



# WIFI PASSWORD STRENGTH ANALYZER

PROJECT REPORT

**Website Link-** <https://vinayakiit.github.io/wifi-password-strength-analyzer/>

VINAYAK

BSC COMPUTER SCIENCE AND DATA ANALYTICS

IIT PATNA

Vinayak\_2312res736@iitp.ac.in

## HOW TO USE MY TOOL

### 1. **Open the Webpage**

Open the WiFi Security Analyzer web page in your browser.

### 2. **Enter Your Password**

Type or paste your current WiFi password into the input field.

### 3. **Toggle Password Visibility (Optional)**

Click the eye icon button to show or hide the password you entered for easier verification.

### 4. **Analyze Security**

Click the Analyze Security button to check your password strength.

### 5. **View Your Results**

After analysis:

- The strength meter displays your overall password score.
- Icons indicate which strength factors you meet or lack.
- The Potential Vulnerabilities section informs you if your password is predictable or common.
- Improvement Tips offer actionable advice.
- The Security Analysis bar chart breaks down your password's characteristics.

### 6. **Improve Your Password**

Use the feedback and tips to modify your password for better security. Then re-test as needed

## Overview

The WiFi Security Analyzer is a full-stack web application developed to assist users in evaluating, understanding, and enhancing the strength of their WiFi passwords. Recognizing that weak or predictable passwords are a primary vulnerability in home and organizational networks, this project provides a practical solution to promote better password practices.

The application offers a modern, intuitive frontend built with HTML, TailwindCSS, and JavaScript, featuring a responsive interface that guides users through password assessment. The frontend presents real-time visual feedback, including color-coded strength meters, actionable improvement tips, and dynamic charts using Chart.js to help users interpret the results at a glance. At the core of the analysis is a robust Python Flask backend. When a user inputs a password, the backend computes a comprehensive security score by evaluating multiple factors:

- **Password Length:** Emphasizes longer passwords for greater security.
- **Character Complexity:** Checks for diverse character sets including uppercase, lowercase, numbers, and symbols.
- **Entropy Estimation:** Calculates Shannon entropy to estimate a password's unpredictability.
- **Common Password Detection:** Flags passwords commonly found in breach lists.
- **Pattern Analysis:** Identifies sequential or repeated characters that reduce password strength.

All processing is performed securely on the backend; no passwords are stored, logged, or transmitted to third-party services, ensuring complete user privacy. The modular design of the system fosters maintainability and extensibility, allowing for the future addition of advanced features such as real-time breach checks or more sophisticated strength algorithms. By combining user-friendly design with sound security engineering, the WiFi Security Analyzer empowers individuals to take control of their network's security with immediate, actionable guidance

---

## Goals

### 1. Enhance WiFi Security Awareness

- Educate users on the importance of strong, complex passwords for protecting wireless networks.
- Highlight common vulnerabilities associated with weak, predictable, or reused passwords.

### 2. Provide a Comprehensive Password Assessment Tool

- Offer an easy-to-use platform for users to evaluate the strength of their WiFi passwords based on industry-standard criteria (length, complexity, entropy, and known patterns).
- Deliver instant feedback using visual indicators (strength meters, charts) and detailed analysis.

### 3. Deliver Actionable Password Improvement Suggestions

- Supply users with clear, personalized tips to improve password quality, such as increasing length, adding character variety, or avoiding patterns.

### 4. Ensure User Privacy and Security

- Conduct all password analysis locally on a secure backend, ensuring no passwords are stored, logged, or transmitted to third parties.

### 5. Demonstrate Sound Security Engineering

- Implement robust, well-documented analysis logic on the backend to objectively evaluate password strength.
- Use modular, extensible code architecture to support future enhancements and integrations (such as breach checks or advanced strength estimators).

### 6. Foster Accessibility and Ease of Use

- Design a responsive, accessible user interface that works across devices and is navigable for all users, including those with assistive needs.

### 7. Encourage Best Practices in Password Management

- Promote awareness of password reuse risks and promote best practices through educational feedback and resources.

## Methodology

The development of the WiFi Security Analyzer followed a structured, modular, and security-conscious approach to ensure usability, accurate analysis, and user privacy. The methodology can be outlined as follows:

### 1. Requirements Analysis

- Assessed common user needs for WiFi password security.
- Identified standard password vulnerabilities (length, complexity, predictability, patterns).
- Decided to separate the system into a frontend UI and a backend analysis engine for security.

### 2. Technology Selection

- Frontend: Choose HTML, TailwindCSS, and JavaScript for a responsive, user-friendly UI.
- Visualization: Integrated Chart.js for dynamic graphical feedback.
- Backend: Selected Python Flask for the analysis logic due to its simplicity, extensibility, and suitability for REST APIs.
- CORS: Utilized Flask-CORS to enable secure cross-origin communication between frontend and backend.

### 3. System Architecture

- Adopted a client-server model:
  - The frontend handles all user inputs, outputs, and visuals, never processing sensitive data directly.
  - The backend performs all password analysis to protect privacy and centralize logic.
- Designed a RESTful API (`/analyze`) as the communication point for password analysis requests.

## 4. Password Analysis Logic

- Implemented backend functions for:
  - Detecting presence of uppercase, lowercase, numeric, and special characters.
  - Calculating password length.
  - Estimating Shannon entropy based on character set diversity and length.
  - Identifying common passwords from a pre-defined list of weak/compromised passwords.
  - Detecting patterns like sequential or repeated characters.
  - Computing an overall strength score with weighted factors and deductions for weaknesses.
- Ensured all checks are stateless and user-specific passwords are never persisted or logged.

## 5. User Interface & Experience

- Created a clear, accessible form for password input with real-time feedback.
- Added visible indicators including color-coded meters, icons, improvement tips, and bar charts.
- Provided dynamic suggestions to help users improve password quality.

## 6. Privacy & Security

- Ensured all communication is via secure endpoints, and all sensitive operations are backend-only.
- Enabled CORS so the system can be used across hosts and domains in both development and production.
- No password data is stored, and no third-party APIs are used for analysis.

## 7. Testing and Deployment

- Performed iterative unit testing of analysis logic with a variety of real-world and synthetic password samples.
- Manual and cross-browser frontend testing for responsiveness and UI accuracy.
- Deployed backend to a public cloud platform (e.g., Render.com) and frontend to GitHub Pages.

## Future Enhancement

### 1. Integration with Public Breach Databases

- Connect to privacy-preserving APIs like HaveIBeenPwned (using k-anonymity) to let users check if their password appears in real-world breaches, while still preserving privacy.

### 2. Advanced Password Strength Estimation

- Incorporate open-source libraries such as Dropbox's (via JavaScript or Python) to provide more nuanced, real-world strength estimates and better crack-time predictions.

### 3. Support for Multi-Language and Internationalization

- Add support for password analyses in different languages and provide the UI in multiple languages to reach a global audience.

### 4. Enhanced UI and Accessibility

- Further refine accessibility (ARIA roles, screen reader support).
- Design improvements for visually impaired users and better mobile responsiveness.

### 5. Mobile App Companion

- Develop and publish an Android/iOS app version for convenient access and offline password assessment.


### 6. User Authentication & Secure Storage

- (Optional) Allow users to register securely and store their trusted WiFi passwords encrypted for future analysis or reminders.

### 7. Customizable Analysis Criteria


- Let users adjust strength score weights (e.g., prioritize length, complexity, or entropy based on user needs or organizational policy).

## Home Page

 WiFi Security Analyzer [About](#)

### Check Your WiFi Password Strength

Enter your WiFi password:



We never store or transmit your password


Analyze Security

A cybersecurity education project

## Password Input Section

### Check Your WiFi Password Strength

Enter your WiFi password:



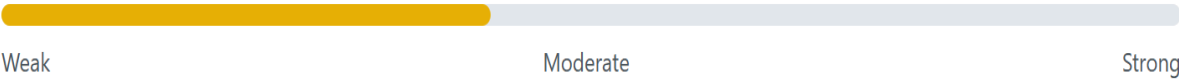
We never store or transmit your password

Analyze Security






# Password Strength Analysis Result Example




## Password Strength



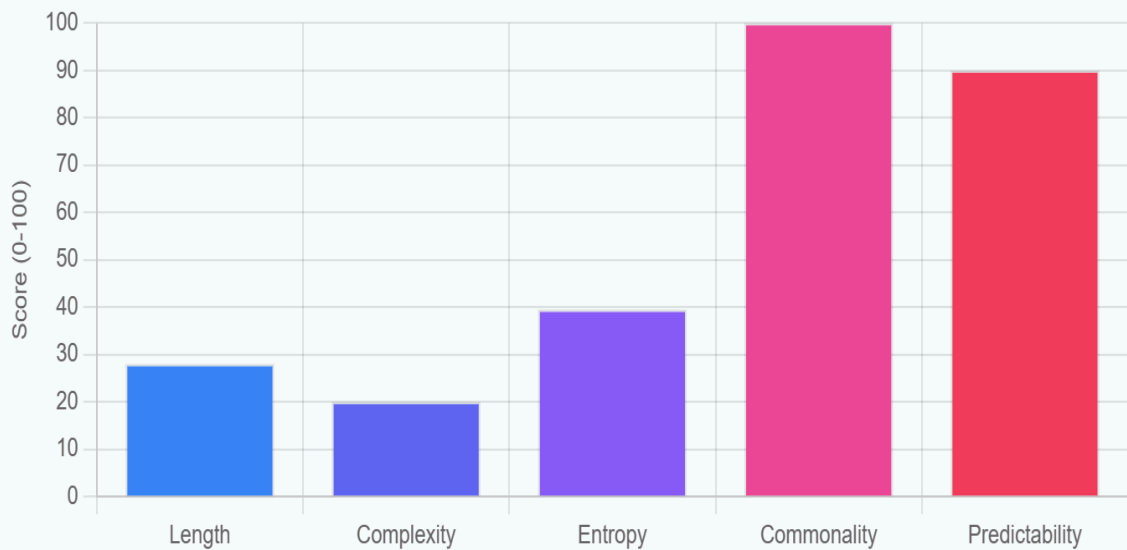
### Strength Factors

-  Length: 7 characters
-  Complexity: lowercase
-  Entropy: 32.9 bits

### Potential Vulnerabilities

-  Common Patterns: No common patterns
-  Predictability: No obvious patterns
-  Breach Check: Not found in common password lists

## Security Analysis



# System Architecture

The WiFi Password Strength Analyzer adopts a modular, client-server architecture designed for security, scalability, and clarity. The main components are the Frontend (client) and the Backend (server), communicating via a secure HTTP API.

## 1. Frontend

- Technologies: HTML, CSS (optionally TailwindCSS), JavaScript, Chart.js
- Function:
  - Provides the user interface for password entry.
  - Sends user input to the backend for analysis via an HTTP POST request.
  - Receives analysis results and dynamically updates the UI, including the strength meter, feedback tips, and bar chart visualizations.
- Hosting: Can run as a static website (e.g., GitHub Pages, Render static sites) or be served from the backend.

## 2. Backend

- Technologies: Python, Flask, Flask-CORS (for cross-origin support)
- Function:
  - Exposes an API endpoint that receives a password (via POST request).
  - Analyzes password strength based on length, variety, entropy, and detection of weak patterns (common passwords, sequential/repeated characters).
  - Calculates a strength score and returns detailed results in JSON format.
  - Does not store or log any password—stateless by design.
- Hosting: Deployed on a cloud platform (e.g., Render, PythonAnywhere), running as an always-on web service.

## 3. Communication Flow

1. User interaction: User enters WiFi password in frontend and clicks "Analyze."
2. API Request: Frontend sends the password to the backend's `/analyze` endpoint as a POST request.
3. Processing: Backend analyzes the password, computes all required metrics, and prepares a JSON response.

## 4. Security Measures

- CORS (Cross-Origin Resource Sharing): Configured so only allowed frontends can access the API endpoint.
- No password storage: Passwords are handled in-memory for analysis only—never written to disk or database.
- No hardcoded credentials: All sensitive operations are handled server-side with best practices.

## 5. Extensibility

- The backend logic is modular, allowing future integration with breach APIs or advanced strength estimators.
- The frontend components are independent, easily supporting theme and UX/UI enhancements.

[User Browser]

|



[Frontend (index.html, JS, Chart.js)]

|

(HTTP POST /analyze)



[Backend (Flask API on Render/PythonAnywhere)]

|

(JSON response with analysis)



[Frontend visualizes and informs user]

## Challenges Faced

### 1. Cross-Origin Resource Sharing (CORS) Issues

- Integrating a separate frontend and backend required configuring Flask-CORS properly to allow secure communication between the website and the API, especially when deployed on different domains or platforms.

### 2. API Endpoint Configuration

- Ensuring the frontend always pointed to the correct backend URL (local, Render, or PythonAnywhere) and handling potential differences in HTTP vs. HTTPS was critical for proper functionality and caused initial connectivity issues.

### 3. Visualization Accuracy and Clarity

- Designing clear, intuitive strength meters and bar charts required refinement to ensure users could easily understand their password's weakness or strength. Mapping backend metrics to user-friendly visual feedback (especially with Chart.js) sometimes led to confusion until further iteration.

### 4. Cloud Deployment Nuances

- Deploying the Flask backend to platforms like Render introduced issues such as environment variable handling (for ports), proper return of JSON using Flask's `jsonify`, and the need to use production-ready servers like Gunicorn instead of the Flask development server.

### 5. Ensuring Stateless and Secure Password Handling

- Guaranteeing that passwords were never stored, logged, or exposed in transit required strict adherence to stateless communication and careful code review for privacy best practices.

### 6. Frontend and Backend Synchronization

- Any change in backend logic or API contract (such as response fields) necessitated simultaneous updates in the frontend's JavaScript; mismatches led to functionality breaks until reconciled.

### 7. Testing on Different Browsers and Environments

- Making sure the app worked smoothly across browsers (Chrome, Firefox, Edge) and on both local and deployed URLs required repeated compatibility testing.

## Security and Privacy Considerations

### 1. No Password Storage or Logging

- The backend processes each submitted password in-memory and does not save, log, or persist any passwords. This minimizes the risk of accidental data exposure or breaches.

### 2. Secure Backend Processing

- All password analysis occurs exclusively on the backend server (Flask app), never in the browser's client-side code. This separation protects sensitive logic and reduces attack surfaces.

### 3. No Third-Party Transmission

- Passwords are not sent to or checked using external APIs, breach-check databases, or analytics tools. All evaluation is local to your controlled backend, ensuring user data is never exposed to untrusted services.

### 4. Stateless API Design

- Each analysis request is stateless. The backend does not track users or maintain session data, making it less vulnerable to session hijacking and ensuring user anonymity.

### 5. Cross-Origin Resource Sharing (CORS) Control

- CORS is correctly implemented using Flask-CORS, enabling secure cross-origin requests only from the intended frontend origin. This prevents unauthorized third-party sites from accessing the backend API.

### 6. No Hardcoded Credentials or Sensitive Data Exposure

- The codebase avoids the use of hardcoded admin credentials or secrets. All configuration for deployment uses environment variables if required.

### 7. HTTPS Recommendation for Public Deployment

- When deployed in production, the server and frontend should both be accessed over HTTPS to encrypt all data in transit, protecting passwords and analysis results from eavesdropping.

### 8. No User Identification or Tracking

- Users are not required to log in or provide personal details. No IP, email, or other identifiers are collected, supporting stronger privacy guarantees.