**Title:** Password Strength Analyzer & Custom Wordlist Generator
**Intern:** Swapnil Dixit
**Course:** Elevate Labs Cyber Security Internship
**Date:** 25-10-2025

**Project:** Password Strength Analyzer with Custom Wordlist Generator

---

## Introduction

Password security remains a critical component of digital safety. Many accounts are compromised because users choose weak, guessable, or personally related passwords. This project implements a compact, reproducible tool that evaluates password strength using an industry-standard estimator and generates focused wordlists from user-supplied artifacts to demonstrate how attackers might target weak passwords.

## Abstract

This deliverable is a CLI-based Password Strength Analyzer that leverages the **zxcvbn** estimator to compute a password's score, entropy, and estimated crack time. A companion custom wordlist generator produces candidate passwords by combining user-provided tokens (e.g., names, pet names, years) with common transformations (capitalization, numeric suffixes, basic leetspeak). The tool is intended for defensive use, to evaluate password robustness and illustrate risks from predictable personal-pattern wordlists.

## Tools Used

- **Python 3.8+** — Implementation language.
- **zxcvbn-python** — Password strength estimator (score, entropy, crack times).
- **argparse** — Command-line interface.

## Steps Involved in Building the Project

1. **Environment setup** — Install Python and required packages:
   - PIP install zxcvbn-python
2. **Analyzer wrapper** — Created a small wrapper around zxcvbn to extract the following fields: score (0–4), entropy (bits), human-readable crack-time estimate, warnings, and suggestions.
3. **Robust output handling** — Defensive checks added so missing fields (e.g., None entropy) do not crash the program; values are normalized for reporting.

4. **Custom wordlist generator** — Implemented a generator that accepts tokens (--name, --pet, --years, --extra) and emits variants:
    - Capitalization variants (LOWER, UPPER, Capitalized)
    - Numeric suffixes/prefixes (years, 0–99 tails)
    - Basic leetspeak transformations (a->4, o->0, e->3)
    - Output exported to text file (.txt) limited to a practical size for submission
5. **Command-line interface** — Two primary modes: --check <password> (analyze) and --gen (generate wordlist). Example usage:
    - python pwd_analyzer.py --check "P@ssw0rd123"
    - python pwd_analyzer.py --gen --name Swapnil --pet Fluffy --years 1990,1998 --out Swapnil_wordlist.txt
6. **Testing & evidence capture** — executed sample runs and ensured clear terminal output for inclusion in the final submission.

**Conclusion**

The delivered prototype is a simple, effective proof-of-concept that demonstrates both how modern password strength estimators score passwords and how easily targeted wordlists can be produced from personal artifacts. It is suitable for classroom demonstrations, quick audits, and as a foundation for extensions (batch analysis, integration with breach datasets). The code is lightweight, dependency-minimal, and produces clear, reproducible output that supports the required 1–2 page report.

---

**Deliverables**

- pwd_analyzer.py (source code)
- Pwd_analyzer_ss(attached screenshots or terminal logs)
- swapnil_wordlist.txt (example wordlist)
- Swapnil_Password_Strength_Analyzer_Report.PDF (Report)

**Contact**

Swapnil Dixit

Swapnil291198@gmail.com