.
- Look for patterns in weak passwords, entropy distributions, or common feedback tags.

8.3 Phase 7: Testing and Final Adjustments

- Unit test key components (rule engine, entropy, hashing, CSV writes).
- User-test the UI and fix UX bugs (e.g., laggy feedback, poor color contrast).
- Final polish, code cleanup, and documentation.