Title: Password Strength Analyzer with Custom Wordlist Generator

Student: Lakshya Mahani
Internship: Cybersecurity Internship — Project 4
Date: 08 September 2025]

Abstract

This project implements a GUI-based Password Strength Analyzer and a Custom Wordlist Generator. The tool evaluates passwords for strength using either the zxcvbn algorithm (if available) or a fallback entropy estimator. It also generates focused attack wordlists using user-provided hints—names, dates, and keywords—combined with leetspeak, case permutations, year suffixes, and common numeric patterns. The objective is to demonstrate both offensive password-analysis techniques (for penetration testing and red-team scenarios) and raise awareness on secure password creation for defenders.

Introduction

Passwords remain one of the most common authentication mechanisms and are often the weak link in system security. Attackers leverage personal user information and predictable patterns to craft targeted wordlists for cracking. This project shows how simple user hints can drastically reduce cracking time when turned into structured wordlists, and simultaneously demonstrates how to evaluate password robustness to mitigate such attacks.

Tools Used

- Python 3.8+ (primary language)
- Tkinter (GUI)
- Optional: zxcvbn for advanced password scoring
- Standard libraries: itertools, datetime, math
- Output: .txt wordlists compatible with popular cracking tools for testing (use ethically)

Design and Steps Involved

1. **Password Strength Module**
    - Primary approach: use zxcvbn (where available) to score passwords 0–4 and provide warnings/suggestions.
    - Fallback: entropy approximation using character-class pool size and length, with penalties for repeats. Entropy values are mapped to human-readable labels (Very Weak — Very Strong).

    - The GUI displays score, entropy in bits, and actionable feedback.
2. **Custom Wordlist Generator**
    - Input: free-text hints (comma-separated): names, pets, dates, favorite words.
    - Tokenization: split hints into base tokens (strip punctuation).
    - Variant generation:
        - Case permutations (lower, upper, title, single-cap).
        - Leetspeak substitutions using a small dictionary (a→4/@, e→3, o→0, s→5/$, etc.).
        - Pairwise concatenations (JohnDoe, DoeJohn).
        - Year appends/prepends and underscore variants for ranges (1960–current year by default).
        - Common suffixes and numeric patterns (123, 1234, !).
    - Output: unique, bounded list of candidate passwords exported as .txt.
3. **GUI**
    - Built using Tkinter for portability and simplicity.
    - Sections: password analysis, hints input, generation options, preview, and logs.
    - Non-blocking generation via a background thread to keep UI responsive.

Conclusion

This project provides a compact, user-friendly tool for both evaluating password strength and generating attack-focused wordlists for controlled testing. It highlights how easily personal information can be transformed into effective cracking candidates and stresses the need for long, unique, and random passwords (or passphrases) combined with multi-factor authentication in production environments. The GUI is suitable for demoing in interviews and can be extended with features like integration with breach databases, advanced pattern detection, or a stronger leetspeak/markov-based generator.