YENEPOYA INSTITUTE OF ARTS, SCIENCE AND COMMERCE MANAGEMENT

Final Project Report

on

Password Cracking Resistance Analyzer

Team members:

Akshay R – 22BSCFDC06
Muhammed Ismail Salim –22BCACDC45
Athuljith Krishna S B – 22BSCFDC09
Akhil Shan – 22BCACDC08
Sivanand Rajesh – 22BCACDC66

Guided by:

Shashank Korram

TABLE OF CONTENTS

Executive Summary:

The **Password Cracking Resistance Analyzer** is a Node.js-based web application designed to evaluate the strength and security of user passwords in real time. The project aims to enhance user awareness around password hygiene by providing instant feedback on password strength and potential vulnerabilities.
The system combines multiple evaluation mechanisms to deliver a comprehensive analysis:

- **Common Password Detection**: Cross-references input against a database of widely used passwords to identify easily guessable entries.
- **Rule-Based Checks**: Validates passwords for complexity, including length, use of uppercase/lowercase letters, numbers, and special characters.
- **Strength Estimation**: Utilizes the zxcvbn password strength estimator to provide a numerical strength score and detailed feedback.
- **Data Breach Lookup**: Integrates with the "Have I Been Pwned" API to check if the entered password has appeared in known data breaches.

This tool serves both educational and practical purposes, offering users a clear understanding of password security best practices and encouraging the use of strong, breach-resistant credentials. The application is lightweight, open-source, and ideal for integration into broader authentication systems or cybersecurity awareness initiatives.

1 Background

1.1 Aim

The aim of this project is to create a simple web-based tool that checks the strength of passwords.
It helps users identify weak or commonly used passwords and gives real-time feedback.
The tool also checks if a password has been exposed in data breaches using the Have I Been Pwned API.
It uses the zxcvbn library to estimate how easy it is to crack the password.
The goal is to help users create stronger and more secure passwords.

1.2 Technologies

- **Node.js –** Backend runtime environment for handling server-side logic.
- **Express.js –** Web framework used to build the server and handle routes.
- **HTML/CSS/JavaScript –** For building the frontend user interface.
- **zxcvbn –** A JavaScript library used to estimate password strength.
- **Have I Been Pwned API –** Checks if a password has appeared in known data breaches.
- **Common Password Dataset –** A text file containing commonly used passwords for comparison.

1.3 Hardware Architecture

• Programming Language: JavaScript (Node.js)
• Framework: Express.js
• Libraries Used: zxcvbn, Axios
• API Used: Have I Been Pwned API
• Frontend: HTML, CSS, JavaScript
• IDE: Visual Studio Code
• Version Control: GitHub

1.4 Software Architecture

• Node.js (v14+) runtime environment
• Express.js web framework
• Password strength estimation: zxcvbn library
• External API: Have I Been Pwned for breach data
• Frontend: Static HTML, CSS, and JavaScript files
• Data input: User-entered passwords via web interface
• Output: Real-time password strength feedback and breach status

2 System

2.1 Requirements

2.1.1 Functional requirements

• Accept user password input through a web interface
• Check password against a list of common passwords
• Analyze password strength using the zxcvbn library
• Verify if the password appears in known data breaches via Have I Been Pwned API

• Provide real-time feedback on password strength and breach status
• Display results clearly on the user interface

2.1.2 User requirements

• System must be simple and easy to use for all users
• Password strength feedback must be clear and understandable
• Real-time response to password input is required
• Users must be informed if their password has been compromised
• Interface should work smoothly on both desktop and mobile devices

2.1.3 Environmental requirements

• Node.js runtime environment installed
• Required Node.js libraries (Express, zxcvbn, Axios) installed
• Internet connection for accessing Have I Been Pwned API
• Web browser for accessing the frontend interface
• File system read/write permissions for storing common password list

2.2 Design and Architecture

• **Data Layer:** Common password list file and user password input
• **Processing Layer:** Node.js backend with password strength analysis and breach API integration
• **Storage Layer:** Temporary in-memory data (no persistent storage)
• **Presentation Layer:** Web frontend built with HTML, CSS, and JavaScript showing real-time feedback

2.3 Implementation

• Set up Node.js and Express server to handle requests
• Loaded common password list from a text file for quick lookup
• Integrated zxcvbn library to analyze password strength
• Connected with Have I Been Pwned API to check password breach status
• Created frontend form to accept user password input and display results
• Provided real-time feedback based on password complexity and breach check

2.4 Testing

2.4.1 Test Plan Objectives

- To validate basic functionality, speed, and integration of password strength checks.
- Ensure that the frontend and backend interact correctly.

2.4.2 Data Entry

- Passwords were tested through API requests from the frontend.
- Various sample inputs were used including weak, strong, and leaked passwords.

2.4.3 Security

- Not formally tested.
- Basic architecture separation (frontend/backend) was followed to reduce exposure.

2.4.4 Test Strategy

- Manual validation through console logs, API responses, and sample user interactions.

2.4.5 System Test

- Checked API integration between frontend and backend using POST /check-password.

2.4.6 Performance Test

- Not performed. However, API handled dictionary checks and HIBP calls efficiently during informal testing.

2.4.7 Security Test

- Not conducted.

2.4.8 Basic Test

- Ensured password validation rules triggered expected warnings and feedback.

2.4.9 Stress and Volume Test

- Not performed.

2.4.10 Recovery Test

- Not applicable as no persistent storage or session management involved.

2.4.11 Documentation Test

- Code is clean and readable with meaningful variable names and structure.

2.4.12 User Acceptance Test

- Not performed with actual users.

2.4.13 System

- Verified smooth interaction between static frontend files and Express.js backend.

2.5 Graphical User Interface (GUI) Layout

- The web interface includes a single input field for entering a password.
- A "Check" button triggers the analysis by sending the password to the backend.
- The result displays:
    - Password strength score (0–4)

    - Estimated time to crack
    - Warnings (e.g., use of common passwords, missing character types)
    - Suggestions for improvement
- The layout is clean and minimal, focused on user clarity and speed.

2.6 Customer testing

- **Not performed.**
- No actual users were involved in testing or providing feedback.
- All testing was performed by the developer manually.

2.7 Evaluation

2.7.1 Table

Test Case	Expected Result	Actual Result
Short password	Minimum length warning	Passed
All-lowercase password	Uppercase & symbol warnings	Passed
Common password (e.g. "123456")	Dictionary warning shown	Passed
Leaked password	Breach count message	Passed
Strong password	High score, no warnings	Passed
API response time	< 1 second	Passed

2.7.2 STATIC CODE ANALYSIS

- No static code analysis tools (e.g., ESLint, Prettier) were used.
- Code was reviewed manually for basic readability and correctness.
- Future improvements could include adding linting tools for consistency.

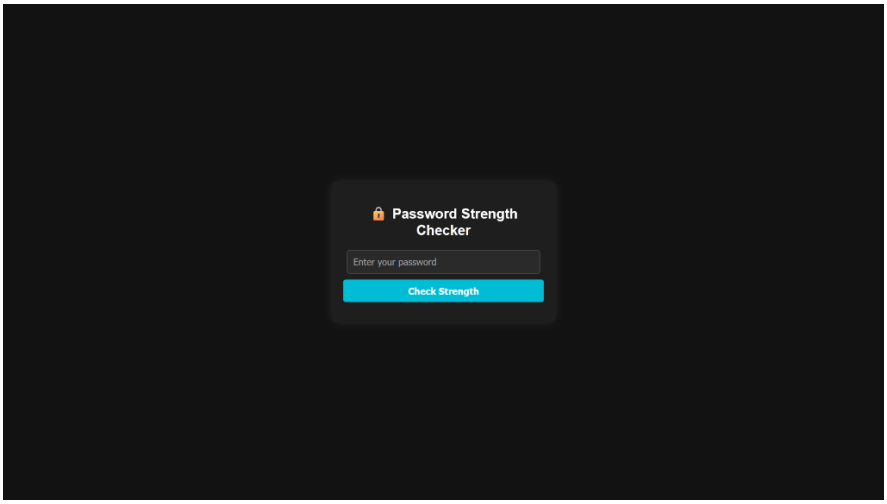
2.7.3 WIRESHARK

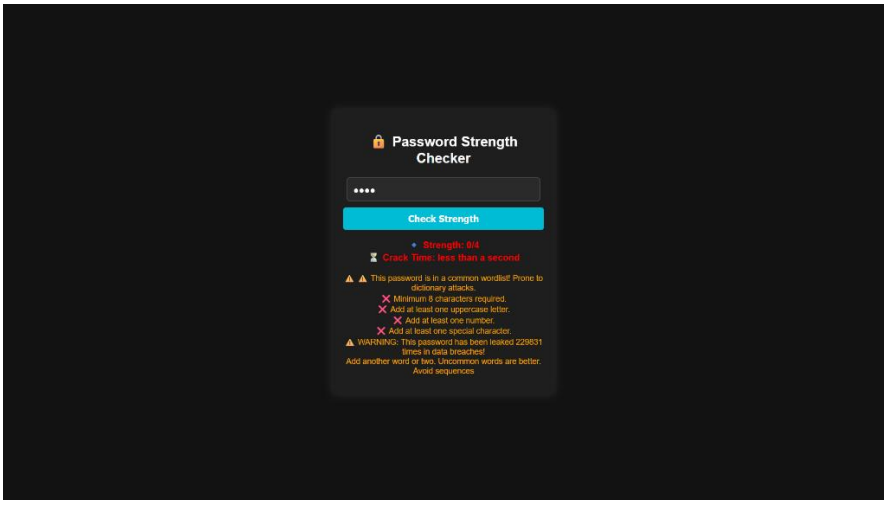
- **Not used.**
- No network packet analysis was performed.
- As the system is local and not hosted online, this was deemed unnecessary.

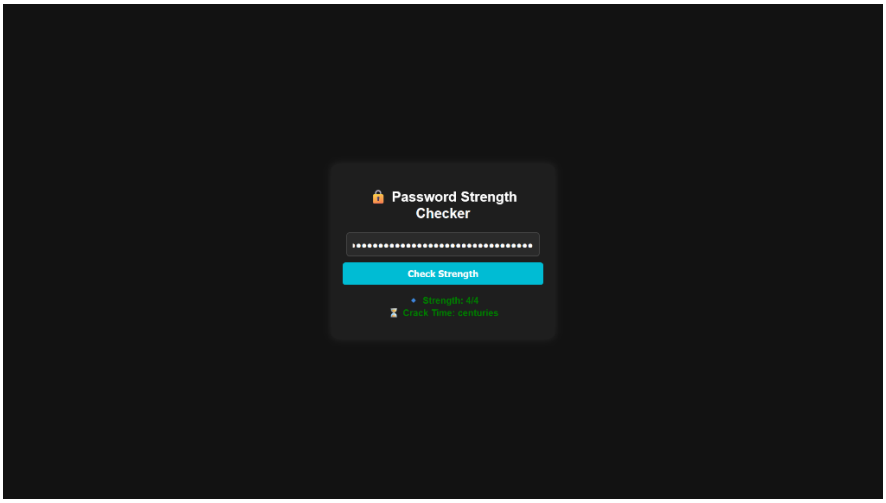
2.7.4 TEST OF MAIN FUNCTION

- The main password checking logic (/check-password endpoint) was tested manually.
- Sample passwords of varying complexity were sent via the frontend.
- No unit tests or automation were used.
- The logic successfully returns structured results as expected.

3 Snapshots of the Project







4 Conclusions

The *Password Cracking Resistance Analyzer* successfully demonstrates a functional, real-time tool for evaluating password strength and vulnerability. It effectively integrates multiple password assessment methods—common password detection, complexity analysis using the zxcvbn library, and breach lookup via the Have I Been Pwned API—to offer users instant and actionable feedback.

The project met its objectives by providing a simple yet powerful interface for users to understand and improve their password practices. Through manual testing, the system proved reliable in identifying weak or compromised passwords and suggesting ways to strengthen them.

Although advanced testing methods like stress testing and user acceptance testing were not performed, the application functioned smoothly in controlled conditions. The codebase, while functional, would benefit from enhancements such as automated testing, static code analysis, and expanded UI features in future iterations.

In conclusion, this project contributes meaningfully to the field of cybersecurity awareness by empowering users to make informed decisions about their digital security. With further development and user testing, it has strong potential for broader adoption.

5 Further development or research

Future improvements for the *Password Cracking Resistance Analyzer* could include:

- **User Accounts** for tracking password history and personalized feedback.
- **Mobile App Version** to improve accessibility.
- **Advanced Security Checks** like entropy and pattern detection.
- **Machine Learning Integration** for smarter strength prediction.
- **Multilingual and UI Enhancements** to reach a broader audience.
- **Offline Mode** to reduce dependency on external APIs.
- **Formal Testing** and user feedback collection for real-world validation.
- **Code Quality Tools** like ESLint for better maintainability.

These upgrades would enhance usability, accuracy, and security, making the tool suitable for wider deployment.

6 References

- Dropbox. *zxcvbn: Low-Budget Password Strength Estimation*.
  GitHub Repository: https://github.com/dropbox/zxcvbn
- Troy Hunt. *Have I Been Pwned API Documentation*.
  Website: https://haveibeenpwned.com/API/v3
- Node.js Official Documentation
  Website: https://nodejs.org/en/docs
- Express.js Web Framework
  Website: https://expressjs.com
- Mozilla Developer Network (MDN). *HTML, CSS, and JavaScript Guides*.
  Website: https://developer.mozilla.org
- Common Password List Dataset
  Source: https://github.com/danielmiessler/SecLists
- Axios – Promise based HTTP client for the browser and Node.js
  GitHub Repository: https://github.com/axios/axios

7 Appendix

- server.js – Node.js backend script for handling password checks

- public/index.html – Frontend HTML page
- public/script.js – JavaScript logic for sending requests and showing results
- public/styles.css – Stylesheet for UI design
- passwords.txt – List of commonly used passwords for comparison
- zxcvbn.js – Password strength estimation library
- README.md – Setup instructions and project overview