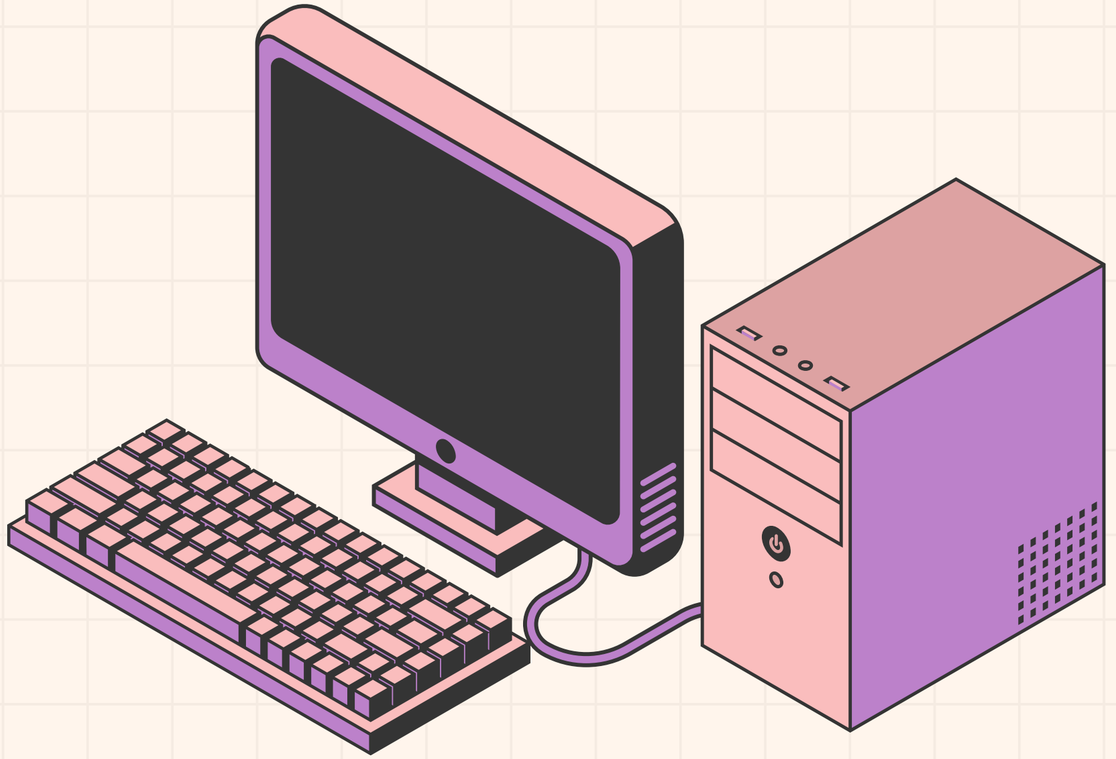
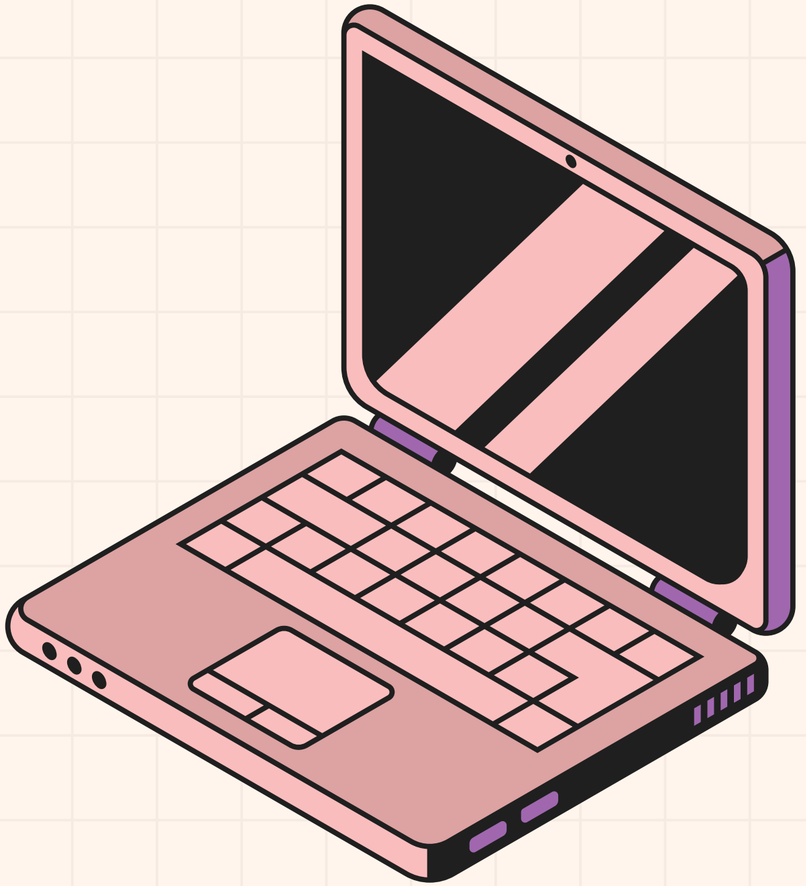
PassFort

OWASP Password Strength Tester with Keylogger Protection

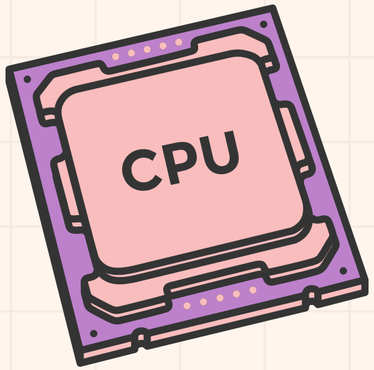


INTRODUCTION

PassFort is a comprehensive password security tool designed to help users create and maintain strong passwords. It evaluates password strength based on OWASP guidelines and checks whether the password has been compromised in known breaches. It also offers a unique feature of protection from keyloggers.

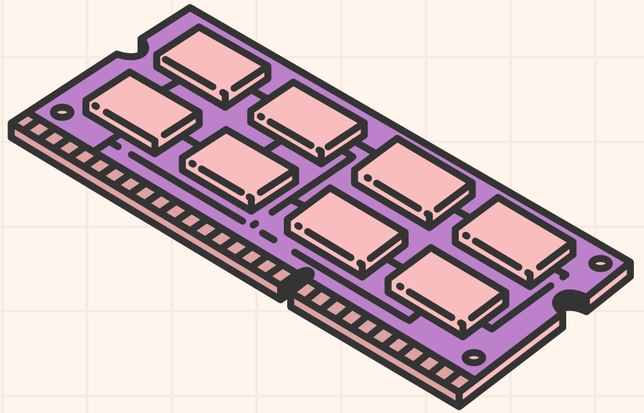


KEY FEATURES



OWASP COMPLIANCE

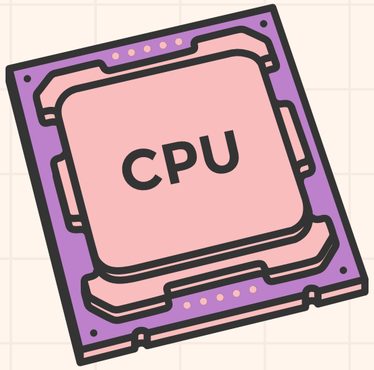
Ensures passwords meet industry-standard security guidelines.



LEAK DETECTION

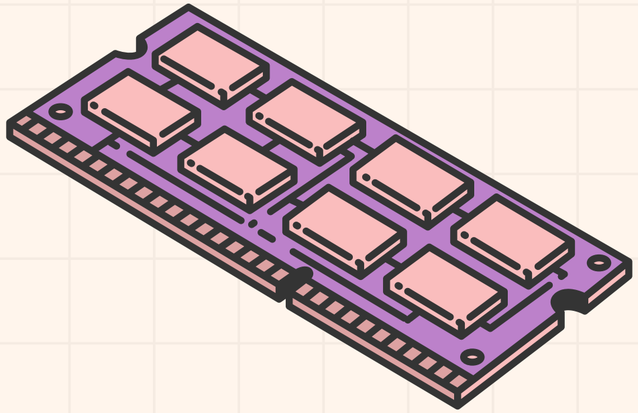
Checks if a password has been exposed in data breaches.

KEY FEATURES



SMART PASSPHRASE GENERATOR

Creates strong, memorable passphrases based on user's existing password.

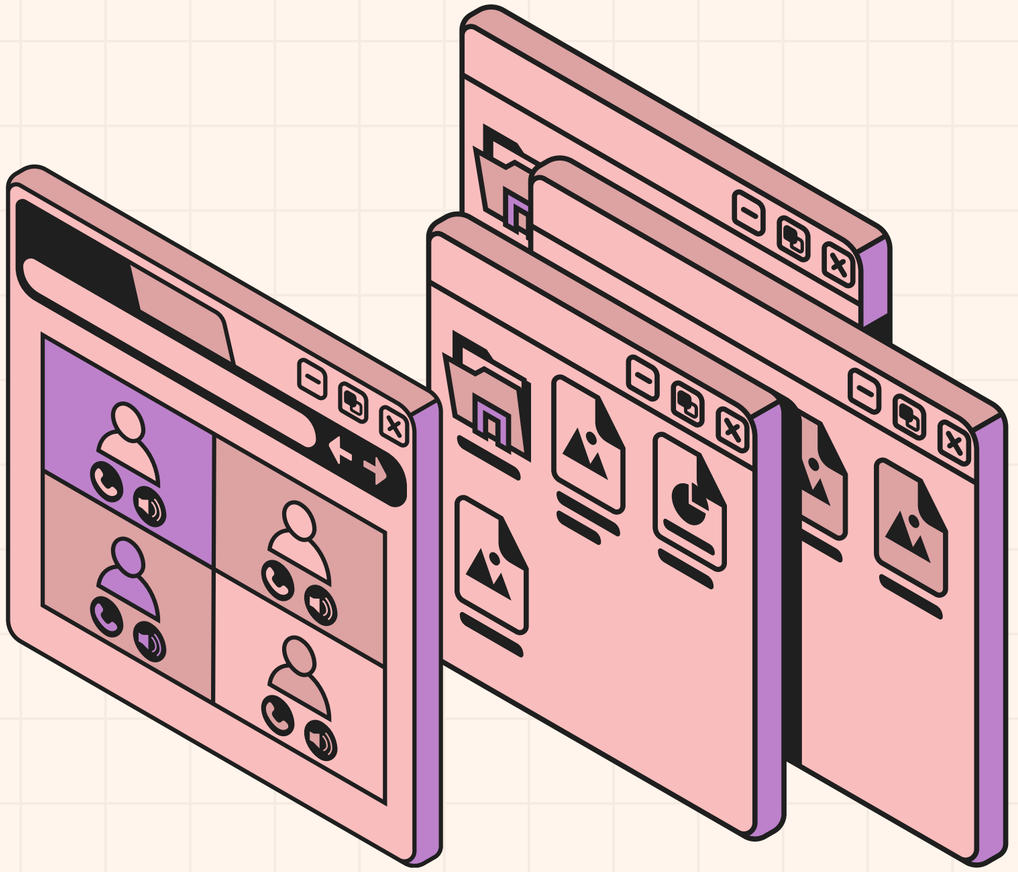


USER FRIENDLY INTERFACE

Simple and intuitive design for quick password analysis.

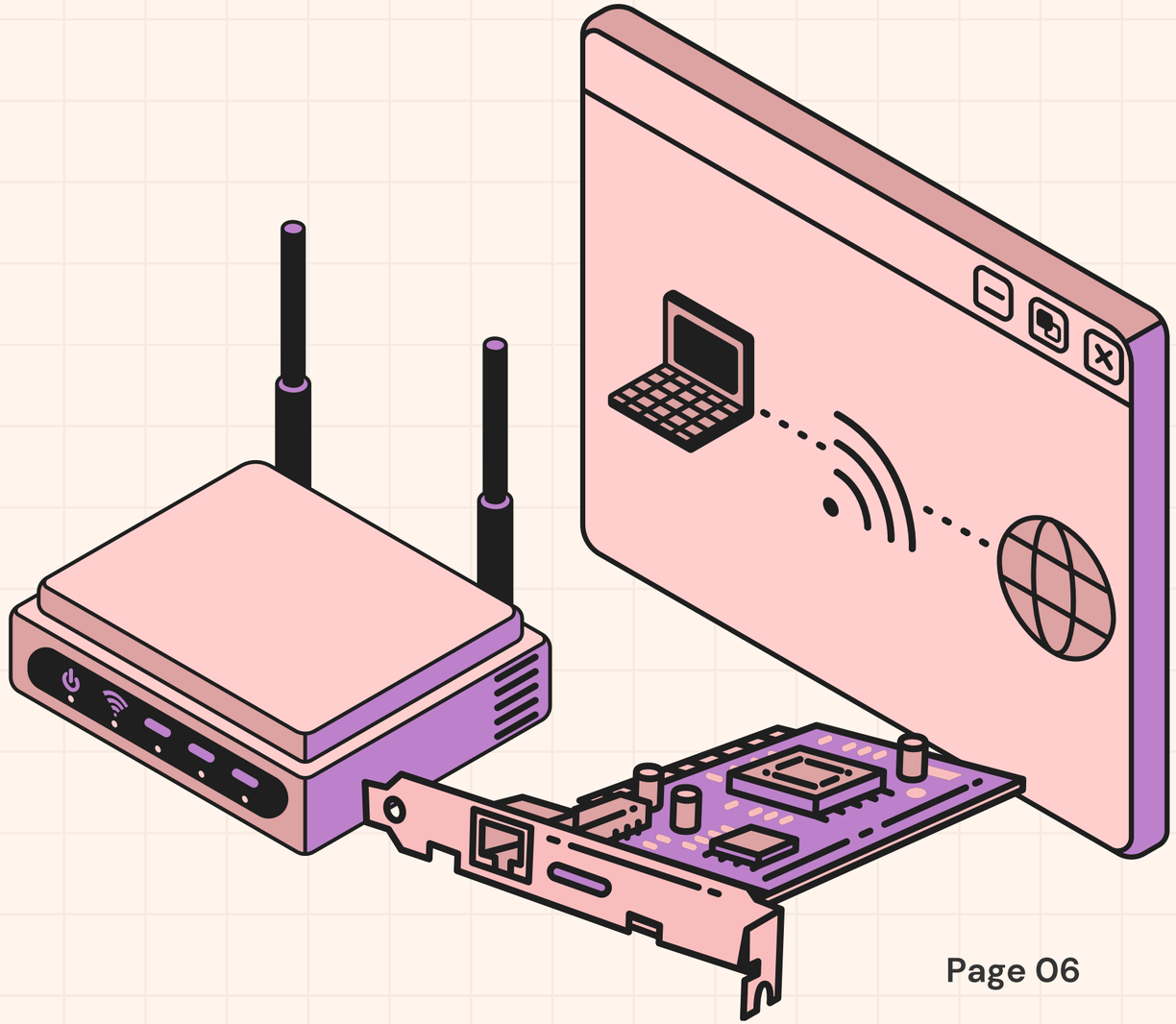
WHAT IS UNIQUE ABOUT PassFort?

Password Camouflage Mode protects users from keyloggers by inserting random decoy characters (*\) while they type their password. The decoys get automatically removed before submission.



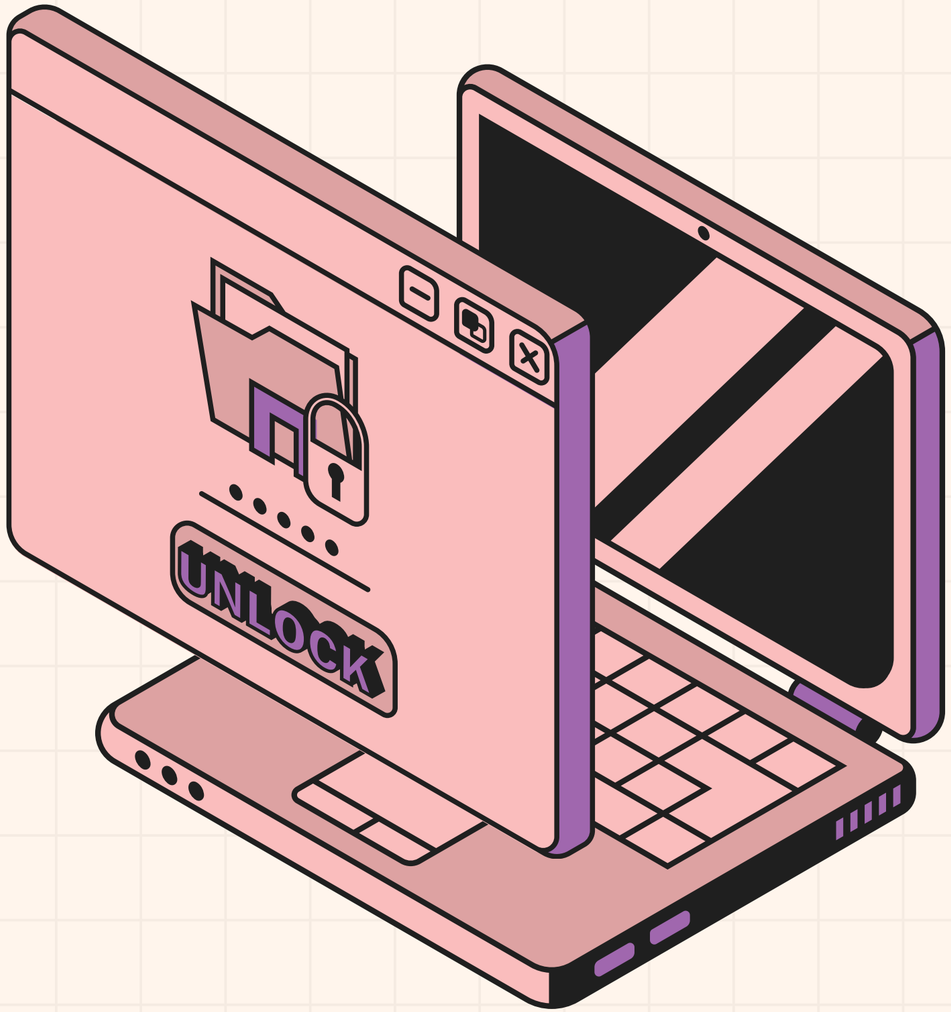
OWASP PASSWORD GUIDELINES

- Allow long passwords (at least 64 characters).
- Do not impose composition rules (e.g., requiring uppercase, numbers, or special characters).
- Block common and breached passwords.



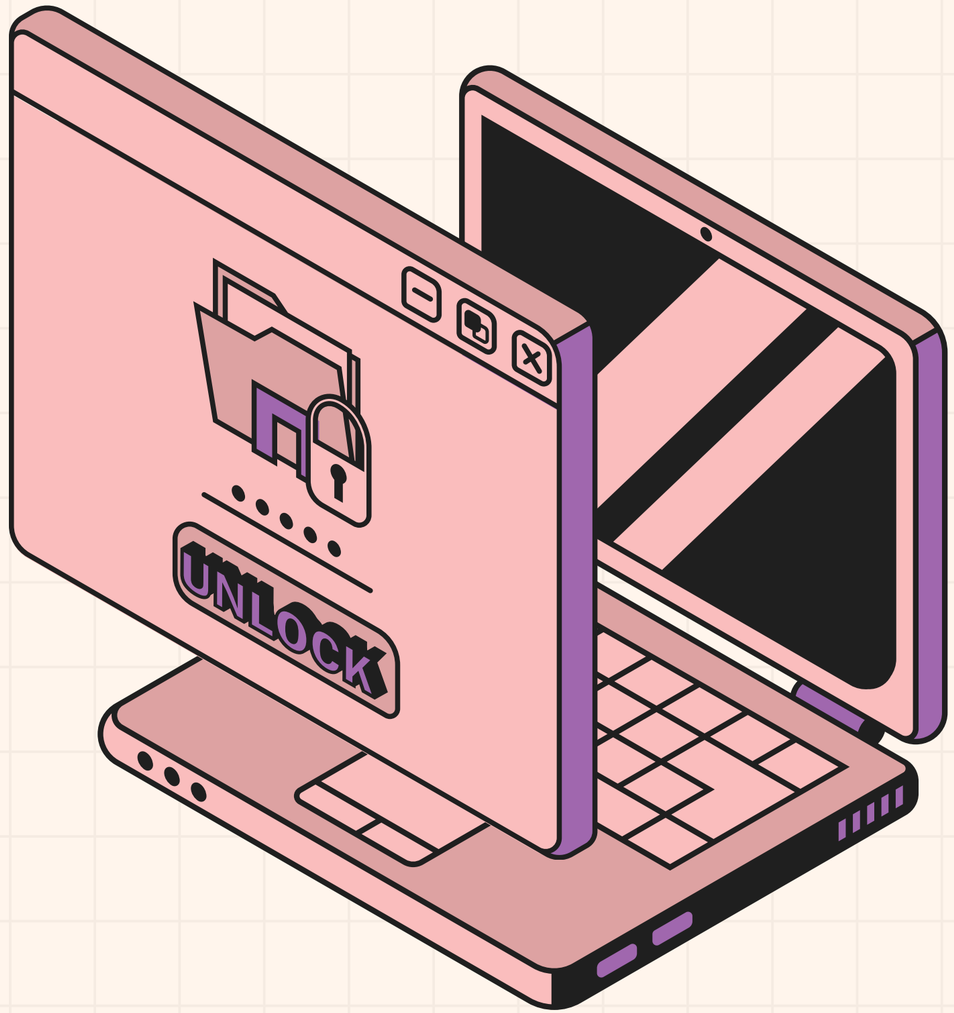
IMPLEMENTATION OF OWASP RULES

A normal python script that includes a class titled **OWASPPasswordStrengthTest** which conducts required and optional password tests as per the guidelines. If longer than 20 characters it is said to be a passphrase. As per the password's performance in the tests, it will be declared as **strong** or **weak**.



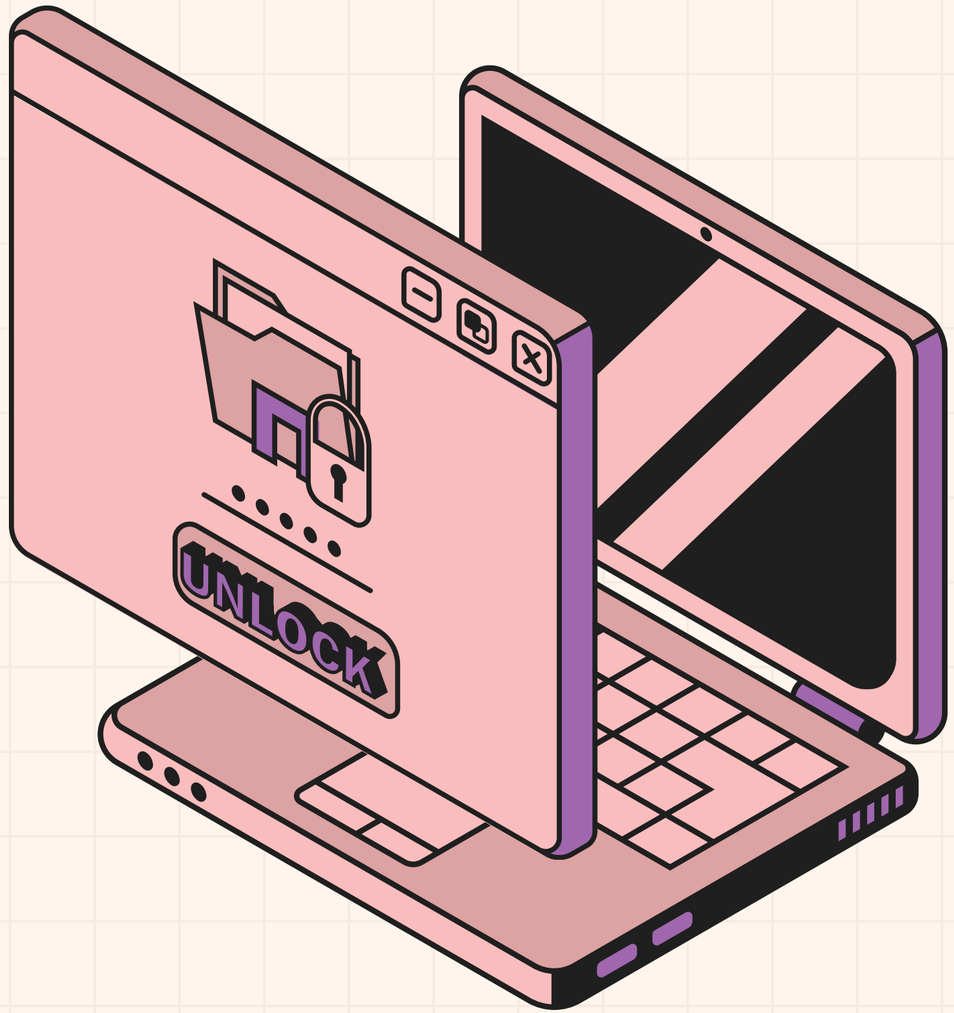
REQUIRED TESTS (MUST PASS ALL)

- Minimum length: Password must be at least 10 characters long.
- Maximum length: Password must be 128 characters or fewer.
- No repeating sequences: Password cannot have three or more consecutive repeating characters (e.g., "aaa" or "111")



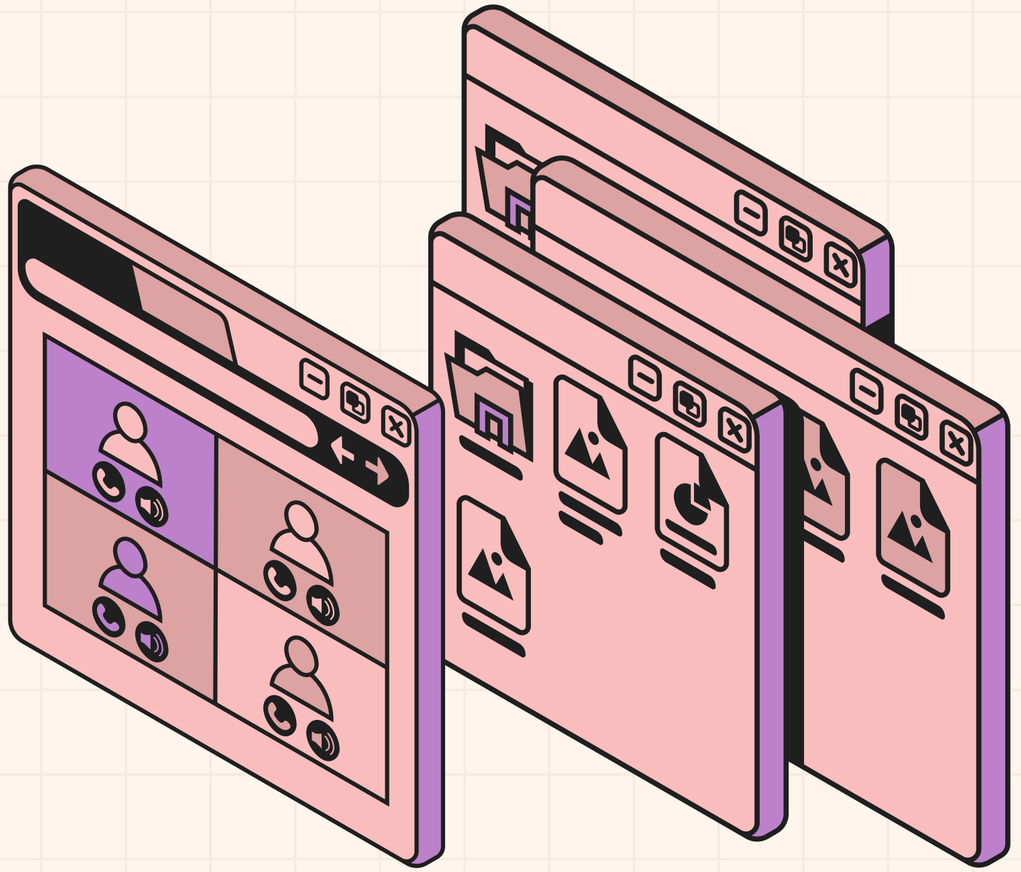
OPTIONAL TESTS (MUST PASS AT LEAST 4)

- Lowercase letters: Password must contain at least one lowercase letter (a-z).
- Uppercase letters: Password must contain at least one uppercase letter (A-Z).
- Numbers: Password must contain at least one digit (0-9).
- Special characters: Password must contain at least one special character (!@#$%^&*, etc.).



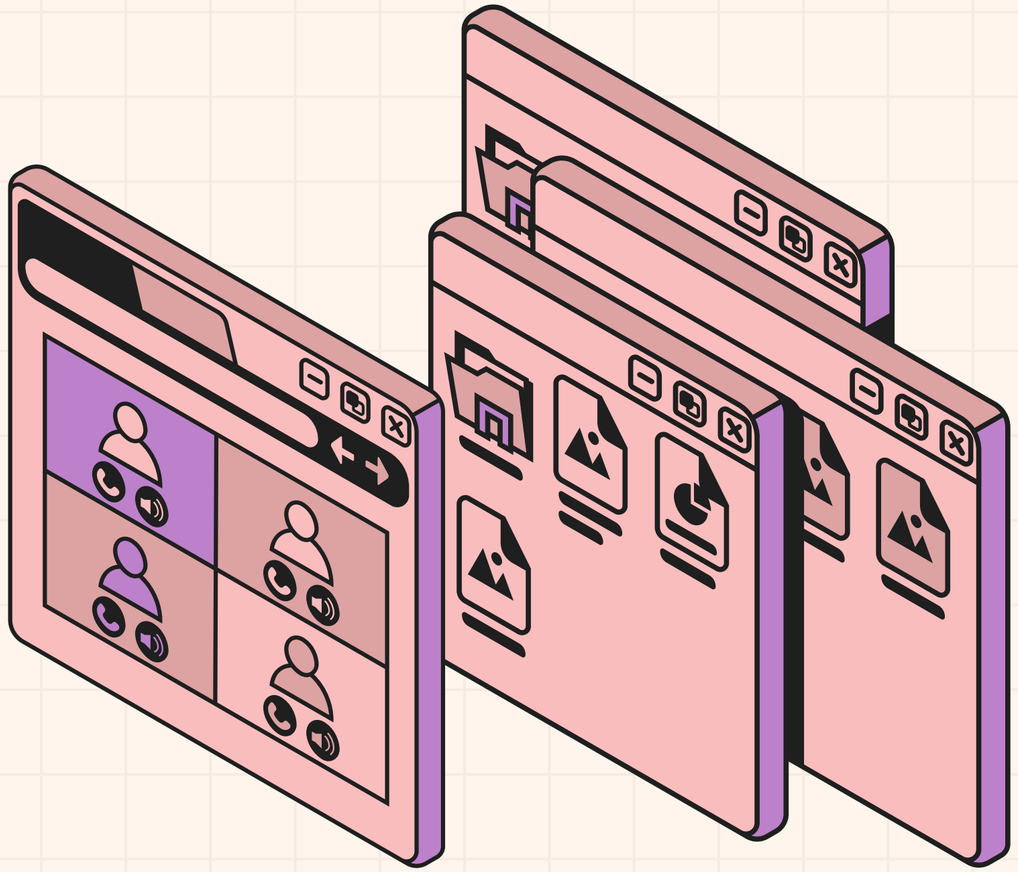
IMPLEMENTATION OF LEAK DETECTION

The entered password is checked against **rockyou.txt** a common wordlist and also against **HIBP API** which contains passwords from real world breaches. This ensures both speed (local check) and depth (API check) in detecting compromised passwords.



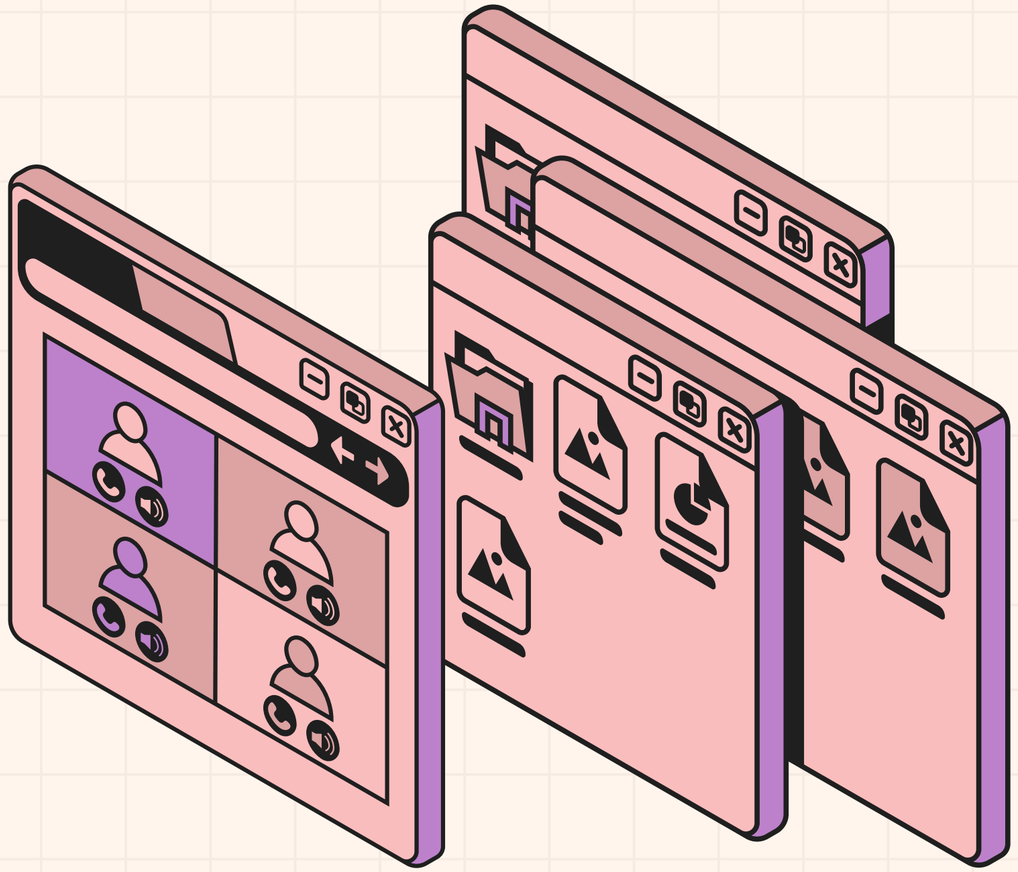
WHAT IS rockyou.txt?

rockyou.txt is a famous password list that contains **14,344,392** real-world passwords. It originated from the RockYou data breach in 2009, where millions of user passwords from a social media app were leaked — in plain text. It comes pre-installed in Kali Linux and is available in multiple GitHub repositories. It is the goto wordlist for brute forcing passwords.



WHAT IS K-Anonymity?

K-anonymity is a privacy-preserving technique that ensures any individual data point (like your password's hash) cannot be distinguished from at least K-1 other points. This means your data gets "hidden in a crowd" of similar data points. For example, if K = 1000, your data looks identical to 999 others, making it hard to identify specifically.



HAVE I BEEN PWNED API
A K-ANONYMITY APPLICATION

HASHING

The password is first hashed using SHA-1 algorithm.
eg: **password123** is hashed into **CBFDAC6008F9CAB4083784CBD1874F76618D2A97**

SPLITTING

The hash is split into a **prefix** (5 characters) and a **suffix** (remaining part). Only the suffix is sent first to the API.
eg: suffix: **CBFDA**, prefix: **C6008F9CAB4083784CBD1874F76618D2A97**

HAVE I BEEN PWNED API
A K-ANONYMITY APPLICATION

RESPONSE

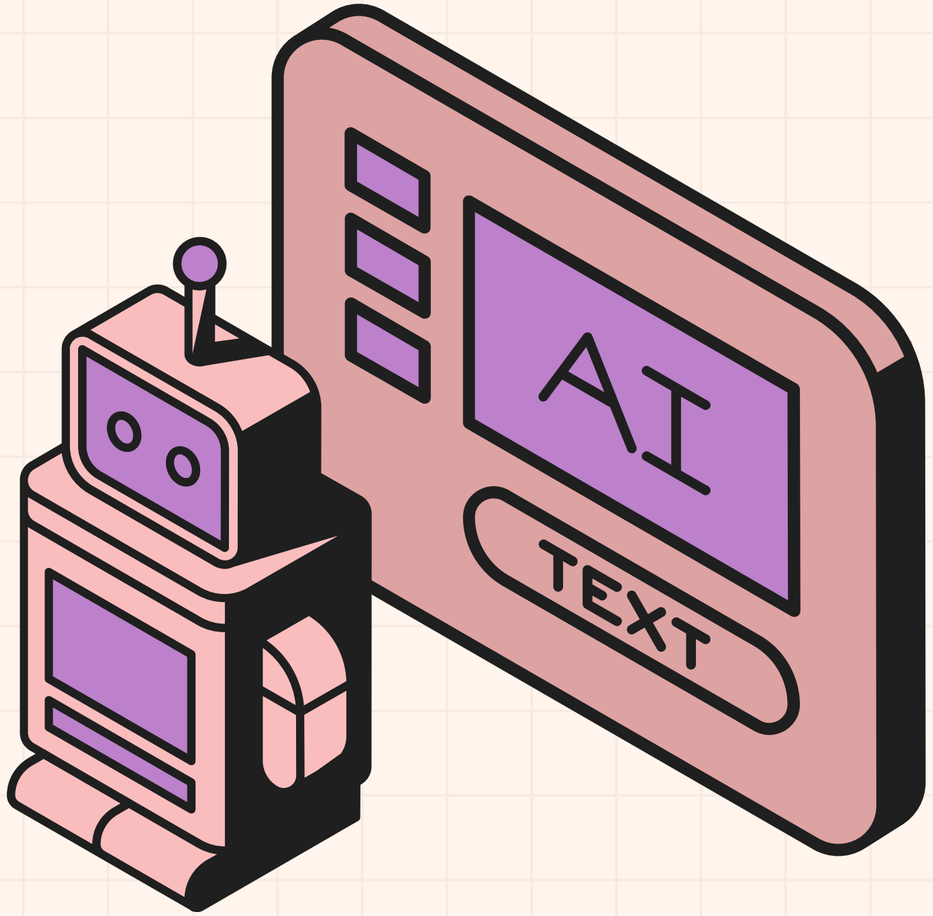
The API responds with all hashes that start with **CBFDA**, along with how many times each one has been seen in breaches. This is typically a list of **500-1000** hashes, making it nearly impossible for anyone to figure out which is the input.

COMPARING

The function checks the list locally (in the code) to see if the full suffix matches any hash from the list. If it matches, the password has been breached, and it shows the count. Otherwise, it’s safe.

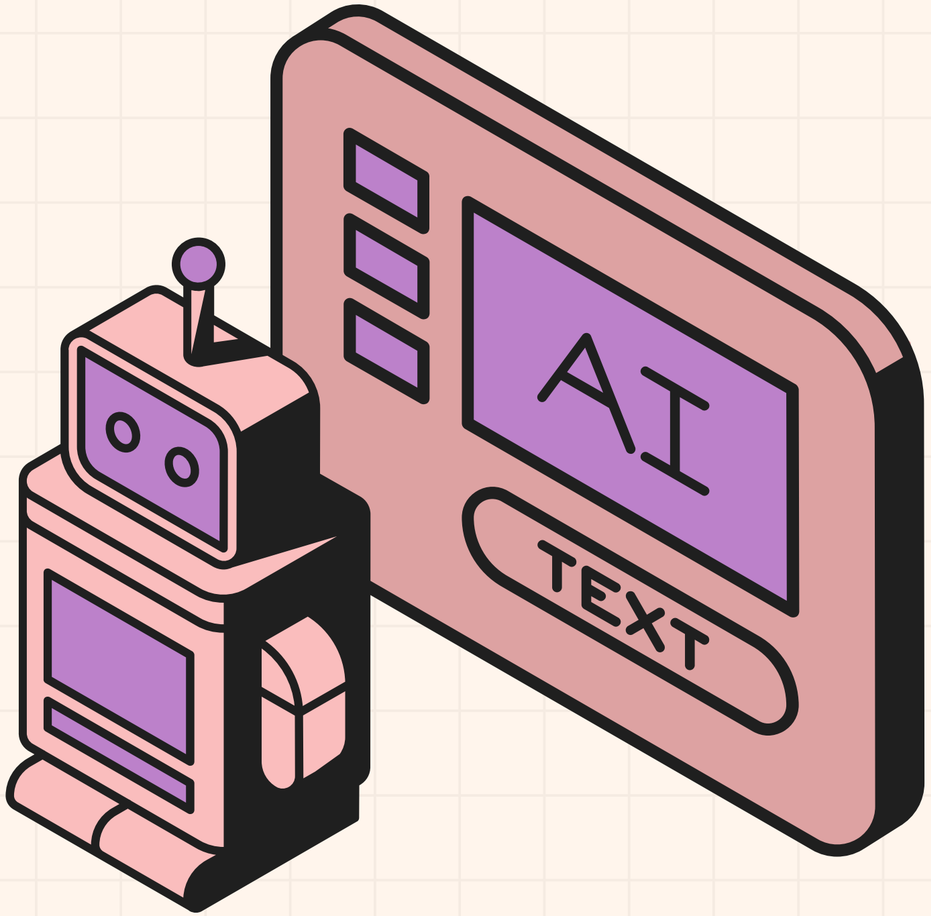
SMART PASSPHRASE GENERATOR

The smart passphrase generator aims to create strong, memorable, yet secure passwords that comply with OWASP guidelines while being resistant to breaches and dictionary attacks. The heart of the generator is the **Markov model**, which produces realistic, human-like snippets of text based on an existing corpus (dataset). In my program, the script uses *"Crime and Punishment"* from **Project Gutenberg**.



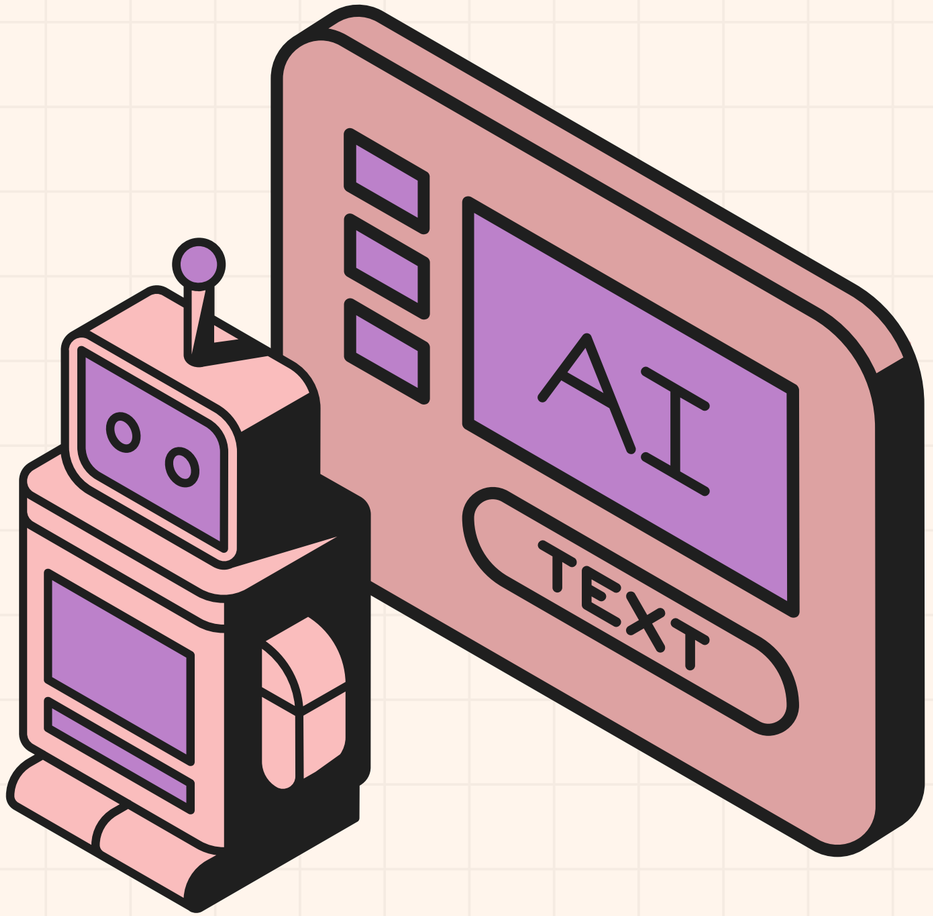
HOW DO MARKOV CHAINS WORK?

- A Markov Chain models the probability of moving from one "state" to another — in this context, a word or character to the next word/character.
- The model learns transitions: e.g., after "the," the word "cat" might follow 30% of the time, while "dog" follows 20%.
- After training on a large text, the model can generate new text snippets that mimic the original style without directly copying it.



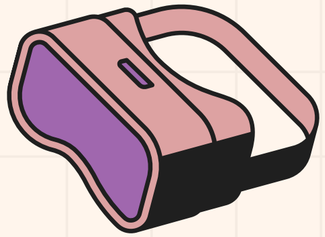
IMPLEMENTATING SMART PASSPHRASE GENERATOR

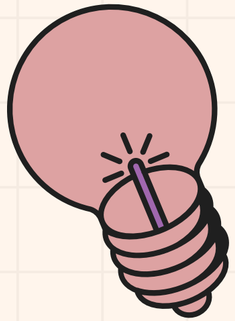
1. User inputs the password/passphrase.
2. The Markov model generates a short random snippet from the *Crime and Punishment* text. If the model fails to generate a valid snippet the script falls back to a default "BucketFishCar" to ensure functionality.
3. OWASP strength compliance.
4. Breach check with rockyou.txt and HIBP API.



USER FRIENDLY DESIGN

Using the Python Flask framework, we can create a user friendly web implementation of PassFort.





THANK YOU

COLAB IMPLEMENTATION