Vulnerability Management with DefectDojo

> DevOps and Security - Project Report
>
> Ahmed Nouralla - a.shaaban@innopolis.university

Introduction

Vulnerability management is the "cyclical practice of identifying, classifying, prioritizing, remediating, and mitigating software vulnerabilities".

A typical DevSecOps CI pipeline may involve the following steps:

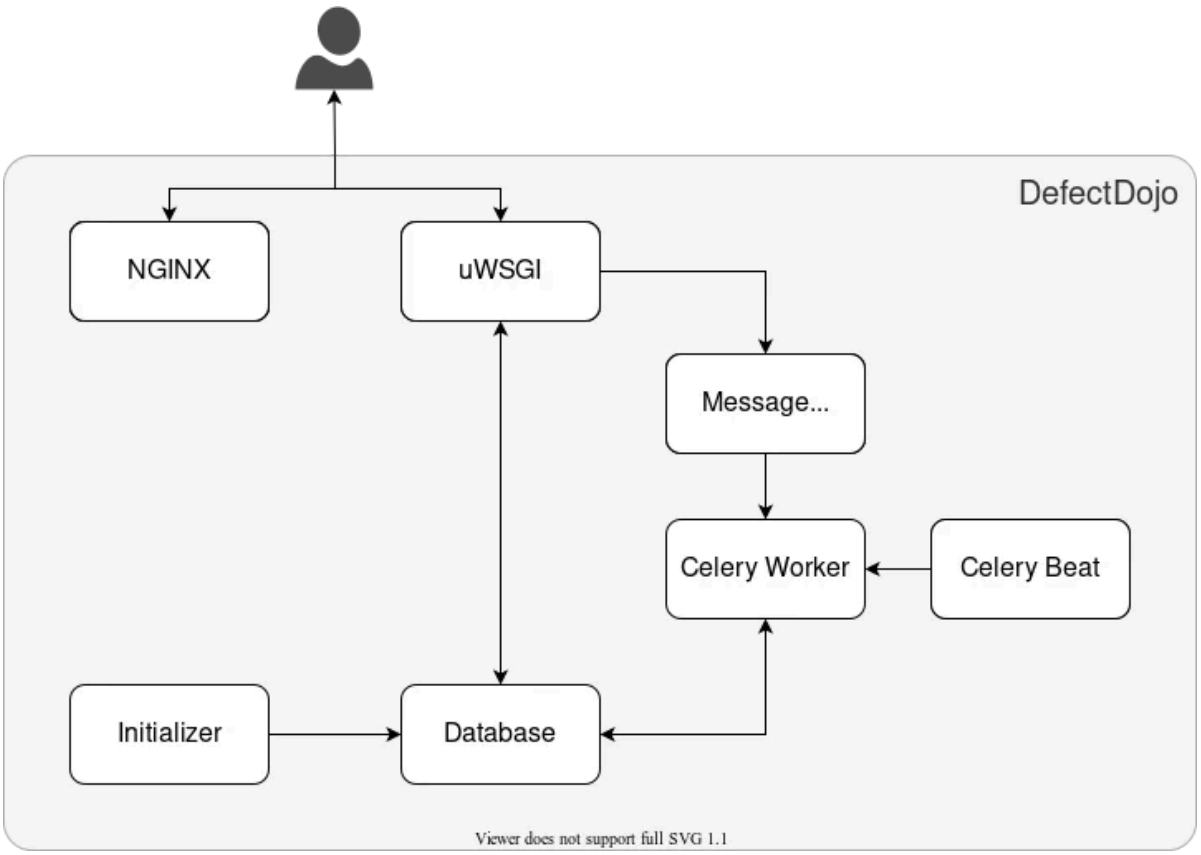
1. Running one or more security scanners against the application source code, this may include:
   - **Static Application Security Testing:** analyzing code for bad practices without running it.
   - **Dynamic Application Security Testing:** interactive security scanning by simulating attacks.
   - **Software Composition Analysis:** identification (e.g., generating SBOM) of 3rd party components used by a software to help reduce risks associated with using them. Tools for container/dependency scanning lie within this category.
   - **Secret Detection:** identifying leaked credentials or keys in source code.
2. Aggregate reports from such tools, cleaning/re-formating them if needed.
3. Upload results to a central vulnerability management tool for an auditor to inspects results for further analysis and mitigation.

Popular open-source tools providing vulnerability management dashboards include DefectDojo, Faraday, ArcheySec, and Wazuh.

Methods

This work implements a simple DevSecOps pipeline scenario with GitLab, DefectDojo, and Three SAST tools (Bandit, NjsScan, Flawfinder).

System Architecture



- Nginx: the webserver delivering all static content, e.g. images, JavaScript files or CSS files.
- uWSGI is the application server that runs the DefectDojo platform, written in Python/Django, to serve all dynamic content.
- Redis: the application server sends tasks to a Message Broker for asynchronous execution.
- Celery Worker: tasks like deduplication or the JIRA synchronization are performed asynchronously in the background by the Celery Worker.
- Celery Beat: In order to identify and notify users about things like upcoming engagements, DefectDojo runs scheduled tasks using Celery Beat.
- PostgreSQL: database storing all the application data of DefectDojo.
- **Initializer:** setups/maintains the database and syncs/runs migrations after version upgrades. It shuts itself down after all tasks are performed.

Supported Features

- Integrations with 150+ scanners and security tools for SAST, DAST, and SCA.
- Bi-directional integration with JIRA.
- Automated deduplication of findings to reduce noise.
- Automatic scan imports in CI/CD through the API.

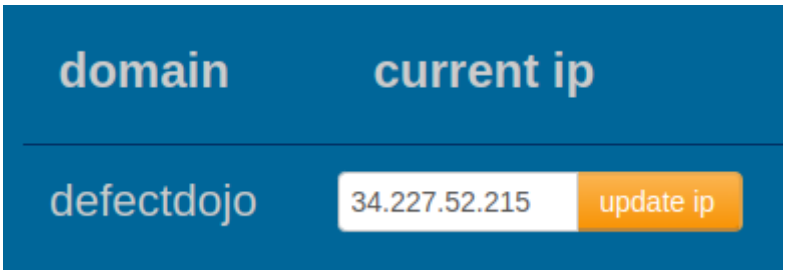
Deployment Options

- The recommended option to deploy DefectDojo is thorough docker compose or their managed SaaS platform.
- Running DefectDojo in Docker in a cloud VM following the docs

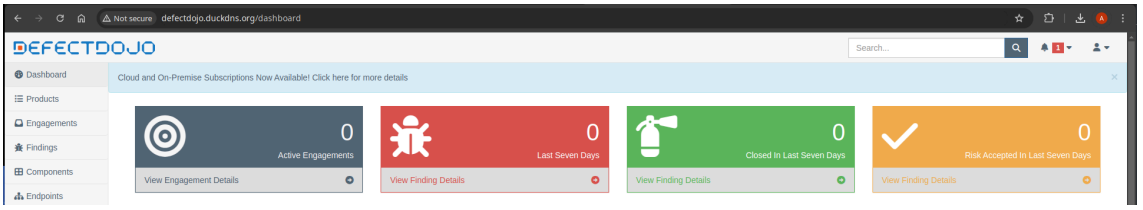
```
git clone https://github.com/DefectDojo/django-DefectDojo
cd django-DefectDojo
docker compose build && docker compose up -d
docker compose logs -f initializer | grep "Admin password:"
```

```
ubuntu@ip-172-31-23-252:~/django-DefectDojo$ docker compose up -d
[+] Running 7/7
 ✔ Container django-defectdojo-postgres-1       Started
 ✔ Container django-defectdojo-redis-1          Started
 ✔ Container django-defectdojo-celeryworker-1   Started
 ✔ Container django-defectdojo-initializer-1    Started
 ✔ Container django-defectdojo-celerybeat-1     Started
 ✔ Container django-defectdojo-nginx-1          Started
 ✔ Container django-defectdojo-uwsgi-1          Started
```

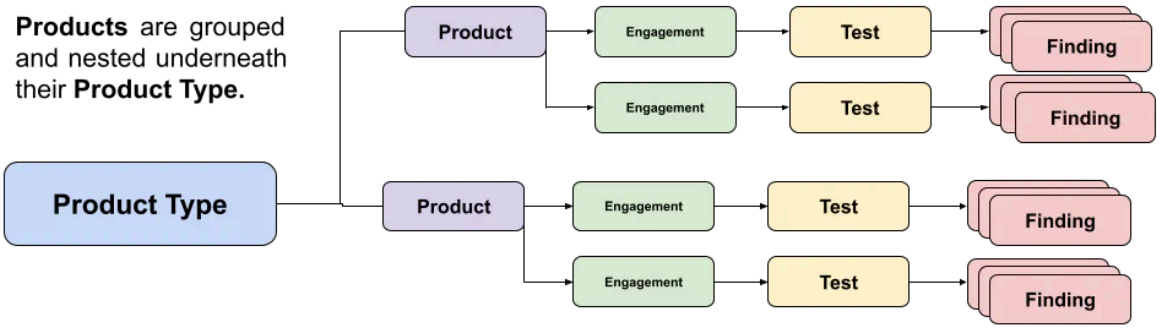
- Added a dynamic DNS record for the server to be accessible at http://defectdojo.duckdns.org



- Logged in with `admin` and the password from initializer logs.

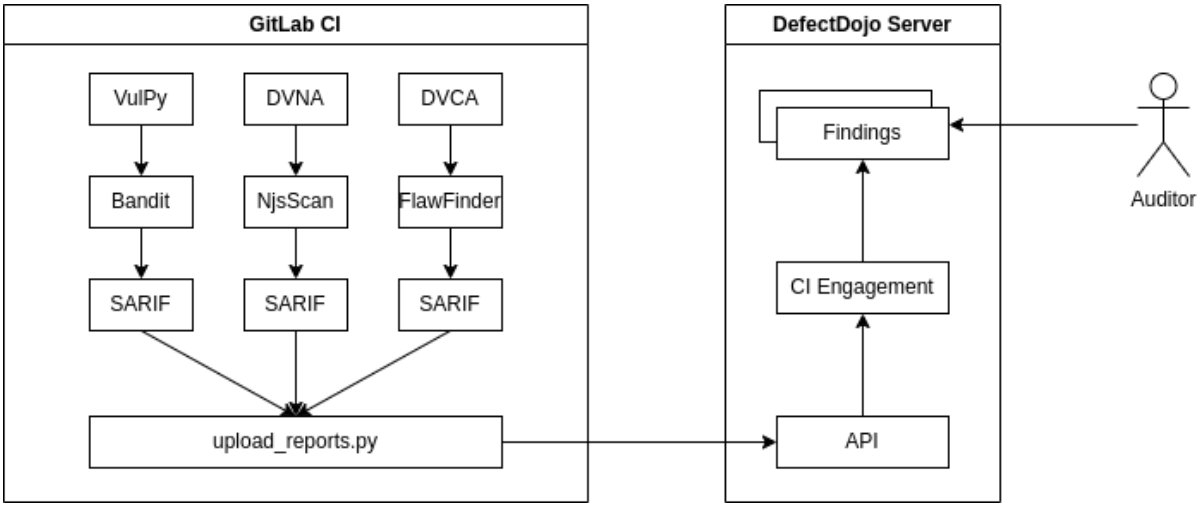


Product Hierarchy

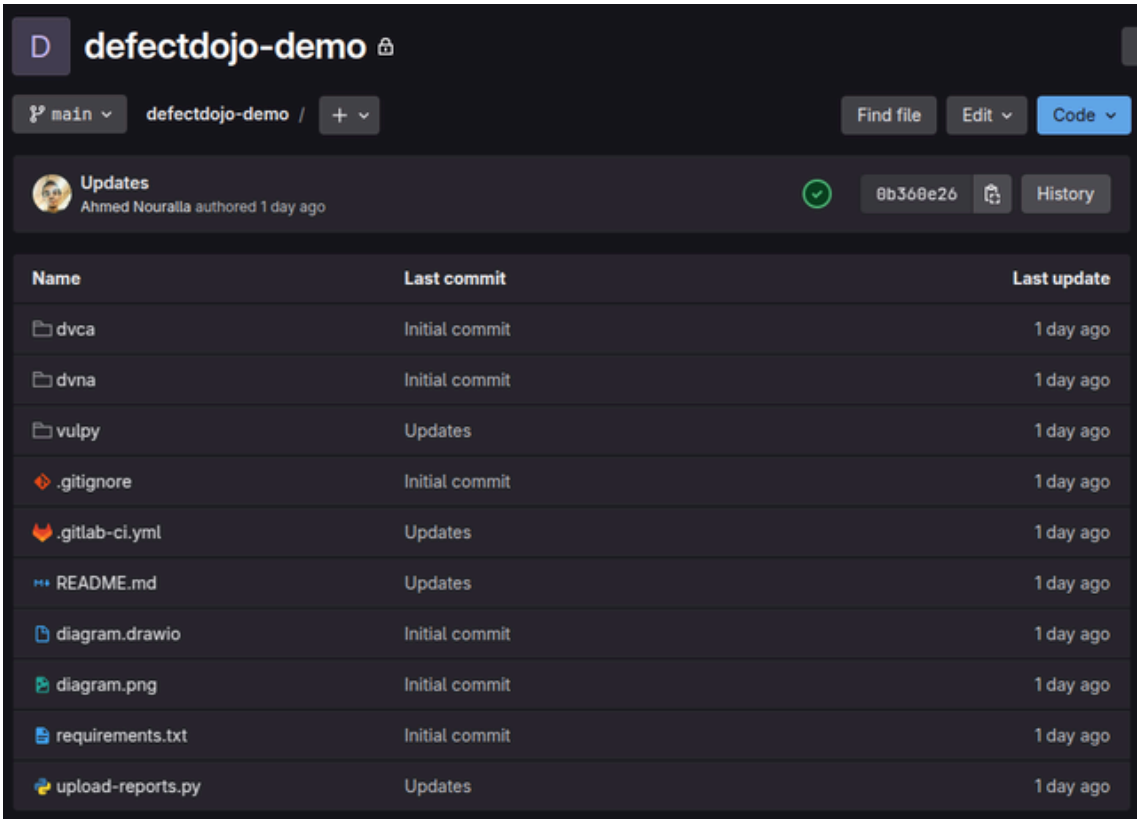


- **Product Type:** high-level categorization of products
- **Product:** individual unit to be tested (e.g., a certain version/build)
  - **Engagement:** represents moments in time when testing takes place
    - Interactive engagement: manual scan imports from the UI
    - CI/CD engagements: automatic result imports from pipelines.
- **Tests:** grouping of activities conducted to attempt to discover flaws in a product.
- **Findings:** representing specific flaws discovered while testing.

Use-case



1. Cloned three sample vulnerable-by-design projects: VulPy, DVNA, and DVCP.



   - **Vulpy:** Flask (Python) WebApp with SQLite database.
   - **DVNA:** Express (NodeJS) WebApp with MySQL database.
   - **DVCA:** C Program with utilities to read and process images.

2. Added a `.gitlab-ci.yml` that utilizes three SAST tools (Bandit, NjsScan, FlawFinder) to scan the respective projects in CI.

```
stages:
  - security-scan
  - upload-reports

security-scan:
  stage: security-scan
  image: python:3-alpine
  before_script:
```

```
    - pip3 install bandit njsscan flawfinder
  script:
    - bandit -r vulpy/ -f sarif --exit-zero -o bandit_report.sarif
    - njsscan --sarif dvna/ -o njsscan_report.sarif || true
    - flawfinder --sarif dvca/ > flawfinder_report.sarif
...
```

3. Scan reports are collected as artifacts in SARIF format

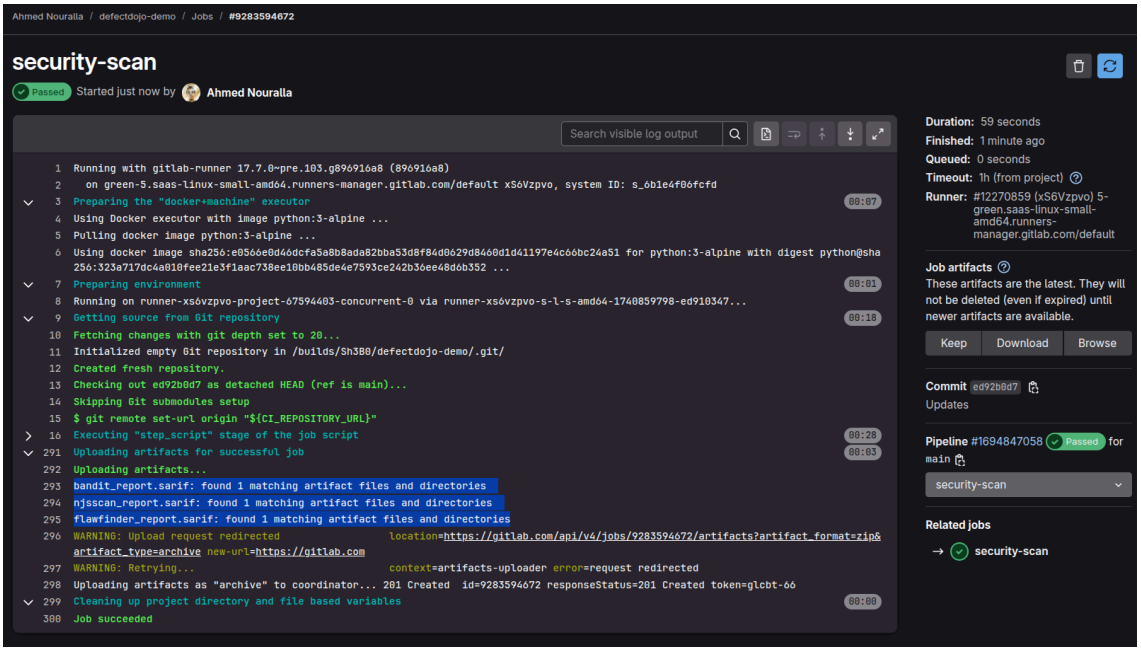
```
...
  artifacts:
    paths:
      - bandit_report.sarif
      - njsscan_report.sarif
      - flawfinder_report.sarif
...
```

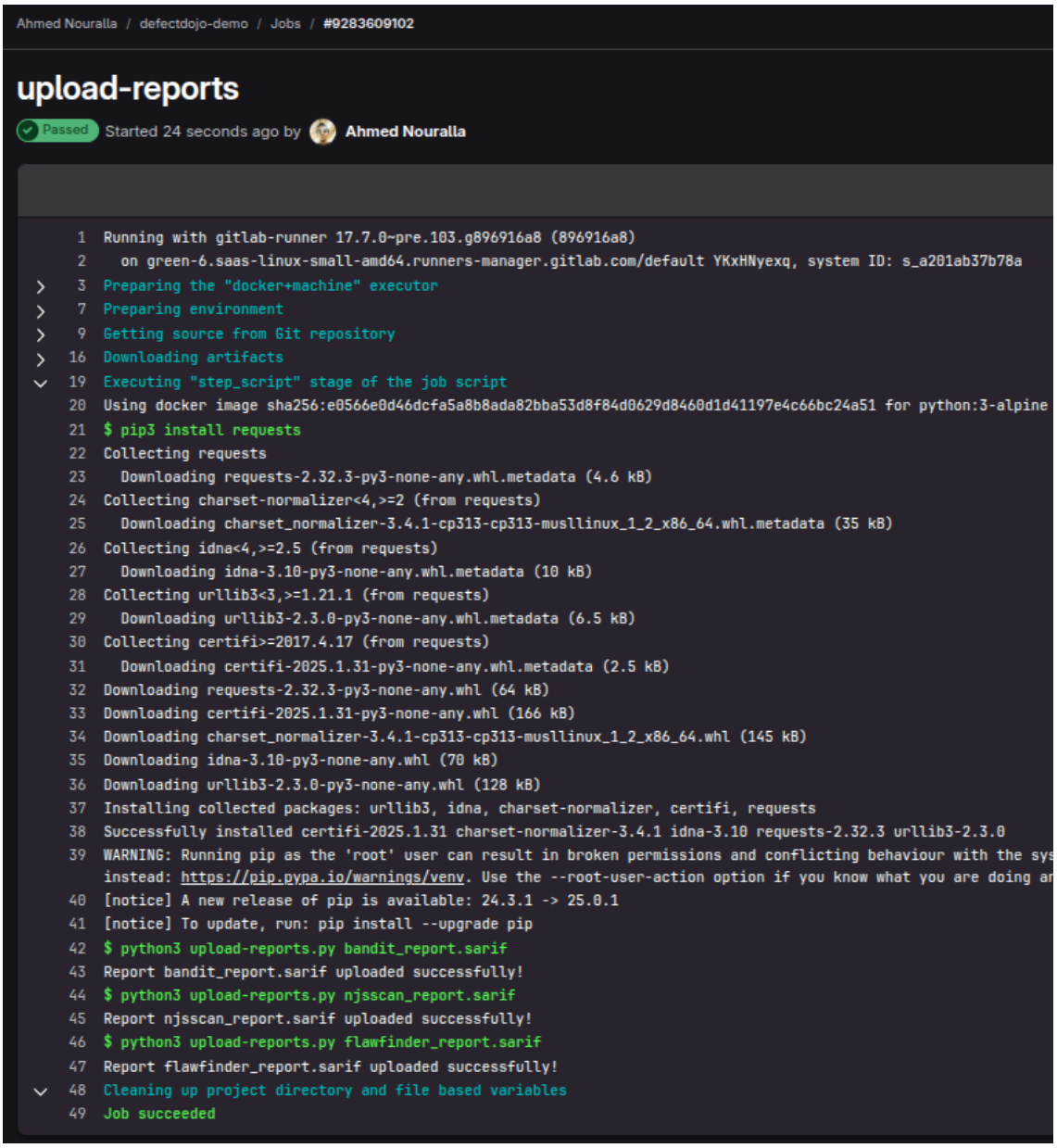
4. A Python script is used to connect to DefectDojo API to upload scan results.
    - API token was obtained from server and configured in GitLab as a secret variable `DD_API_TOKEN` to be used by the Python script.

```
...
upload-reports:
  stage: upload-reports
  image: python:3-alpine
  needs: ["security-scan"]
  before_script:
    - pip3 install requests
  script:
    - python3 upload-reports.py bandit_report.sarif
    - python3 upload-reports.py njsscan_report.sarif
    - python3 upload-reports.py flawfinder_report.sarif
```

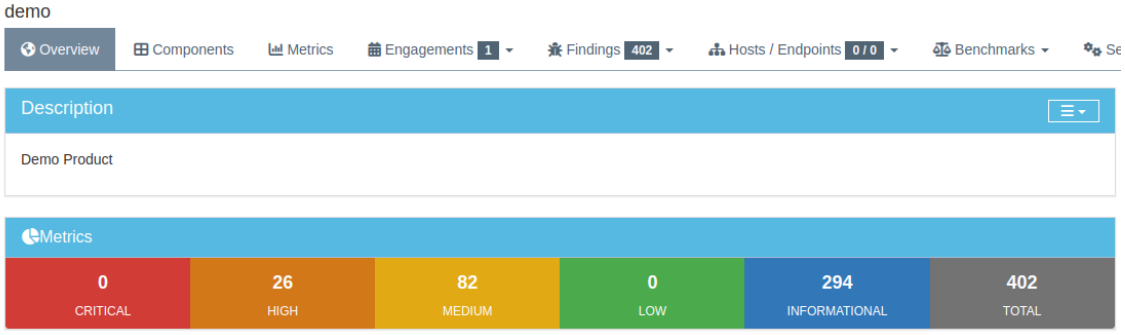
Results

- CI Run

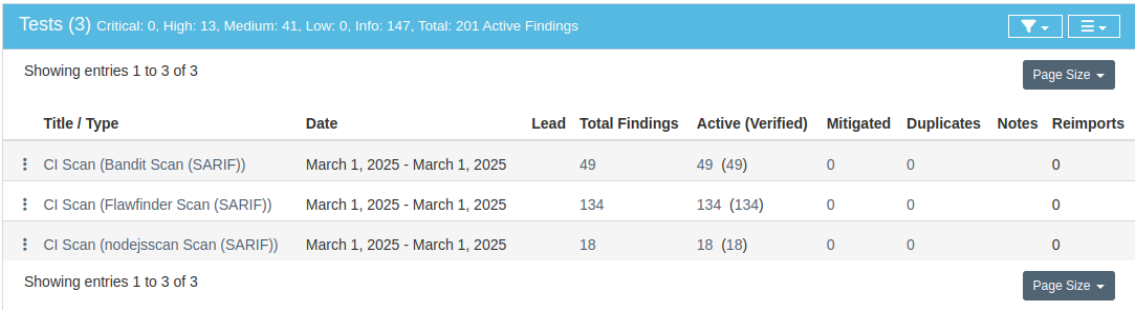




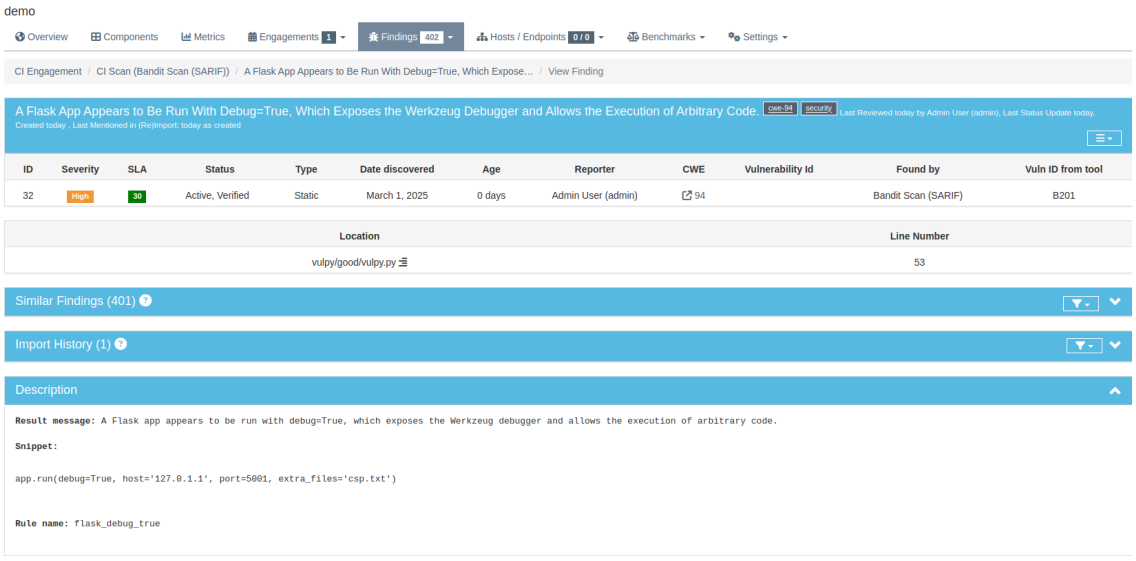
- Findings were imported to DefectDojo.



- Tests



- Finding view



Discussion

Workflow improvements:

- Integrate email notification for new findings
- Merge and deduplicate similar findings with different contexts.
- Create different user accounts and groups for different types of users
- Optimize tool settings to avoid reporting false positives

Server Monitoring:

- One can expose metrics from Django used by DefectDojo services by setting the variable `DD_DJANGO_METRICS_ENABLED`.
- These metrics can later be consumed by other tools (e.g., Prometheus)
- An open-source exporter for collecting metrics is also available.

References

- https://docs.defectdojo.com/en/open_source/installation/architecture/s
- https://docs.defectdojo.com/en/open_source/installation/running-in-production/
- https://docs.defectdojo.com/en/working_with_findings/organizing_engagements_tests/product_hierarchy/
- https://github.com/PyCQA/bandit
- https://github.com/ajinabraham/njsscan
- https://github.com/david-a-wheeler/flawfinder
- https://github.com/fportantier/vulpy
- https://github.com/appsecco/dvna
- https://github.com/hardik05/Damn_Vulnerable_C_Program
- https://docs.oasis-open.org/sarif/sarif/v2.0/csprd01/sarif-v2.0-csprd01.html