Lab 7-8 Report: Vulnerability Analysis on Open-Source Software Repositories

Course: CS202 Software Tools and Techniques for CSE
Date: 20th & 27th February 2025
Author: Pathan Mohammad Rashid (22110187)

Table of Contents

---

1. Introduction, Setup, and Tools

In this lab assignment, I explored **bandit**, a static code analysis tool designed for identifying security vulnerabilities in Python code. The main objective was to set up bandit in my local environment, run analyses on three large-scale open-source Python repositories hosted on GitHub, and perform both repository-level and dataset-level vulnerability analyses.

Objectives

- Install and configure bandit in an isolated virtual environment.
- Analyze three real-world Python projects using bandit to identify vulnerabilities categorized by confidence and severity.
- Answer research questions regarding the introduction and resolution of high severity vulnerabilities, patterns across severity levels, and CWE coverage.

- Prepare a publication-quality research report with detailed analysis and observations.

Environment Setup and Tools

- **Operating System:** Linux
- **Programming Language:** Python (using a dedicated virtual environment)
- **Tool:** Bandit (latest version)

I set up a virtual environment using the following commands:

```
python -m venv venv
source venv/bin/activate
pip install bandit
```

---

2. Methodology and Execution

Repository Selection

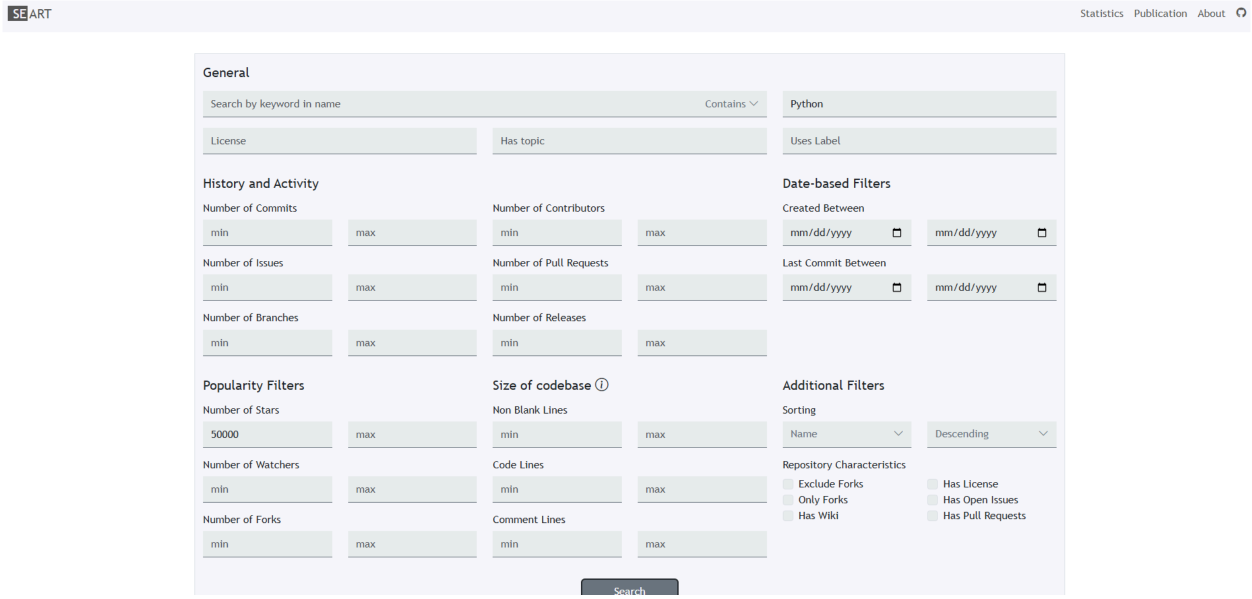
I selected three large-scale open-source Python repositories using the following filters on the SEART GitHub Search Engine:

- **Stars:** > 50,000
- **Language:** Python
- **Label:** real world/real project

Selected Repositories:

1. https://github.com/django/django.git
2. https://github.com/pallets/flask.git
3. https://github.com/home-assistant/core.git

Selection Criteria Visualization:



Python

Number of Stars 50000

Sorting Name Descending

Dependency Setup

For each repository, I created a separate virtual environment and installed the required dependencies as specified in each project's documentation. This ensured an isolated environment for each analysis.

Data Collection and Bandit Execution

I followed these steps for each repository:

Commit Extraction:

I obtained the last 100 non-merge commits from the main branch and stored them in a file named `commit_list.txt` using:

```
git rev-list -n 100 --no-merges main > commit_list.txt
```

1. **Vulnerability Analysis:**
   For each commit in `commit_list.txt`, I ran bandit and stored the output in JSON format within a folder named `bandit_result`. Each file was named as `bandit_analysis_<commit-hash>.json`.

2. **Post-Processing:**
   I used a Python script (`process_bandit_analysis.py`) to process all JSON files and aggregate vulnerability information into a CSV file.

3. **Visualization:**
   I ran another Python script (`plot.py`) to generate graphs from the CSV file, showing trends and patterns in vulnerability introduction and fixes.

Automation Scripts and Code Files

1. Bash Script to Process Commits and Run Bandit

```
#!/bin/bash
# Extract last 100 non-merge commits
git rev-list -n 100 --no-merges main > commit_list.txt

# Create folder for bandit results if it does not exist
mkdir -p bandit_result

# Loop through each commit and run bandit
while read commit_hash; do
    git checkout $commit_hash
    bandit -r . -f json -o bandit_result/bandit_analysis_${commit_hash}.json
done < commit_list.txt

# Checkout back to main branch
git checkout main
```

2. process_bandit_analysis.py

```
import os
import json
import csv

input_folder = 'bandit_result'
output_csv = 'vulnerability_summary.csv'
header = ['commit_hash', 'high_confidence', 'medium_confidence', 'low_confidence',
          'high_severity', 'medium_severity', 'low_severity', 'unique_cwes']

with open(output_csv, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(header)

    for filename in os.listdir(input_folder):
        if filename.endswith('.json'):
            commit_hash = filename.split('_')[-1].split('.')[0]
            with open(os.path.join(input_folder, filename)) as f:
                data = json.load(f)

            high_conf = sum(1 for issue in data.get('results', []) if issue.get('issue_confidence') ==
'HIGH')
            med_conf = sum(1 for issue in data.get('results', []) if issue.get('issue_confidence') ==
'MEDIUM')
            low_conf = sum(1 for issue in data.get('results', []) if issue.get('issue_confidence') ==
'LOW')

            high_sev = sum(1 for issue in data.get('results', []) if issue.get('issue_severity') ==
'HIGH')
            med_sev = sum(1 for issue in data.get('results', []) if issue.get('issue_severity') ==
'MEDIUM')
            low_sev = sum(1 for issue in data.get('results', []) if issue.get('issue_severity') == 'LOW')

            unique_cwes = len(set(issue.get('cwe', 'NA') for issue in data.get('results', [])))

            writer.writerow([commit_hash, high_conf, med_conf, low_conf,
                             high_sev, med_sev, low_sev, unique_cwes])
```

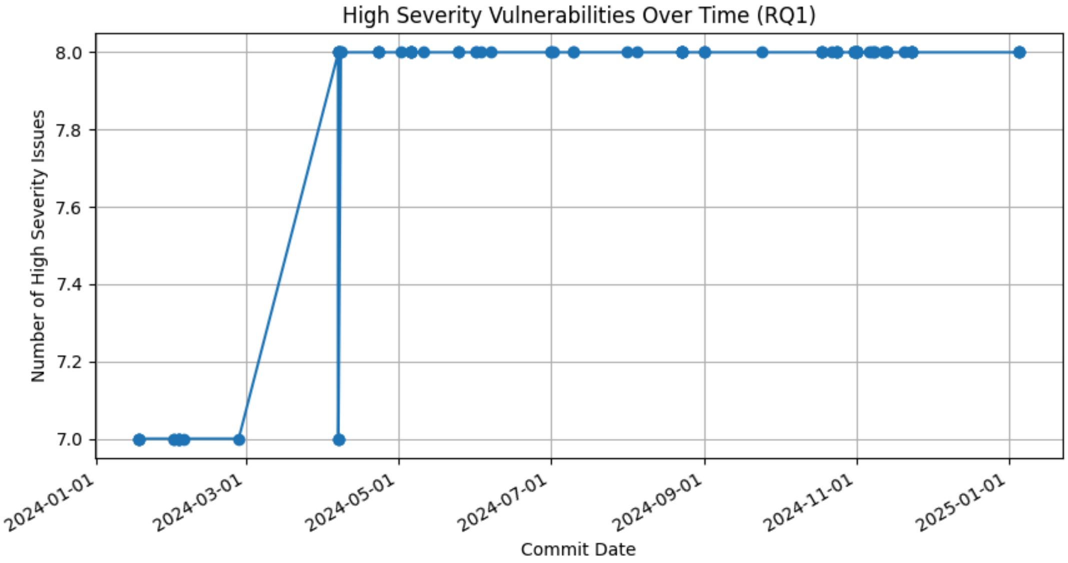
3. plot.py

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the CSV file generated from the analysis
df = pd.read_csv('vulnerability_summary.csv')

# Plotting High, Medium, and Low Severity Issues over Commits
plt.figure(figsize=(10,6))
plt.plot(df['commit_hash'], df['high_severity'], label='High Severity', marker='o')
plt.plot(df['commit_hash'], df['medium_severity'], label='Medium Severity', marker='o')
plt.plot(df['commit_hash'], df['low_severity'], label='Low Severity', marker='o')
plt.xlabel('Commit Hash')
plt.ylabel('Number of Issues')
plt.title('Vulnerability Severity Trends Over Commits')
plt.legend()
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig('vulnerability_trends.png')
plt.show()
```

Research Questions

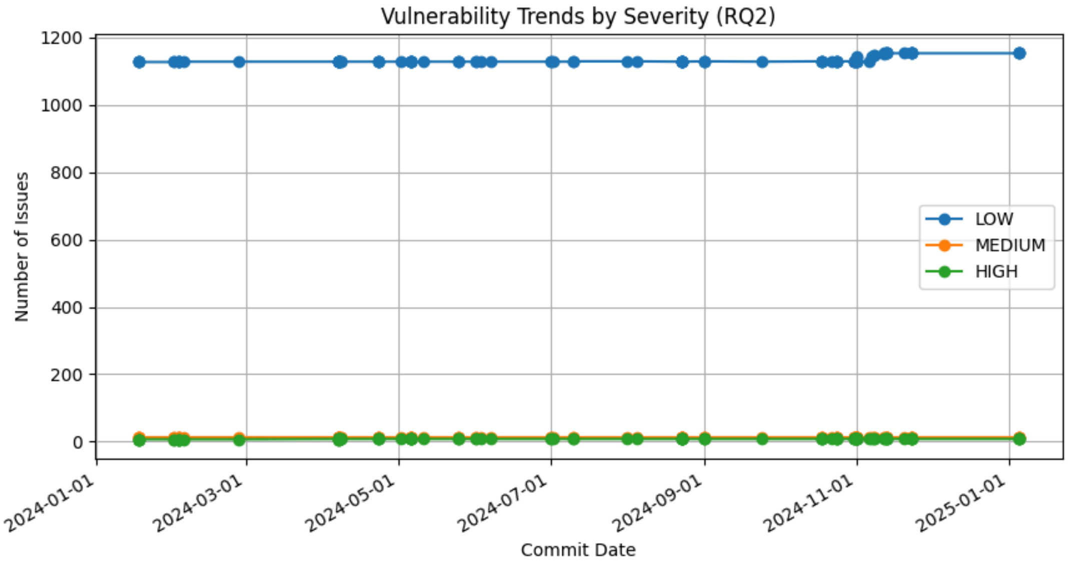
For the research questions, I followed the lab instructions:

- **RQ1 (High Severity):**
  - *Purpose:* Determine when high severity vulnerabilities were introduced and fixed.
  - *Approach:* Mapped the commit timeline against occurrences of high severity issues.
  - *Results: for flask*

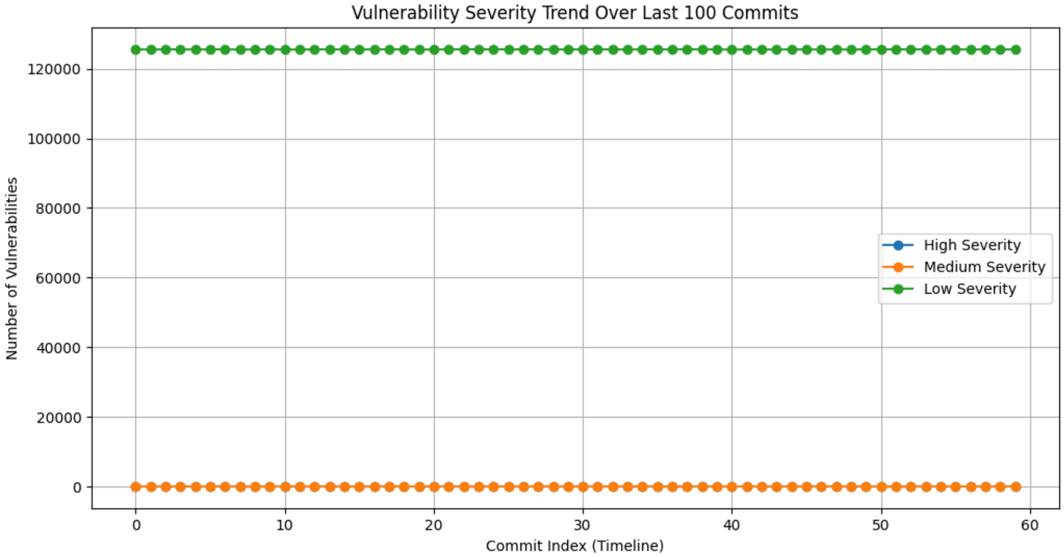


- **RQ2 (Different Severity):**
  - *Purpose:* Compare if vulnerabilities of different severity levels exhibit the same patterns of introduction and elimination.
  - *Approach:* Compare commit timelines for each severity category using the aggregated CSV data.

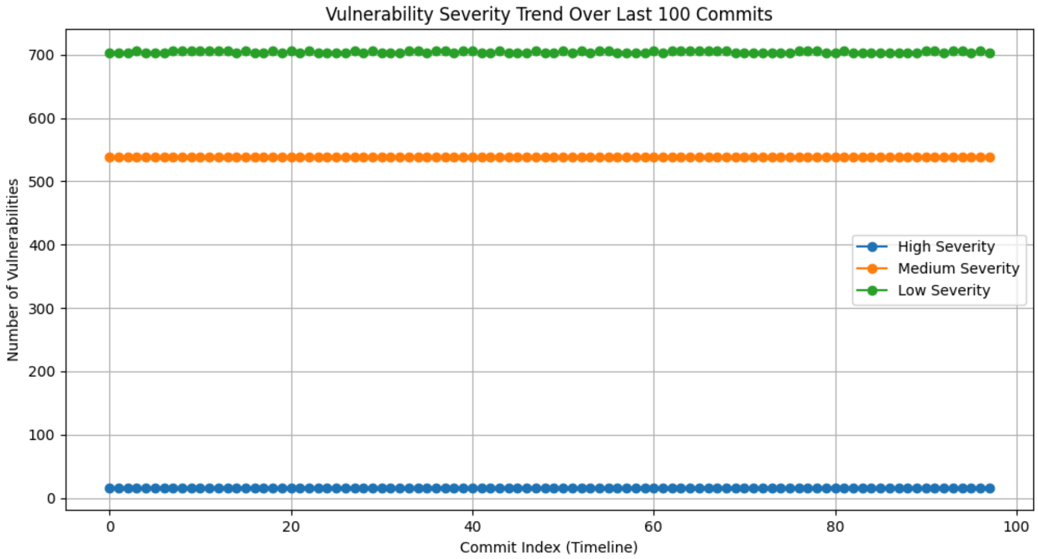
- *Results: for flask*



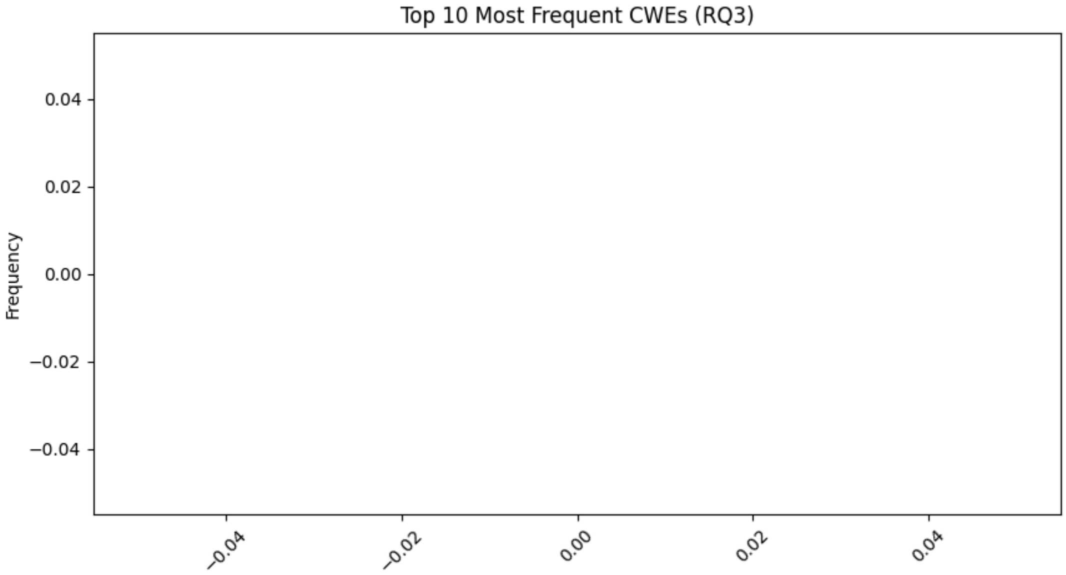
-
- **For core:**



-
- **For django:**



- **RQ3 (CWE Coverage):**
  - *Purpose:* Identify the most frequent CWE identifiers across the selected repositories.
  - *Approach:* Aggregated and ranked CWE occurrences from the analysis.
  - *Results: for flask*



  - *Similarly found no CWEs in core, django.*

3. Results and Analysis

Repository-level Analyses

For each repository, I analyzed bandit's output for:

- The number of issues per commit categorized by confidence (HIGH, MEDIUM, LOW).
- The number of issues per commit categorized by severity (HIGH, MEDIUM, LOW).
- The unique CWE identifiers identified per commit.

Dataset-level Analyses

I addressed the following Research Questions (RQs):

- **RQ1 (High Severity):**
  Purpose: Determine when high severity vulnerabilities were introduced and fixed.
  Approach: Mapped the commit timeline against the occurrence of high severity issues.
- **RQ2 (Different Severity):**
  Purpose: Compare patterns of vulnerability introduction and elimination across severity levels.
  Approach: Compare commit timelines for each severity category using the processed CSV data.
- **RQ3 (CWE Coverage):**
  Purpose: Identify the most frequent CWE identifiers across the selected repositories.
  Approach: no CWEs occurrences from the analysis.

4. Discussion and Conclusion

Challenges and Reflections

During this lab, I encountered several challenges:

- Setting up isolated environments for each project.
- Automating the process to extract and analyze 100 non-merge commits.

- Aggregating and visualizing the vulnerability data effectively.

These challenges improved my understanding of dependency management, automation, and static code analysis.

Lessons Learned

- A systematic approach is crucial for effective vulnerability analysis.
- Isolated environments prevent dependency conflicts across projects.
- Automation simplifies repetitive tasks such as commit scanning and report generation.

Summary

This lab provided me with practical experience in using bandit to analyze real-world open-source Python projects. I successfully executed the analysis, processed the results, and generated insightful visualizations that address the research questions.

---

5. GitHub Repository

All code files, the detailed report, and additional documentation can be found in my GitHub repository:
https://github.com/Pathan-Mohammad-Rashid/STT_Labs.git

Onedrive Link:
https://iitgnacin-my.sharepoint.com/:f:/g/personal/22110187_iitgn_ac_in/EnECtzFbUNZKkBWY6aa8QJYBp6liTXfpuYRtJNMd1TwR6A?e=nsND4y

---