Password Strength Analyzer

Abstract

The Password Strength Analyzer is a robust, client-side web application designed to evaluate the security and resilience of user passwords in real-time. Built exclusively with HTML, CSS, and JavaScript.The core functionality involves calculating a dynamic strength score, estimating brute-force crack time based on entropy, and critically, performing checks against common vulnerabilities. This report details the system architecture, the tools utilized, and the methodical steps taken to construct this essential cybersecurity utility.

Introduction

In the current digital landscape, weak passwords remain one of the most significant security vulnerabilities. The purpose of this project was to develop an accessible, intuitive, and highly effective tool that empowers users to proactively assess and improve their own security.

Tools Used

The entire project was implemented using fundamental and universally supported web technologies, ensuring maximum compatibility and portability. No external frameworks or backend languages were required.

- **HTML5 (index.html):** Provided the semantic structure of the application, including the main analysis input, options sections (for personal information and wordlist toggles), and dedicated displays for results and security tips.
- **CSS3 (styles.css):** Applied the dark-themed, modern aesthetic and ensured a fully **responsive design** across mobile and desktop breakpoints. It handled the visual representation of the strength score and created engaging user interface elements.
- **Vanilla JavaScript (script.js):** Contained the entire application logic, encapsulated within the PasswordStrengthAnalyzer class. This includes event handling, the core scoring algorithm, entropy-based crack time estimation, and local data persistence via the browser's localStorage API.

Steps Involved in Building the Project

The project was developed following a structured process, moving from foundational architecture to complex logical implementation and user interface integration.

1. Interface and Structure Setup

The initial step involved setting up the three core files: index.html, styles.css, and script.js. The HTML was structured using semantic divisions for the input area, analysis options, results display, and security tips. CSS was concurrently developed to establish a clear, professional visual hierarchy and ensure responsiveness.

2. Core Analysis Logic Development

The central PasswordStrengthAnalyzer class was defined. This involved creating the three critical methods:

1. **calculateStrengthScore(password):** A proprietary function that assesses length bonuses and character variety (uppercase, lowercase, numbers, symbols) and applies significant penalties for common patterns (sequences, repetitions, keyboard runs).
2. **calculateCrackTime(password):** Calculates password entropy using the formula: $\text{Combinations} = \text{Charset Size}^{\text{Length}}$. This is then divided by an estimated modern crack speed (e.g., one billion attempts per second) to yield a time-to-crack metric.
3. **extractCommonPatterns(password):** Utilized complex **Regular Expressions (Regex)** and custom logic to detect specific vulnerabilities, such as 123, abc, qwerty, or repetitive characters like aaa.

3. Custom Security Checks Implementation

To provide deeper, context-aware analysis, two advanced checks were integrated:

- **Personal Info Check:** A form was created to allow users to input personal identifiers (name, nickname, pet name). The checkPersonalInfo method compares the input password against these values and substrings, deducting score points for matches.
- **Common Wordlist Check:** A static list of over 100 common, weak passwords was embedded in script.js. The application can toggle a check against this list, penalizing the score if a match is found.

4. User Experience and Data Persistence

The final phase focused on user interaction and data handling:

- **Event Listeners:** Input events on the password field were linked to the analyzePassword method, ensuring **real-time updates**. Button clicks were configured for toggling visibility and options.
- **Dynamic Output:** The results were dynamically rendered to the DOM, including score color-coding (red, orange, green) to visually represent strength levels.
- **Local Storage:** The saveSettings and loadSavedData methods were implemented using localStorage to persist user-defined options (personal info, wordlist preference) and the **Analysis History** across sessions.

Conclusion

The Password Strength Analyzer successfully delivers a practical and educational tool for cybersecurity awareness. By combining fundamental web development principles with a sophisticated, multi-factor analysis algorithm, the project fulfills its goal of helping users move beyond simple length requirements to understand the importance of complexity, randomness, and contextual pattern avoidance.