Vulnerability Analysis on Open-Source Software Repositories

Bhoumik Patidar

March 25, 2025

1 Introduction, Setup, and Tools

1.1 Overview and Objectives

This report presents a vulnerability analysis on open-source Python repositories using Bandit, a static code analysis tool. The main objectives of this study are to:

- Understand and configure Bandit for analyzing Python code.
- Assess the security vulnerabilities in three real-world repositories: *Deeplake*, *Flower*, and *Cookiecutter*.
- Investigate trends in vulnerability confidence, severity, and CWE (Common Weakness Enumeration) coverage over the development timeline.

1.2 Environment Setup

The analysis was conducted in the following environment:

- **Operating System:** macOS
- **Python Version:** [e.g., Python 3.12]
- **Virtual Environment:** A virtual environment was created using `conda`.
- **Bandit**
- **Other Python Packages:** `pandas`, `matplotlib`, and `seaborn` were used for data processing and visualization.

1.3 Tools and Their Purpose

- **Bandit:** A security linter for Python code that identifies common vulnerabilities by performing static analysis.
- **Git:** Used for version control and for extracting commit metadata.
- **Python Libraries:**
  - `pandas` was used for data aggregation and manipulation.
  - `matplotlib` and `seaborn` were used for creating visualizations.

2 Repository Selection

The repositories selected for this analysis were chosen based on their significant impact in the open-source community. These projects not only have a high number of stars, indicating community endorsement, but also feature an extensive commit history that reflects continuous development and active usage. The selected repositories are:

- **Flower** (https://github.com/adap/flower)
  With approximately 5.6k stars and 3.5k commits, Flower is widely recognized as a robust tool for distributed training, demonstrating both popularity and ongoing maintenance.
- **Deeplake** (https://github.com/activeloopai/deeplake)
  Boasting around 8.5k stars and 9k commits, Deeplake is a highly active repository that underscores its importance in data management and machine learning applications.
- **Cookiecutter** (https://github.com/cookiecutter/cookiecutter)
  With an impressive 23.3k stars and 3k commits, Cookiecutter is one of the most popular project scaffolding tools in the Python ecosystem, widely adopted for creating consistent project templates.

The combination of high star counts and extensive commit histories makes these repositories ideal candidates for a detailed vulnerability analysis using Bandit, as they represent mature and actively maintained projects with substantial real-world usage.

3 Methodology and Execution

3.1 Step-by-Step Procedure

The analysis was carried out in the following steps:

1. **Environment Setup:** A virtual environment was created and necessary tools were installed

2. **Generating Bandit Reports:** For each repository, Bandit was run over the last 100 commits (excluding merge commits) to generate CSV reports. An example Bash script used was:

```
#!/bin/bash
# Create the output directory for Bandit reports if it doesn't exist
mkdir -p bandit_reports

# Loop through the last 100 commits (ignoring merge commits)
for commit_hash in $(git log --no-merges -n 100 --pretty=format:"%H"); do
    # Checkout the commit in quiet mode
    git checkout "$commit_hash" --quiet
    echo "Processing commit: $commit_hash"

    # Define the file path for the Bandit report of the current commit
    report_path="bandit_reports/report_${commit_hash}.csv"

    # Run Bandit recursively on the repository, outputting results in CSV format
    bandit -r . -f csv -o "$report_path"
done
```

3. **Data Organization:** The generated CSV files were organized into separate folders for each repository:

   - `deeplake_results/`
   - `flower_results/`
   - `cookiecutter_results/`

   Additionally, output plots and aggregated CSV files were stored in the `bandit_plots/` folder.

4. **Data Aggregation and Visualization:** Custom Python scripts were used to process the CSV files. Key steps included:

   - Aggregating per-commit counts for HIGH, MEDIUM, and LOW confidence and severity issues.
   - Calculating unique CWE counts per commit.
   - Generating repository-level plots and overall comparative plots.

5. **Error Handling:** During execution, errors such as missing CSV files or issues reading files were logged and handled to ensure smooth execution.

4 Result And Analysis

This section details the per-commit analysis of Bandit's output for each of the three repositories: **deeplake**, **flower**, and **cookiecutter**. For each repository, I report the following metrics:

- The number of **HIGH**, **MEDIUM**, and **LOW** confidence issues per commit.
- The number of **HIGH**, **MEDIUM**, and **LOW** severity issues per commit.
- The unique Common Weakness Enumeration (CWE) identifiers reported per commit.

4.1 Deeplake Repository

Figure 1 shows the trend of confidence-level issues across commits for the Deeplake repository. Similarly, Figure 2 presents the severity-level trends. In addition, Figure 3 shows the trend of unique CWE counts per commit.

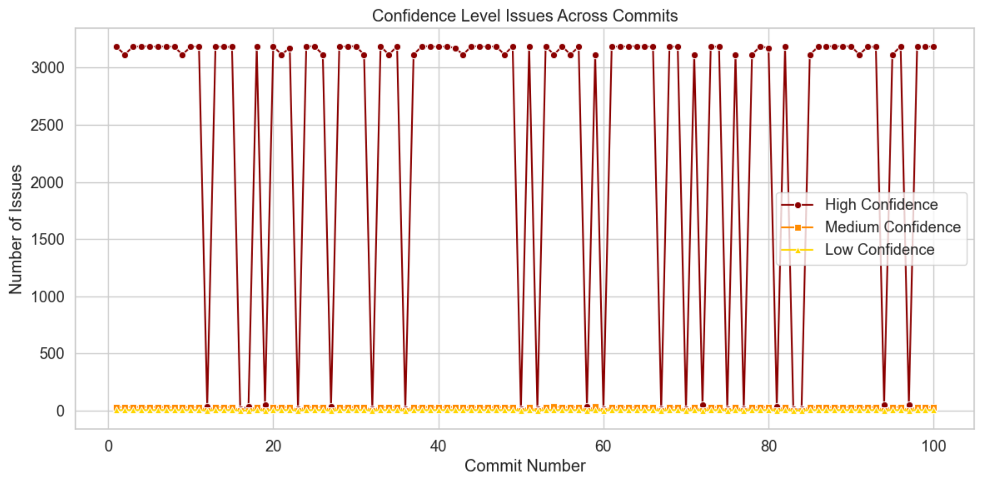

Figure 1: Trend of HIGH, MEDIUM, and LOW confidence issues per commit for the Deeplake repository.

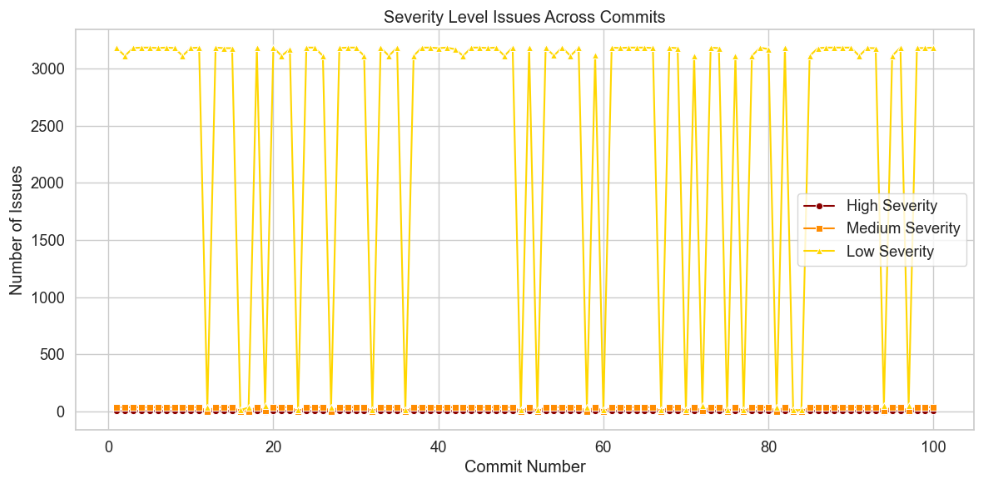

Figure 2: Trend of HIGH, MEDIUM, and LOW severity issues per commit for the Deeplake repository.

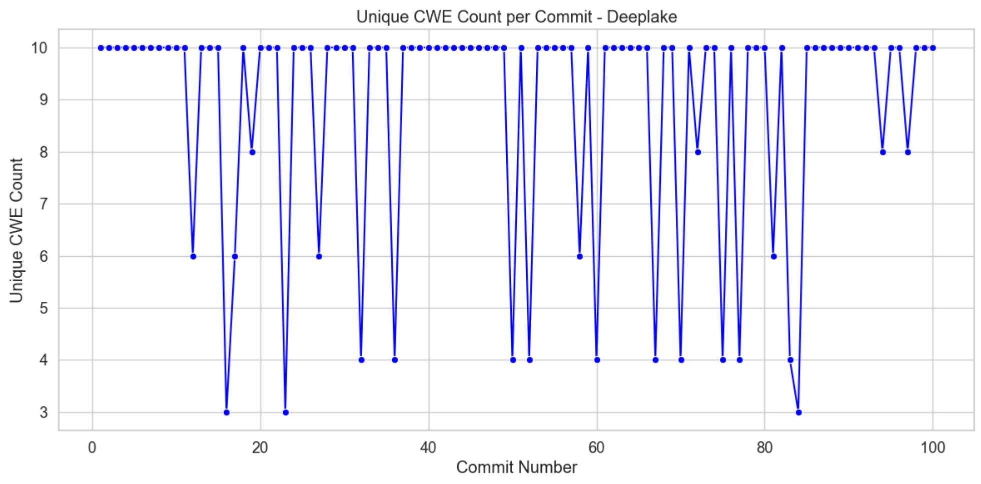

Figure 3: Trend of unique CWE count per commit for the Deeplake repository.

4.2 Flower Repository

Figures 4, 5, and 6 present the corresponding analyses for the Flower repository.

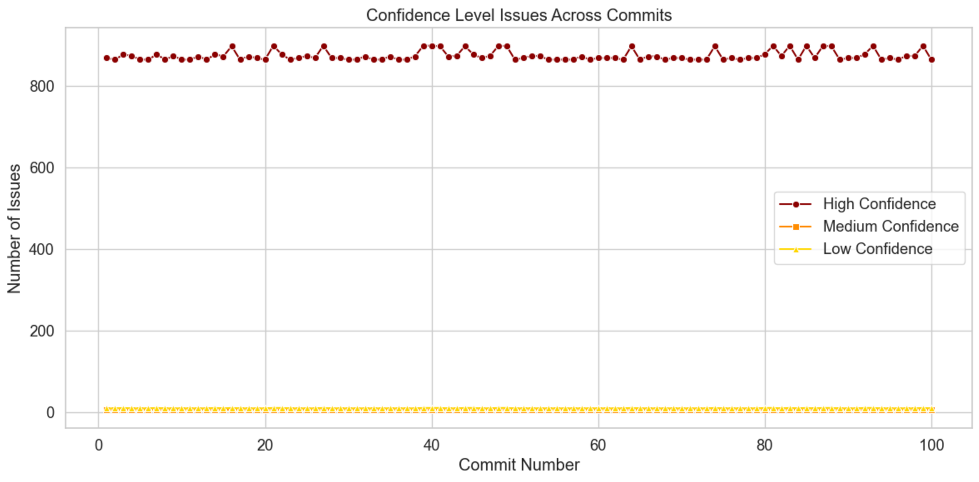

Figure 4: Trend of confidence-level issues per commit for the Flower repository.

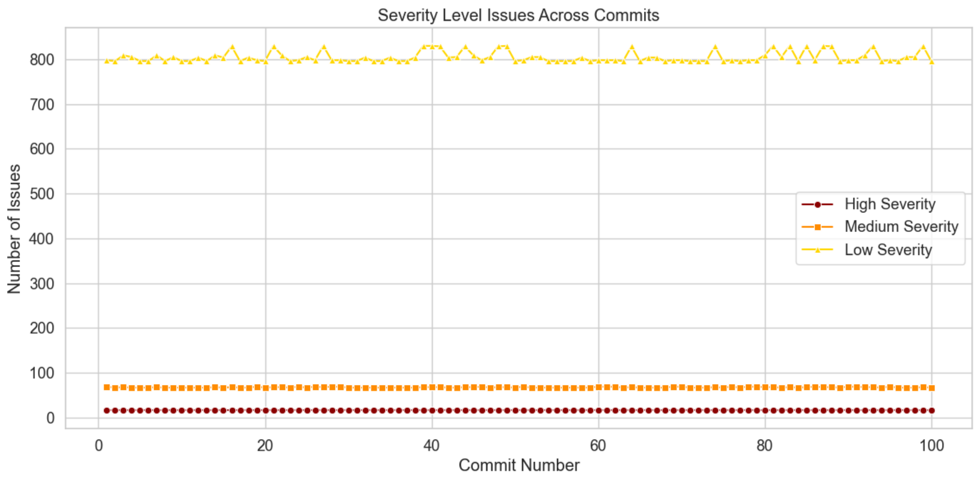

Figure 5: Trend of severity-level issues per commit for the Flower repository.

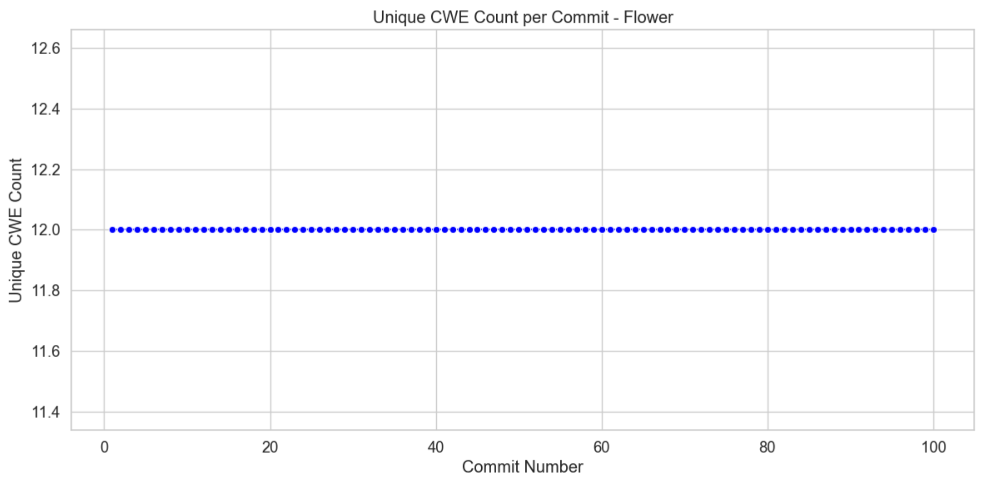

Figure 6: Trend of unique CWE count per commit for the Flower repository.

4.3 Cookiecutter Repository

Figures 7, 8, and 9 present the corresponding analyses for the Cookiecutter repository.

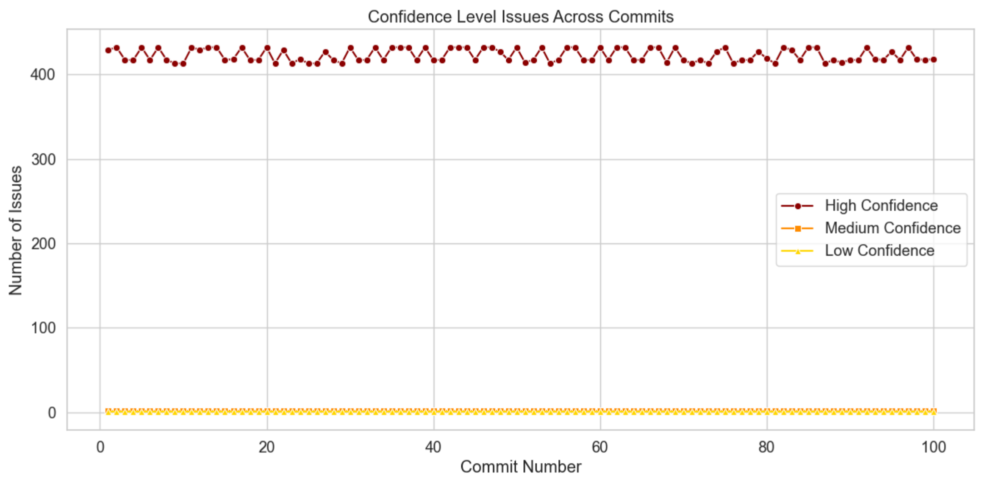

Figure 7: Trend of confidence-level issues per commit for the Cookiecutter repository.

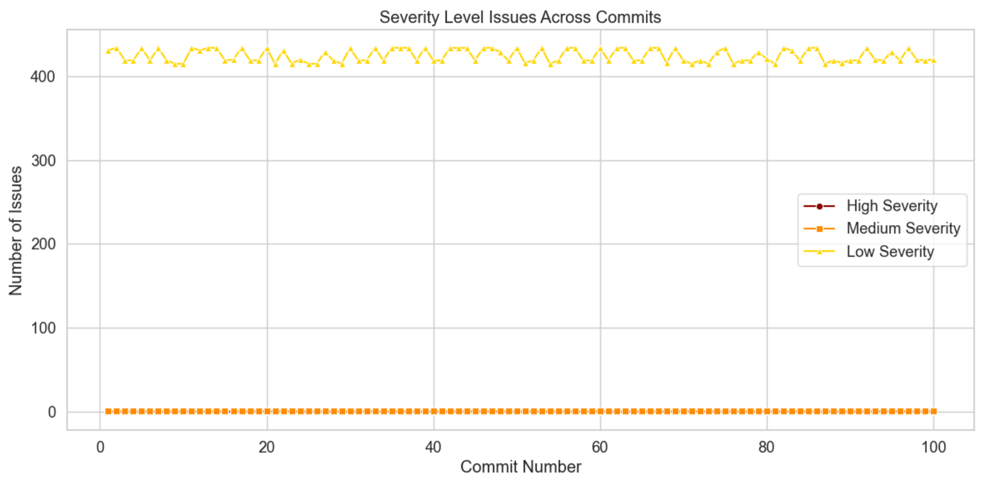

Figure 8: Trend of severity-level issues per commit for the Cookiecutter repository.

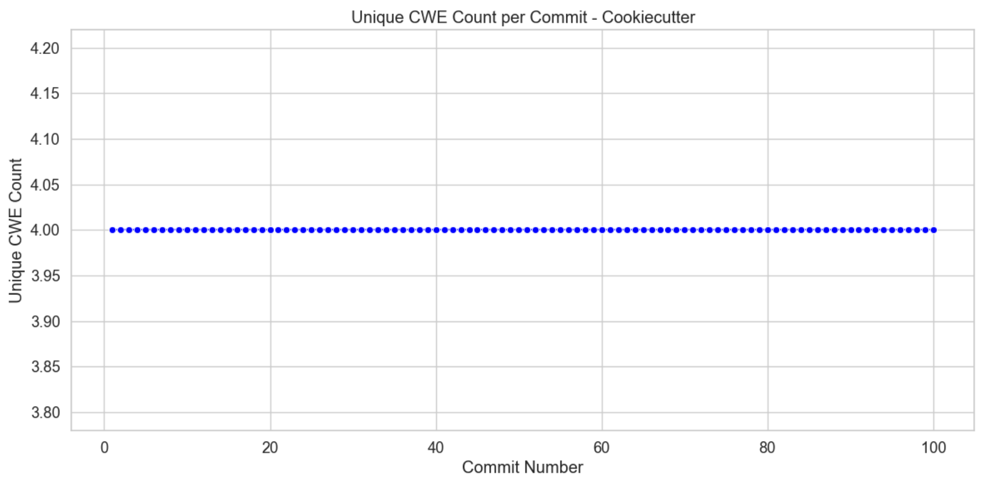

Figure 9: Trend of unique CWE count per commit for the Cookiecutter repository.

5 Overall Dataset-Level Analyses

This section addresses the research questions (RQs) posed by the assignment. The analyses are based on the aggregated data from all repositories.

RQ1: High Severity Vulnerabilities

Purpose: This research question investigates when vulnerabilities with high severity are introduced and subsequently fixed in the open-source repositories. It aims to understand the timeline of critical issues and their remediation.

Approach: I analyzed the per-commit data for high severity issues, using the CSV data and the corresponding trend plots for each repository. The analysis involved identifying peaks (indicating introduction of vulnerabilities) and dips (indicating remediation).

Results:

- **Deeplake:** The trend in high severity issues fluctuates, with notable peaks early in the commit history and subsequent dips, suggesting active remediation.
- **Flower & Cookiecutter:** Both repositories show a relatively stable trend in high severity issues, indicating that the same critical vulnerabilities persist over time.

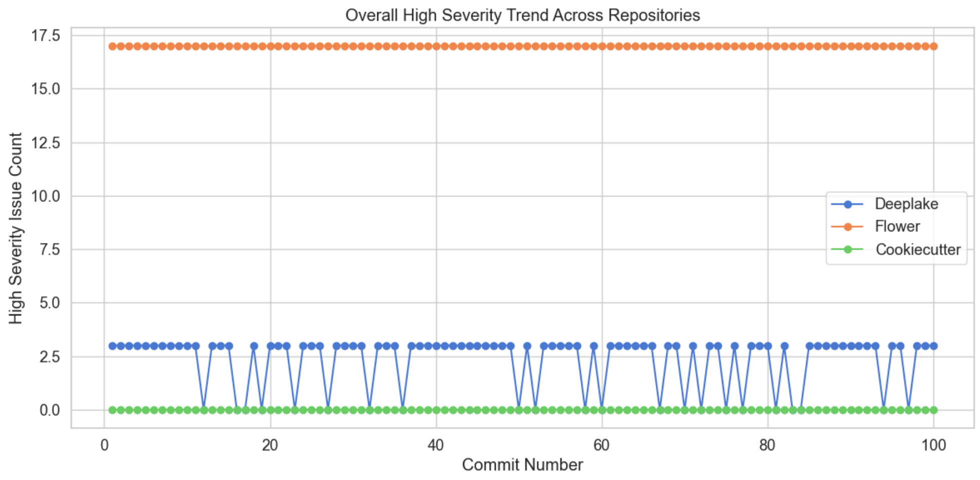

Figure 10: Overall trend of high severity issues across repositories. (Placeholder image)

Takeaway: The analysis suggests that while some repositories (e.g., Deeplake) show active remediation of high severity vulnerabilities, others maintain a consistent level of high severity issues, highlighting the need for targeted security improvements.

RQ2: Different Severity Patterns

Purpose: This question examines whether vulnerabilities of different severities (HIGH, MEDIUM, LOW) follow similar patterns of introduction and elimination.

Approach: I compared the trend lines for confidence and severity issues across commits for each repository. This involved overlaying the counts of high, medium, and low severity issues from the CSV data.

Results:

- In repositories like Deeplake, distinct variations were observed among different severity levels.
- In contrast, Flower and Cookiecutter display stable patterns across all severity levels.

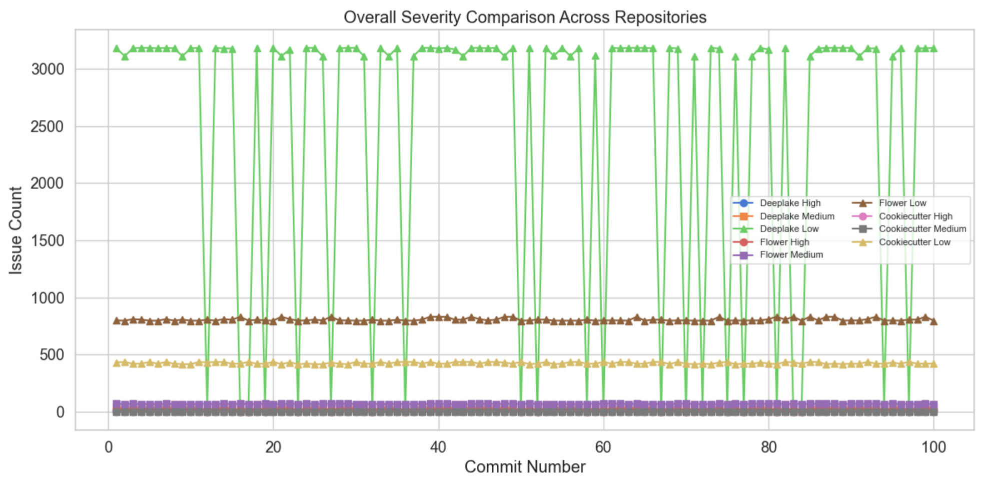

Figure 11: Comparison of vulnerability severity patterns. (Placeholder image)

Takeaway: The results indicate that some repositories experience dynamic changes in vulnerability severity, whereas others remain static. This difference can inform tailored security strategies.

RQ3: CWE Coverage

Purpose: This analysis identifies which CWE identifiers are most frequently reported across the analyzed repositories, providing insight into common vulnerability types in open-source projects.

Approach: I aggregated the total occurrences of each CWE from the CSV data and generated bar charts to visualize the frequency distribution across repositories.

Results:

- **Cookiecutter** consistently reports around 4 unique CWEs per commit.
- **Flower** shows a stable pattern with approximately 12 unique CWEs.
- **Deeplake** exhibits variability, with unique CWE counts ranging between 2 and 10.

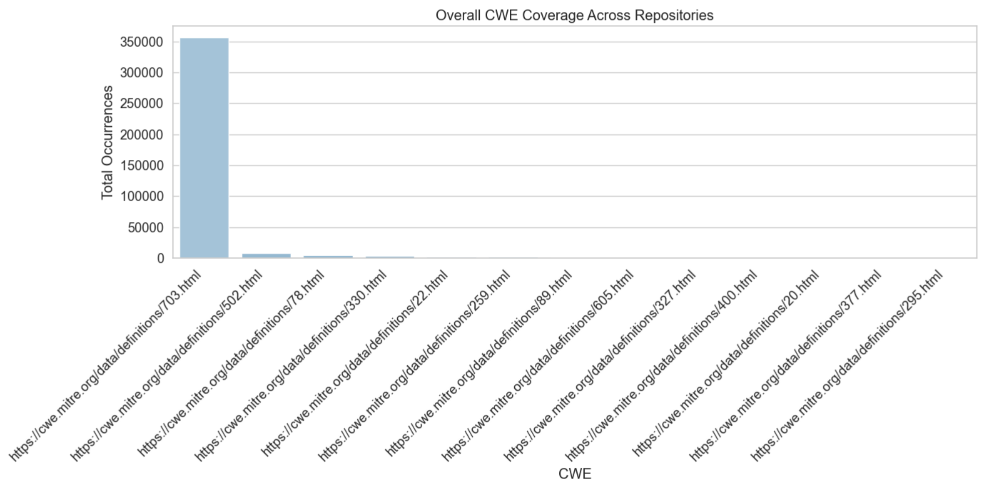

Figure 12: Overall CWE frequency distribution across repositories. (Placeholder image)

Takeaway: The aggregated data reveals that certain CWE types occur more frequently, suggesting that these vulnerabilities are common and may require focused attention in security audits and remediation efforts.

6 Discussion and Conclusion

6.1 Discussion

The analysis of Bandit's output across the three repositories yielded several important insights:

- **Repository-Level Observations:**
  - **Deeplake:** The unique CWE count fluctuated significantly, indicating dynamic changes in vulnerabilities—suggesting active introduction and remediation of issues during development.
  - **Flower:** The unique CWE count remained relatively constant at approximately 12 per commit, suggesting a persistent set of vulnerabilities throughout its history.
  - **Cookiecutter:** A stable pattern with around 4 unique CWEs per commit was observed, implying consistent code behavior with regard to vulnerability triggers.
- **Overall Dataset-Level Insights:**
  - **RQ1 (High Severity Vulnerabilities):** In repositories like Deeplake, high severity issues show peaks and dips, indicating phases of vulnerability introduction and subsequent remediation. In contrast, Flower and Cookiecutter exhibited relatively constant levels of high severity issues.
  - **RQ2 (Different Severity Patterns):** The trends for HIGH, MEDIUM, and LOW severity issues differ in repositories with dynamic development patterns

(e.g., Deeplake), while more stable projects (e.g., Flower and Cookiecutter) show similar patterns across all severity levels.

  - **RQ3 (CWE Coverage):** The overall CWE aggregation revealed that certain CWE identifiers are consistently reported across repositories. This indicates common vulnerabilities that may require prioritized remediation.

6.2 Challenges and Reflections

During the analysis, several challenges were encountered:

- **Data Consistency:** Ensuring the correct organization of CSV files and matching them to the appropriate repositories was critical.
- **Error Handling:** Handling missing or misformatted CSV files required additional debugging and adjustments in the code.
- **Interpreting Results:** Distinguishing between false positives and actual vulnerabilities in the Bandit reports required careful examination of the outputs.

These challenges provided valuable lessons in data preprocessing, tool configuration, and the importance of clear documentation.

6.3 Conclusion

In summary, the analysis demonstrates that:

- The dynamic changes in vulnerability patterns (especially in Deeplake) highlight the need for continuous security monitoring and proactive remediation.
- Stable patterns in repositories like Flower and Cookiecutter suggest either persistent vulnerabilities or false positives that warrant further review.
- Aggregated CWE data identifies common areas of concern across multiple projects, which can inform future security audits and code improvements.

The insights derived from this study provide a quantitative foundation for understanding and improving the security posture of open-source software repositories.