PASSWORD STRENGHT ANALYZER & CUSTOM WORDLIST GENERATOR

Abstract

This project delivers a single-file, easy-to-run Python solution that combines password strength analysis and customizable wordlist generation. The analyzer uses an entropy based heuristic and integrates with zxcvbn when available. The generator produces attack Style candidate passwords by transforming user-supplied personal data and standard patterns. The primary aim is to support authorized, ethical password testing for individuals and organizations, emphasizing defensive use and responsible disclosure.

Introduction

The Password Strength Analyzer & Custom Wordlist Generator is a compact Python application designed to help users evaluate the strength of passwords and create targeted wordlists for defensive security testing. The tool accepts personal inputs (names, dates, pets, keywords), applies common transformations (case variants, leetspeak, separators, year suffixes) and exports deduplicated .txt wordlists compatible with password auditing tools. It provides both a Command Line Interface (CLI) and a simple Tkinter GUI so it can be run in Python IDLE or from the terminal.

Tools Used

- **Python 3.x** primary programming language and runtime.
- **Tkinter** built-in GUI toolkit for the application front end (works in IDLE).
- **zxcvbn (optional)** third-party password strength estimator (pip install zxcvbn) used when present for more accurate analysis.
- **Standard libraries** argparse, itertools, re, datetime, pathlib, and json for core functionality and file handling.

Steps Involved in Building the Project

1. **Requirement & Design**
   - Defined objectives: analyze password strength, generate targeted wordlists, export to .txt, and offer GUI + CLI.
   - Identified ethical considerations and added explicit warnings in the UI and CLI help text.
2. **Implementing the Password Analyzer**
   - Implemented a custom entropy heuristic: compute bits of entropy from character classes and length, then apply pattern-based deductions (repeats, common words, simple sequences).

- Added optional integration with zxcvbn to provide a widely-accepted, richer analysis when available.
- Designed output to include both entropy and a 0–4 score to aid quick interpretation.

3. **Building the Wordlist Generator**

   - Collected input parsing for names, dates, pets, keywords, and extras via CLI arguments and GUI fields.
   - Implemented transforms: case variants (lower/title/upper/camel), limited leetspeak substitutions, combining words pairwise with separators, appending years, and adding common prefixes/suffixes.
   - Added safeguards to limit combinatorial explosion (max lines, capped leet variants, length checks) and deduplicated results.

4. **User Interfaces**

   - CLI: implemented analyze and generate subcommands with clear arguments and help text.
   - GUI: simple Tkinter layout with two panels (analyzer + generator), file output selection, checkboxes for options, and completion messages suitable for use in IDLE.

5. **Export & Testing**

   - Implemented robust text-file export with UTF-8 encoding and ensured compatibility with common cracking tools by using one candidate per line.
   - Performed manual tests to verify transforms, file output, and analyzer behavior with and without zxcvbn.

Conclusion

The completed tool provides a practical, ethical utility for password auditing and targeted wordlist generation. Its single-file implementation and dual interfaces make it accessible to learners and practitioners who want to test passwords in controlled environments or to build defensive assessments for their systems. Future enhancements could include more transformations (keyboard walks, substring extraction), progress indicators for very large lists, and integration with enterprise password policy checks. The tool should always be used responsibly, only against systems where the user has explicit authorization.