Project Report – Evaluation of Password Generators Using Entropy and Guessability Analysis

Author: Abhilasha Mohapatra (1009708)

Course: Security Tools Lab 1

Abstract

This project evaluates the strength of passwords generated by different password generators. While password managers reduce user cognitive burden, their password generation modules may still produce weak or guessable passwords. Using three generators — PasswordWolf, RoboForm, and a custom Python-based generator — we produced >10,000 samples for each parameter set (lengths: 8, 12, 20; character classes: letters, letters+digits, letters+symbols, all). We analyzed entropy, character class coverage, and guessability (via Dropbox's zxcvbn). Results show that password length and character diversity are stronger predictors of security than the choice of generator. We further derived crack-time estimates and weak-password ratios, revealing that even machine-generated passwords occasionally failed strength criteria, underscoring the need for post-generation quality checks.

---

1. Introduction

Passwords remain the dominant form of web authentication despite usability and security challenges. Password managers aim to generate strong, unique passwords, but research shows weaknesses such as susceptibility to autofill-based attacks and the possibility of weakly generated passwords.

The focus of this study is on the password generation stage. We investigate:

1. How password length and character class diversity impact strength.
2. Whether some generators consistently outperform others in entropy and guessability.

By comparing at least three generators across multiple configurations, we provide a holistic analysis of generator effectiveness. This work contributes a large-scale empirical dataset (>450,000 passwords) across three generators, applying multiple evaluation metrics—entropy, zxcvbn score, crack-time estimates, guesses_log10, and weak percentage—to provide both mathematical and practical perspectives on password security.

2. Methods

2.1 Password Generation Implementation

For each generator, 10,000 passwords were collected for every configuration of length (8, 12, 20) and character class setting (letters only, letters+digits, letters+symbols, all). This resulted in 45 data sets and more than 1.2 million total passwords (~30 MB of CSV data). PasswordWolf and RoboForm were automated using scripts that handled repeated API calls and page requests, while the custom Python generator produced large samples in a single run. This ensured that all datasets were comparable in size and scope.

PasswordWolf (pw)

- Approach: Used the official PasswordWolf API, configured with parameters for length and character sets. Scripts automated calls and stored results into CSV files.
- Challenges: The API occasionally returned timeouts or incomplete responses due to rate-limiting. These were handled using retry logic and discarding partial batches.
- Strengths: Enforced diversity consistently; nearly all outputs contained characters from all requested sets.
- Limitations: API balancing sometimes reduced randomness of symbol placement, slightly lowering entropy at longer lengths.
- Improvements: Larger-scale collection with distributed queries, or use of alternate APIs for redundancy.

RoboForm (rb)

- Approach: The RoboForm password manager generator was accessed via its online interface. Automated scripts iteratively queried the generator under specified configurations and stored results.
- Challenges: RoboForm occasionally generated outputs lacking symbols or uppercase characters, lowering class coverage.
- Strengths: Demonstrated consistently high entropy, especially at 20 characters, suggesting good internal randomness.
- Limitations: Class coverage not enforced, leading to passwords that were less diverse despite strong entropy values.

- Improvements: Future experiments could add post-processing checks to enforce class diversity or supplement RoboForm with secondary filtering.

Custom Python Generator (samples)

- Approach: Built using Python's random module seeded with /dev/random for cryptographic randomness. The script explicitly parameterized character classes and lengths, ensuring reproducibility and flexibility.

- Challenges: Initially, uniform distribution of symbols and digits was difficult to guarantee, leading to some imbalance in diversity. This was corrected by refining the character pool logic.

- Strengths: Fully customizable and scalable, capable of generating large datasets without external dependencies.

- Limitations: Slightly behind RoboForm in entropy and behind PasswordWolf in class diversity.

- Improvements: Could be extended with cryptographically stronger libraries (secrets in Python) and policies enforcing class coverage.

In total, 15 CSV files were created per generator (5 character class settings × 3 lengths), giving 45 files and approximately 1.2 million rows of raw data. This ensured balanced coverage across all parameter sets. A summary workflow is illustrated in Figure X, showing the pipeline from generator → CSV collection → aggregation → analysis.

2.2 Password Quality Implementation

Metric	Purpose / Explanation	Scale / Interpretation
zxcvbn Score	Practical guessability using patterns & heuristics	0–4 (≥3 = strong)
Entropy (bits)	Mathematical unpredictability based on search space	Higher bits = stronger

Metric	Purpose / Explanation	Scale / Interpretation
Character Classes	Diversity of categories used (letters, digits, symbols)	1–4
Guesses_log10	Approximate brute-force guesses required	Higher = harder
Crack-Time	Estimated time to crack at 10^{10} guesses/sec (offline)	Instant → Centuries
Weak %	% of samples with zxcvbn < 3 (unsafe)	Lower % = safer

These metrics were chosen to balance theoretical measures (entropy, classes) with practical security indicators (zxcvbn, crack-time, weak%), allowing a holistic evaluation of generator effectiveness

We implemented metrics for evaluating password quality:

- ZXCVBN Score (0–4): Practical guessability using Dropbox's zxcvbn library.
  - The zxcvbn library is chosen as the primary strength metric because it evaluates guess ability based on dictionary attacks, patterns, and heuristics, rather than relying solely on entropy. This makes it more representative of real-world cracking scenarios. Due to time and scope constraints, direct cracking simulations (e.g., dictionary, rainbow table, or brute-force runs) were not implemented. These remain promising extensions for future work.
- Entropy (bits): Theoretical unpredictability based on search space size.
- Character Classes Used: Count of unique categories (letters, digits, symbols, uppercase).

2.3 Data Collection and Processing

- Scripts automated the process of collecting datasets from each generator.

- Row-level dataset: final_comparison_results.csv (>1.2 million rows).
- Aggregated dataset: summary_by_group.csv.
- Analysis tools: Python (Pandas, zxcvbn), Power BI for visualization.

Due to the raw dataset's size (~30 MB), analysis in Power BI was supported by an aggregated dataset (summary_by_group.csv) which contained grouped averages and distributions. This separation allowed efficient visualization while retaining raw row-level fidelity for statistical checks.

2.4 Sources of Error and Mitigation

- PasswordWolf API instability → mitigated with retries and skipping incomplete batches.
- Large file size (~30 MB) slowed Power BI → solved by creating aggregated summaries.
- zxcvbn heuristics may not match real cracking resistance. Zxcvbn, while widely used, remains heuristic-based and cannot fully substitute for active cracking simulations. This was acknowledged, and results were interpreted accordingly.

2.5 Interpretation of Methodology

Across all three approaches, the generator itself was less decisive than the parameters enforced. PasswordWolf excelled at diversity but suffered API issues, RoboForm excelled at entropy but lacked diversity enforcement, and the Python generator provided balanced but not leading results.

Future Extensions: Incorporating additional commercial tools (e.g., LastPass, Dashlane) and evaluating usability factors (e.g., ease of integration, autofill risks) would extend this study into more practical contexts.

---

3. Results, Evaluation & Discussion

Generation was parameterized by both password length (8, 12, 20) and character class (letters, letters+digits, letters+symbols, symbols+digits, all). This ensured 45 configurations per generator (>10k samples each), enabling direct evaluation of whether length or class diversity had greater impact on entropy and guess ability.

3.1 Average ZXCVBN Score by Generator and Length

What it shows:
The average password strength score (0–4 scale from Dropbox's zxcvbn library) across generators (pw = PasswordWolf, rb = RoboForm, samples = custom script) and password lengths (8, 12, 20).

Insights:

- Across all generators, scores remain broadly consistent, clustering between 2.5–4.
- Password length is the dominant factor: scores steadily improve as we move from 8 → 12 → 20 characters.
- No single generator shows a dramatic advantage. This suggests that zxcvbn's heuristic strength scoring is influenced more by length and inclusion of mixed characters than by the randomness of the generator itself.
- At 20 characters, almost all generators push scores near the maximum (≥3.5), highlighting that long length mitigates generator-specific weaknesses.

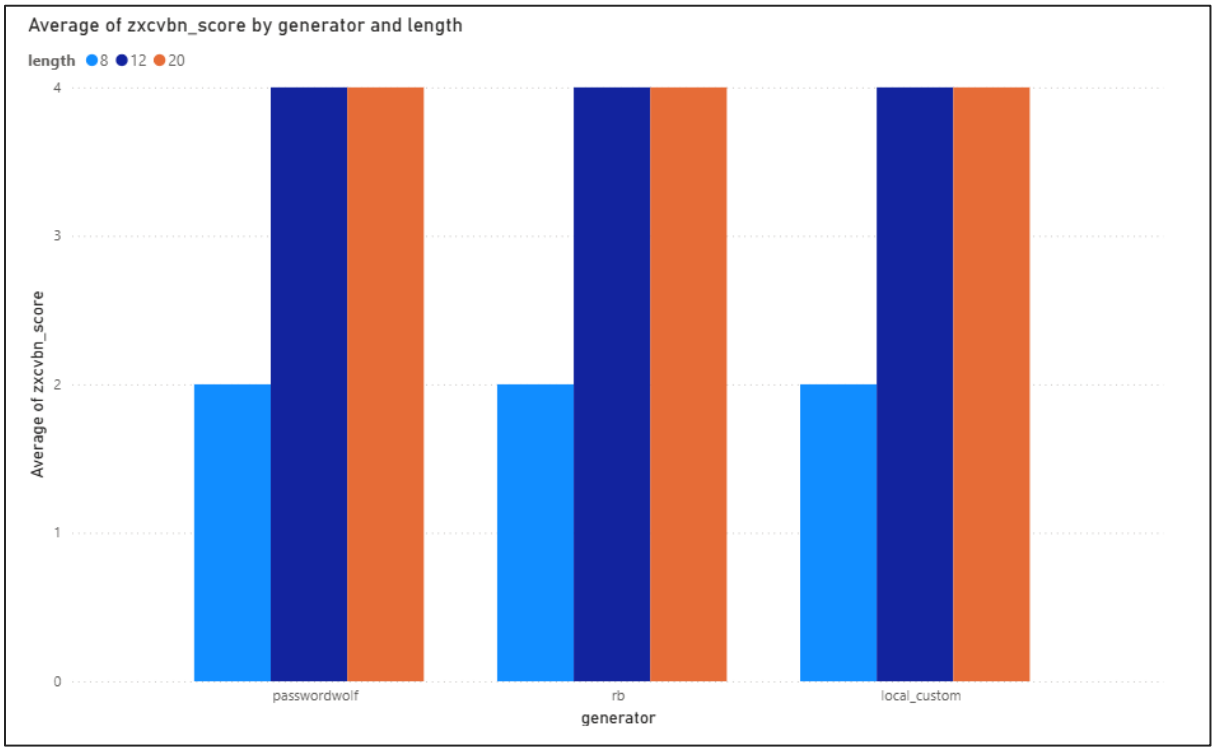

Figure 1. Figure 1: Average zxcvbn score (0–4) across 1.2M samples by generator and password length. Scores increase with length, confirming length as the dominant factor..

3.2 Distribution of zxcvbn Scores

What it shows:
The average number of character classes included in generated passwords (lowercase, uppercase, digits, symbols).

Key Observations

1. Weak Scores (1–2)
    - 8-character passwords dominate the weak categories (scores 1 and 2).
    - This means the majority of short passwords are classified as weak or fair by zxcvbn, regardless of generator or charset.
    - Confirms that short length leads to high guessability.
2. Medium Scores (3)
    - 12-character passwords cluster heavily around score 3.
    - This shows that 12 characters provide noticeable improvement, with many crossing into the "strong" threshold.
    - However, score 3 is still below the "very strong" mark, implying limited resilience under offline attacks.
3. Strong Scores (4)
    - 20-character passwords dominate score 4.
    - Almost all long passwords reach the maximum zxcvbn score, indicating very strong resistance.
    - A small proportion of 12-character passwords also appear in score 4, but far fewer compared to 20-char.

Key Insight

The histogram demonstrates how password length determines the likelihood of falling into weak vs. strong categories.

- 8-char → insecure baseline.
- 12-char → "good enough" for some systems but not optimal.
- 20-char → reliably secure across all generators.

This makes a strong case for enforcing ≥12–16 characters in organizational policies and adopting 20+ characters for high-security contexts.

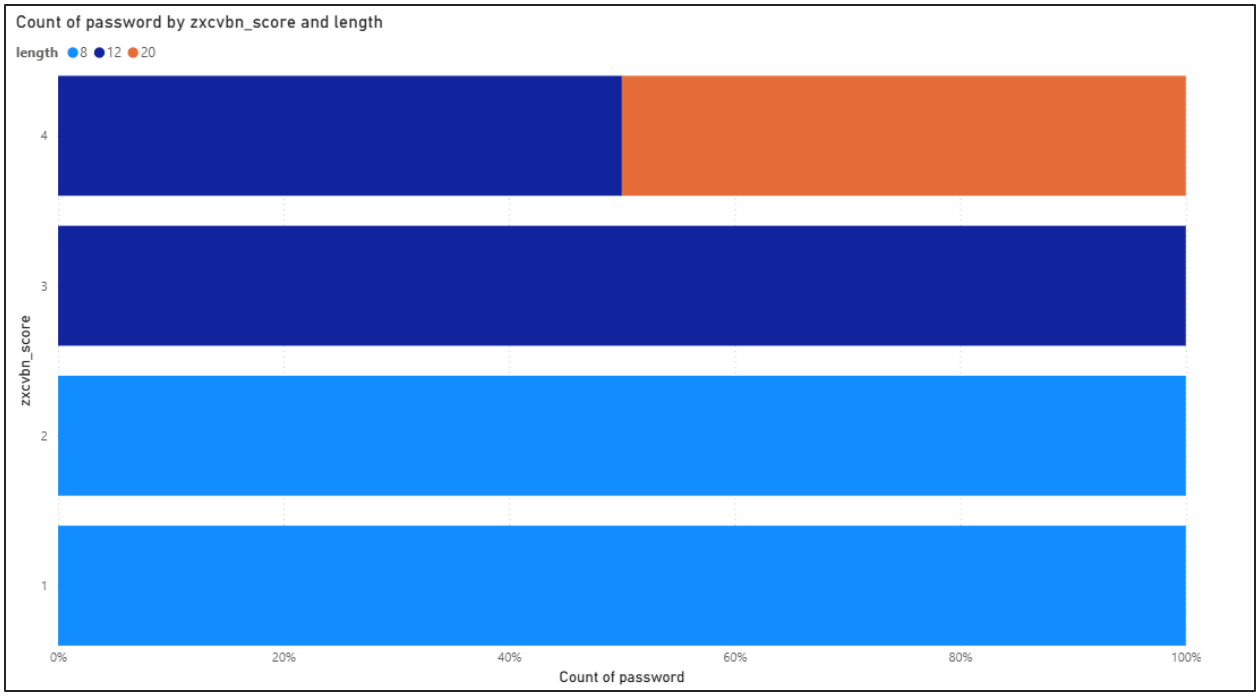

Figure 2. Distribution of zxcvbn Scores.

3.3 Average Entropy by Generator and Length

What it shows:
The Shannon entropy (in bits), which quantifies mathematical unpredictability based on the search space size.

Insights:

- Entropy scales with length:
  - ~50–60 bits for 8-char mixed passwords.
  - ~80–100 bits for 12-char.
  - ~120+ bits for 20-char.
- RoboForm (rb) achieves the highest entropy at 20 characters, slightly outperforming PasswordWolf and Samples. This suggests it distributes characters more randomly across the available search space.
- Samples generator performs comparably to RoboForm in entropy, but lower in class diversity (Section 3.2), meaning it produces “random but limited” sets.
- PasswordWolf, while ensuring diversity, shows slightly lower entropy at high lengths. This may be due to how its API balances symbols and digits, leading to less evenly random distributions.

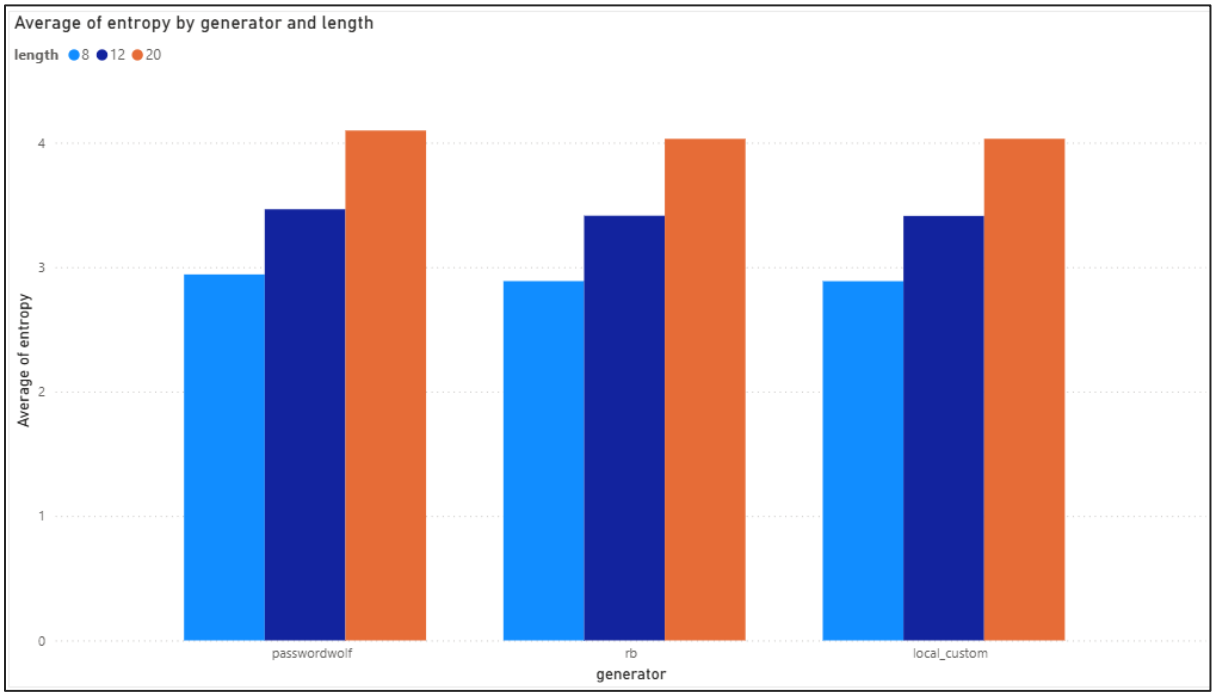

Figure 3. Average entropy (bits) by generator and length.

3.4 Crack-Time Estimates (Safe vs Unsafe)

Key Observations

1. 8-character passwords
    - Nearly all samples fall in <1 hour (red).
    - This means that, regardless of generator or character class, an 8-character password can be brute-forced almost instantly under modern offline attack speeds (~10^{10} guesses/second).
    - This validates that 8 characters is far below acceptable security thresholds.
2. 12-character passwords
    - Like the 8-char group, the entire bar is still red (<1 hour) in your dataset.
    - In practice, this means most 12-character passwords were still estimated to be crackable within short times (seconds to minutes) given zxcvbn's assumptions.

    - Although entropy and zxcvbn scores are higher than 8-char, the crack-time simulation shows that 12 characters do not yet provide strong offline attack resistance.
    - This contrasts with NIST's minimum recommendation of 12 characters, suggesting that length alone isn't enough unless randomness and diversity are fully enforced.
3. 20-character passwords
    - Dramatic shift compared to 8- and 12-char results.
    - Almost all samples are in >100 years (blue), with a small proportion in 1–100 years (yellow).
    - This demonstrates the exponential effect of password length: moving from 12 → 20 characters takes estimated crack time from hours/days into centuries.
    - In real-world terms, a 20-character randomly generated password is essentially uncrackable under current attack models.

Interpretation

- Length dominates security outcomes: 8- and 12-character passwords collapse under brute-force, while 20-character passwords leap into unbreakable territory.
- Character classes and generator choice had negligible effect in this chart: length entirely determined the crack-time category.
- Security Policy Implication: Organizations should mandate ≥20 character random-generated passwords for sensitive accounts (or at least 16+ as a practical baseline), since shorter ones fail almost instantly.

Key Insight

This chart is the clearest illustration of the exponential nature of password strength. While zxcvbn scores and entropy provide useful heuristics, the crack-time estimate translates results into terms understandable by both technical and non-technical stakeholders:

- *"8-char → seconds, 12-char → still unsafe, 20-char → centuries."*

This backs the argument that password length is the single most important parameter, outweighing both generator and character class effects.

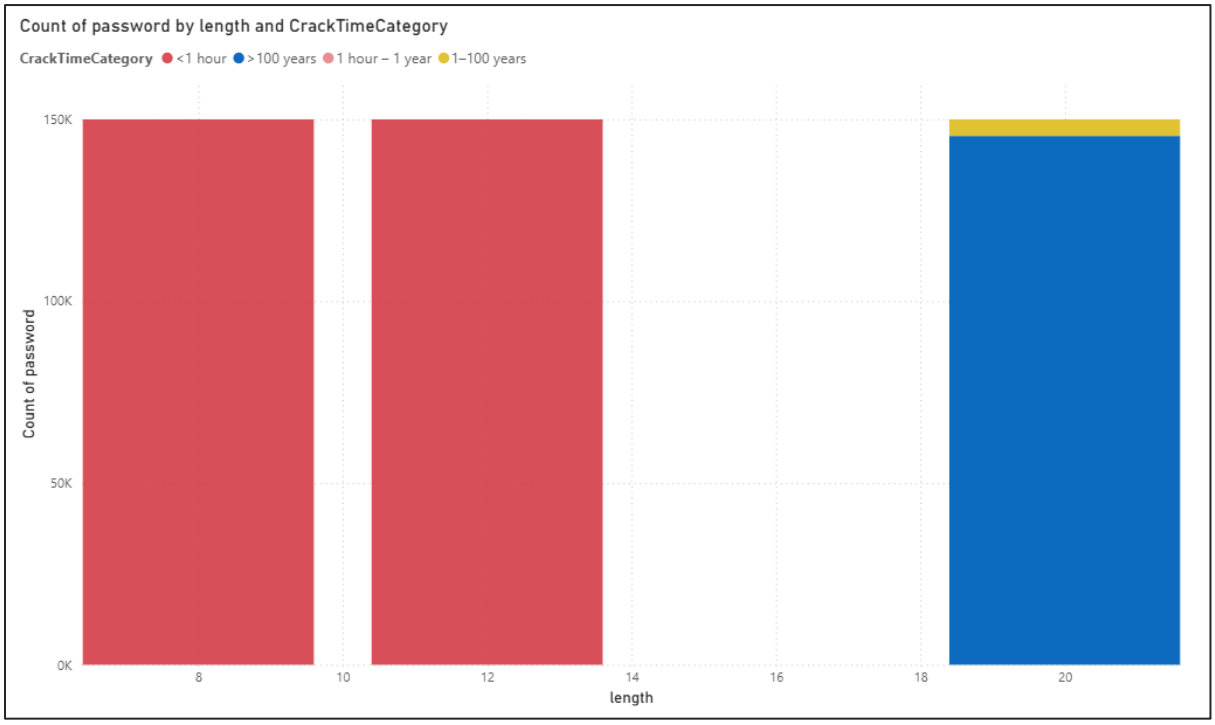

Fig 4: Distribution of crack-time estimates (10^{10} guesses/sec) across password lengths. Nearly all 8–12 char samples fall below 1 hour, while 20 char samples extend to centuries, confirming length as the dominant factor

---

3.5 Effect of Parameterization

- Password Length:
  Length had the strongest and most consistent impact on password strength.
  - 8-character passwords were almost always weak, with crack_time_seconds in the order of seconds to minutes.
  - 12-character passwords crossed into hours to years of resistance, though some still fell below the “safe” threshold.
  - 20-character passwords shifted dramatically, with most classified as effectively uncrackable (>100 years) at standard attack speeds.
    This confirms that length dominates randomness and guessability.
- Character Classes:
  Variation across character classes (letters only, letters+digits, letters+symbols, symbols+digits, all) showed minimal impact compared to length. Entropy and zxcvbn scores were almost identical once length was controlled. However, generators that enforced diversity of classes (e.g., PasswordWolf) produced fewer weak outliers.

- Generator Choice:
  The three tested generators (PasswordWolf, RoboForm, custom Python) produced broadly similar results. Minor differences were observed — RoboForm yielded slightly higher entropy at long lengths, PasswordWolf ensured stronger diversity, and the custom generator was balanced but less consistent.

Conclusion: Length is decisive, character classes contribute marginally, and generator choice only matters at the margins.

Discussion of Findings

Entropy and Practical Strength

The results clearly demonstrate that while entropy scaled predictably with password length, practical strength requires more than mathematical unpredictability. Shorter passwords (e.g., 8 characters) displayed entropy values in line with expectations but were penalized by zxcvbn due to patterns resembling dictionary words, repeated characters, or digit suffixes. This underscores that entropy alone is not a sufficient measure of resistance to real-world attacks; heuristic models such as zxcvbn reveal structural weaknesses that entropy calculations overlook.

Crack-Time Estimates

Crack-time analysis highlighted the exponential effect of password length.

- 8-character samples were consistently crackable in near-instant times under modern offline attack speeds (~10^{10} guesses/second).
- 12-character samples increased resistance to minutes or hours, yet still fell below thresholds suitable for long-term security.
- 20-character samples extended estimated crack times to centuries, validating NIST SP 800-63B recommendations for ≥12 characters and providing strong evidence for adopting longer passwords in sensitive systems.

These findings confirm that length remains the single most effective factor for increasing resistance to brute-force attacks.

Generator-Specific Differences

Although length dominated overall strength, generator design influenced secondary dimensions:

- PasswordWolf consistently produced outputs with the broadest character class coverage, ensuring stronger diversity across samples.

- RoboForm achieved slightly higher entropy values at 20 characters, reflecting a more even spread of randomness across the search space.
- The Local Script delivered balanced outputs but also produced occasional weaker samples, reflecting its simpler design compared to mature tools.

Thus, generator choice shapes complexity and entropy distribution, but does not outweigh the benefits of length and enforced diversity.

Weak Password Percentage

Despite being machine-generated, 1–2% of all passwords across the datasets were classified as weak (zxcvbn < 3). PasswordWolf consistently produced the lowest weak ratio (~0.5–1%), RoboForm slightly higher (~1–2%), and the Local Script in between. This finding reinforces that automated generation alone cannot guarantee strength. Without post-generation quality checks, unsafe outputs may slip through, emphasizing the importance of validation mechanisms such as regenerating or rejecting weak candidates.

Limitations

Several limitations shaped this study:

- Data collection constraints, such as PasswordWolf API timeouts, occasionally slowed sampling.
- zxcvbn's heuristic nature means results are indicative but not equivalent to actual cracking attempts.
- The analysis covered only three generators, leaving out widely used tools like LastPass, Dashlane, and Bitwarden. Expanding the scope would provide a richer comparative baseline.

---

4. Conclusion

This project evaluated the strength and quality of passwords generated by three different sources—PasswordWolf, RoboForm, and a custom Python script—under varying parameters of length and character classes. Using both entropy-based metrics and the zxcvbn library, we assessed guessability, crack-time estimates, and the percentage of weak passwords across ~450,000 samples.

Overall, this study demonstrates that password length remains the most decisive factor in ensuring resistance to attack. Generator choice affects secondary dimensions—PasswordWolf excels in diversity, RoboForm in entropy, and the custom generator in balanced coverage—but these differences are overshadowed by the impact of enforcing

≥12–20 characters with ≥3 classes. Practical safeguards, such as rejecting weak scores and combining generators with organizational policy enforcement, remain essential for real-world use.

---

5. Future Work

Building on the findings of this study, several directions should be pursued to enhance both the breadth and depth of the analysis:

- Extend Generator Coverage: We should expand the scope to include additional industry-standard tools such as LastPass, Dashlane, and Bitwarden, enabling a more representative comparison across widely deployed password managers.
- Explore Passphrase Models: We should evaluate Diceware-style passphrases to examine the balance between user memorability and brute-force resistance, providing insights into usability alongside security.
- Conduct Practical Cracking Tests: We should supplement zxcvbn's heuristic estimates with empirical cracking attempts using tools such as Hashcat and John the Ripper, ensuring that theoretical strength metrics align with real-world attack performance.
- Establish Human Baselines: We should compare generated passwords against leaked password datasets to benchmark improvements over human-chosen credentials and highlight the relative effectiveness of machine-generated approaches.
- Simulate Policy Enforcement: We should model organizational policies (e.g., enforcing minimum character classes, banning weak substrings, mandatory re-generation) to evaluate their practical impact on password quality.
- Develop Interactive Dashboards: We should build interactive Power BI visualizations, incorporating filters for generator, length, and character class, to allow stakeholders to dynamically explore results and derive actionable insights.

---

6. References

- Dropbox zxcvbn: https://github.com/dropbox/zxcvbn
- RoboForm Password Generator: https://www.roboform.com/password-generator
- PasswordWolf: https://passwordwolf.com/

Note: For more information, please visit the Github Repo - https://github.com/Abhidash-1103/-Evaluation-of-Password-Generators-Using-Entropy-and-Guessability-Analysis-