TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

AN OVERVIEW OF OUR PROJECT

Security Project Overview: Strengthening Our Future

Done By :

Farah Toumi

Molka Braham

Thouraya Harrabi

Introduction

Why FortiPass?

- Passwords are the first line of defense—but often weak.
- FortiPass is a web-based password strength checker that evaluates user-entered passwords based on OWASP guidelines.
- Built to help users create safer passwords.
- Uses OWASP standards, real-time feedback, and breach checking.
- Educates users, not just validates.

Main Concepts

Core Security Concepts Behind FortiPass :

- OWASP Password Policy Requirements: Enforces rules on length,
- character variety, common passwords, personal info.
- Password Entropy: Measures unpredictability
- Levenshtein Distance: Prevents passwords similar to
- usernames/emails.
- zxcvbn Algorithm: ML-based estimator using real-world data.
- Breach Intelligence: Uses “Have I Been Pwned” API with k-anonymity.
- Passphrase Encouragement: Promotes strong, memorable phrases

Tools Used :

Development Tools & Tech Stack:

- HTML/CSS/JavaScript or React – Frontend
- Python / Flask – Backend
- OWASP Guidelines – Rules logic
- GitHub
- VS Code – Code editor

1. Frontend (HTML/CSS/JavaScript)

index.html:

A password manager with two tabs:

Password Check: Validates password strength.

Generate: Creates secure passwords based on user preferences.

Includes input fields, templates for password generation, and security tips.

style.css:

Modern design with gradients, shadows, and responsive layout.

Features:

Gradient text for the title.

Animated buttons and input fields.

Visual feedback for password strength (color-coded bars).

app.js:

Tabs: Switches between password check and generator.

Password Check:

Sends password/email to the backend for validation.

Displays results (strength, criteria met/unmet).

Password Generator:

Creates random passwords using customizable rules (length, symbols, etc.).

Predefined templates (e.g., "Strong," "PIN").

2. Backend (Python/Flask)

- main.py:
  - Flask server with two routes:
    i. /: Serves the HTML file.
    ii. /api/check-password: Validates passwords using OWASP criteria:
    - Minimum 12 characters, mix of uppercase/lowercase/numbers/symbols.
    - Blocks common passwords (e.g., "123456"), repeating/sequential patterns.
  - Saves the last 10 password checks to password_history.json.

How it is Done ?

In the **/api/check-password** endpoint, you analyze passwords using multiple OWASP criteria:

- Minimum length (≥ 12 characters)
- Contains lowercase letters
- Contains uppercase letters
- Contains digits
- Contains special characters
- Does not contain common patterns
- Does not contain repeating characters
- Does not contain sequential patterns

Each of the above criteria gives 1 pint if satisfied. You then:

- Score the password (0–8).
- Classify it as:
  - Strong (score ≥ 6)
  - Medium (score ≥ 4)
  - Weak (otherwise)

Saved Password Evaluation History

Stored history in password_history.json.

Each entry logs:

Email address

Password strength

Score

Date/time (ISO format)

You keep only the last 10 entries in the file(json.dump(history[-10:], f)

).

implemented this with two functions:

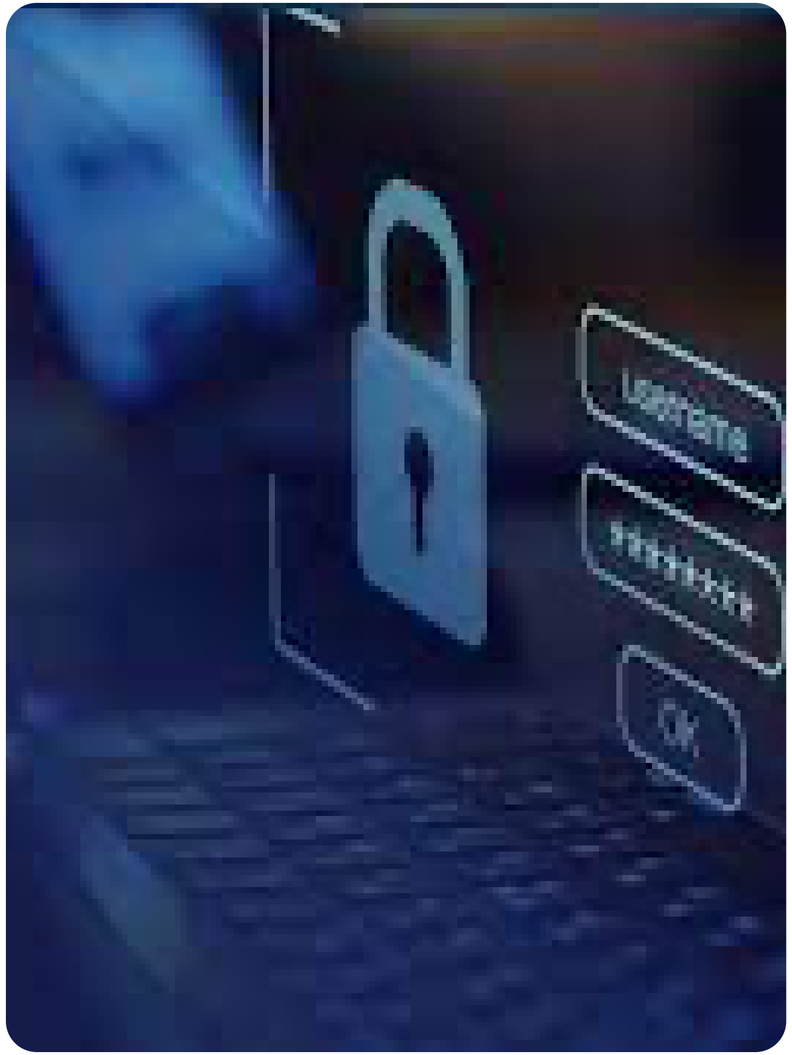
load_password_history() → Reads JSON file

save_password_history(email, strength, score) → Appends entry

& saves

This file is automatically updated each time a password is evaluated.

Functional Flow:



1.**Password Validation**
Checks password according to OWASP

2.**Frontend Features**
Password input field with show/hide toggle
Email input field
Real-time strength checking
Visual feedback on criteria met/unmet
Password generator tab with options
3.**Backend (Flask)**
/api/check-password endpoint for validation
Basic password history storage
Static file serving
Storage
Simple JSON file storing password history
Keeps last 10 password checks with email and timestamp

Python Backend Overview

Technology Used: Python (Flask Framework)
Functionality: API for password strength validation based on OWASP best practices
File: main.py

API Endpoint: /api/check-password

Purpose: Evaluate the strength of a password using OWASP-aligned checks.

Input (via POST request)

json

```
{
  "email": "user@example.com",
  "password": "Ex@mple123456"
}
```

Output (JSON Response)

json

```
{
  "score": 6,
  "strength": "Strong",
  "criteria": {
    "length": true,
    "lowercase": true,
    "uppercase": true,
    "numbers": true,
    "special_chars": true,
    "no_common": true,
    "no_repeating": false,
    "no_sequential": false
  }
}
```

Roles & Users

User:

- Enters password
- Receives score and suggestions

Admin:

- Updates policies and rules
- Manages evaluation settings

Unique Value Proposition

What Makes FortiPass Stand Out?

- ligned with OWASP password security standards
- Clear, testable API with JSON input/output
- Tracks user evaluations with timestamps
- Modular and easy to improve (e.g., add blacklist, integrate hashing, etc.)
- Clear, adaptive feedback

Conclusion

FortiPass: Empowering Secure Passwords

- OWASP-compliant, real-time password evaluator
- Combines theory (entropy, ML) with real-world data (breaches)
- Educates users and encourages safe password habits
- Simple yet powerful solution to a widespread problem

Thank you