Vulnerability Management with DefectDojo

DevOps & Security - Course Project

Ahmed Nouralla

Content

- **Introduction**
  - Vulnerability Management
  - DefectDojo Architecture, Features, and Essential Concepts
- **Methods**
  - Server Deployment
  - Implemented use-case
- **Results**
- **Discussion**
- **References**

Intro: Vulnerability Management

Vulnerability management is the cyclical practice of identifying, evaluating (classifying/prioritizing), addressing (mitigating/remediating) and reporting software vulnerabilities.

Practical DevSecOps Pipeline:

1. Run one or more security scanners against the application source code
   - E.g., SAST, DAST, SCA, and Secret Detection tools.
2. Aggregate and evaluate reports from such tools
   - May need some cleaning/reformatting of findings.
3. Upload results to a central **vulnerability management** tool
   - E.g., Automatically in CI/CD using a script
4. Inspect results for further analysis and remediation

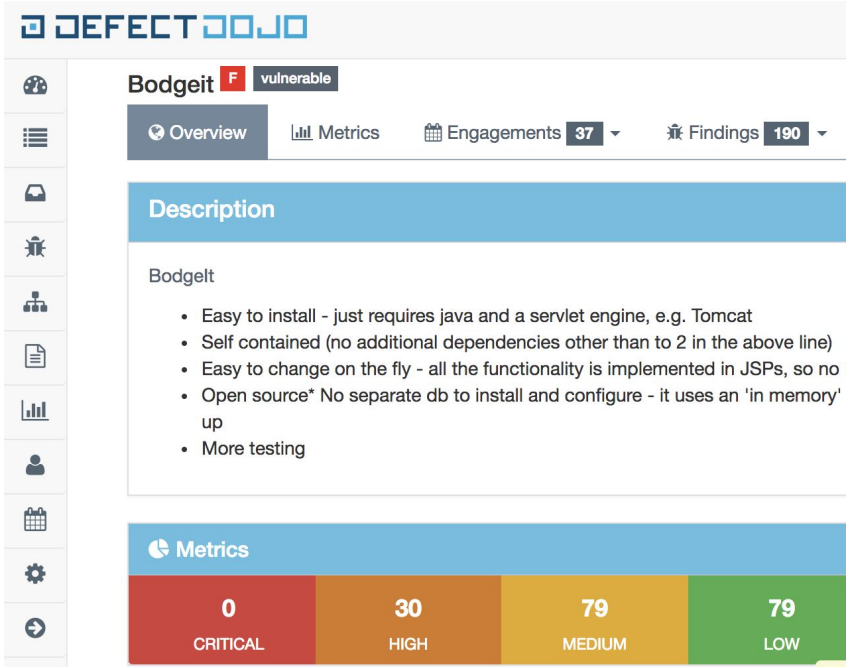
https://www.microsoft.com/en-us/security/business/security-101/what-is-vulnerability-management

DefectDojo Features

A popular vulnerability management solution written in Python.

It provides:

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

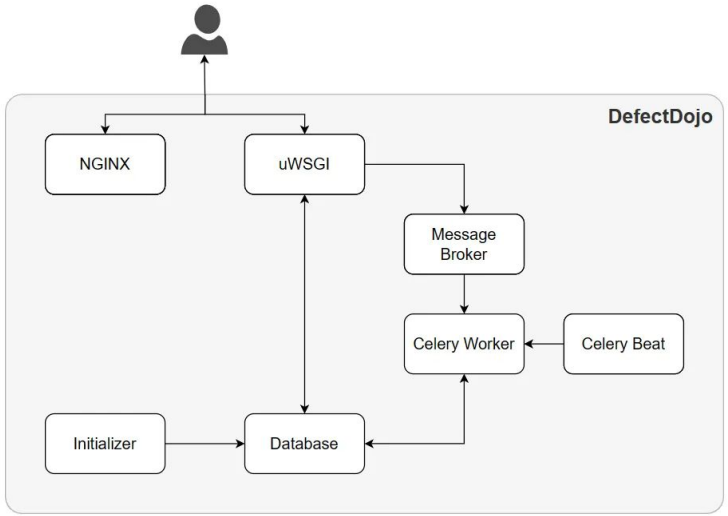


DefectDojo Interface

DefectDojo Architecture

DefectDojo server utilizes multiple components:

- **Nginx**: webserver delivering all static content
- **uWSGI**: application server serving all dynamic content.
- **Redis**: message broker for asynchronous execution of tasks.
- **Celery Worker**: performs tasks like finding deduplication or JIRA synchronization asynchronously in the background
- **Celery Beat:** to run periodic scheduled tasks.
- **PostgreSQL**: database storing all the application data.
- **Initializer**: setups/maintains the database and syncs/runs migrations after version upgrades. It shuts itself down after all tasks are performed.

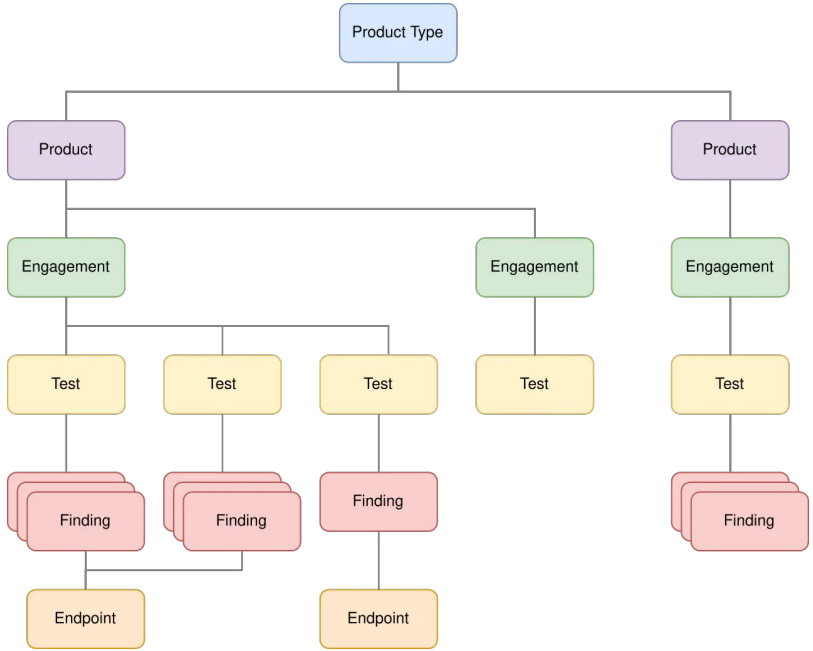


DefectDojo Components

Essential Concepts

Data classes in DefectDojo:

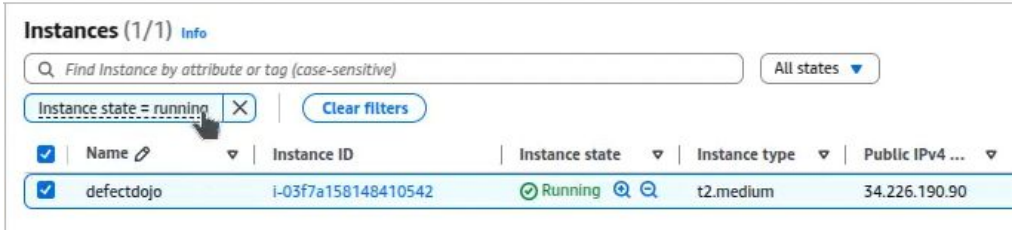
- **Product Type:** high-level categorization of products
- **Product:** individual unit to be tested (e.g., a certain version)
- **Engagement:** moments in time when testing takes place.
  - **Interactive engagement:** manual scan imports from the UI
  - **CI/CD engagements:** automatic result imports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

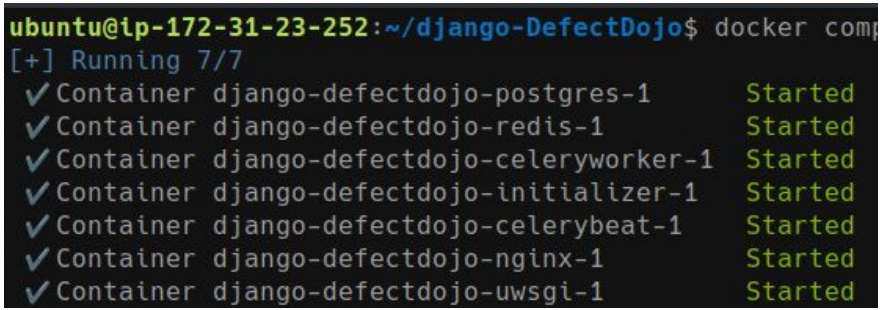


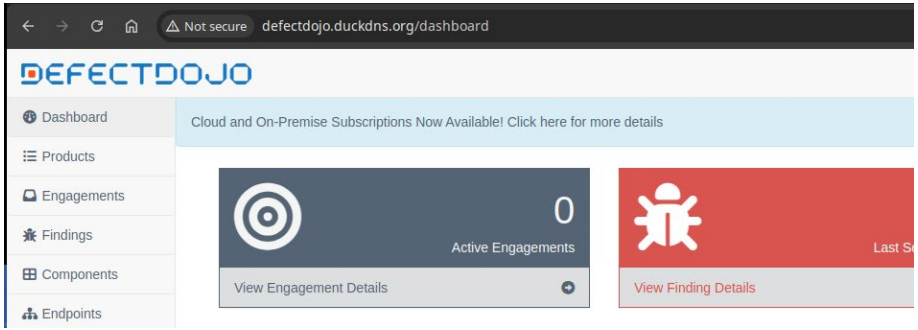
DefectDojo Product Hierarchy

Methods: Server Deployment

- DefectDojo recommends deployment through **Docker Compose**
- For enterprises, a managed **SaaS platform** is also offered
- **Local** or **K8s** installation is possible but not officially supported.

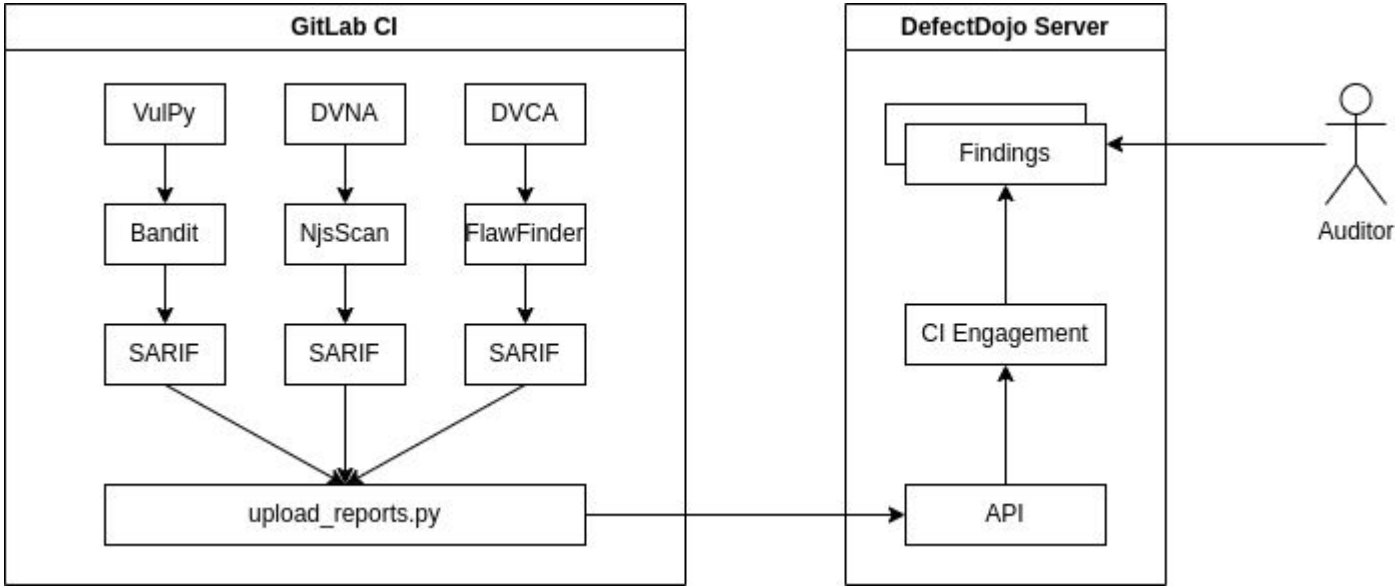






Use Case

- Cloned three sample vulnerable-by-design projects: VulPy (Python), DVNA (NodeJS), and DVCA (C).
- Ran three SAST tools (Bandit, NjsScan, and FlawFinder) to scan the projects in CI and produce SARIF reports.
- Wrote a Python script to automatically import scan reports (CI engagements) to DefectDojo via its API.



Technical Details: Pipeline and Script

.gitlab-ci.yml > [ ] stages

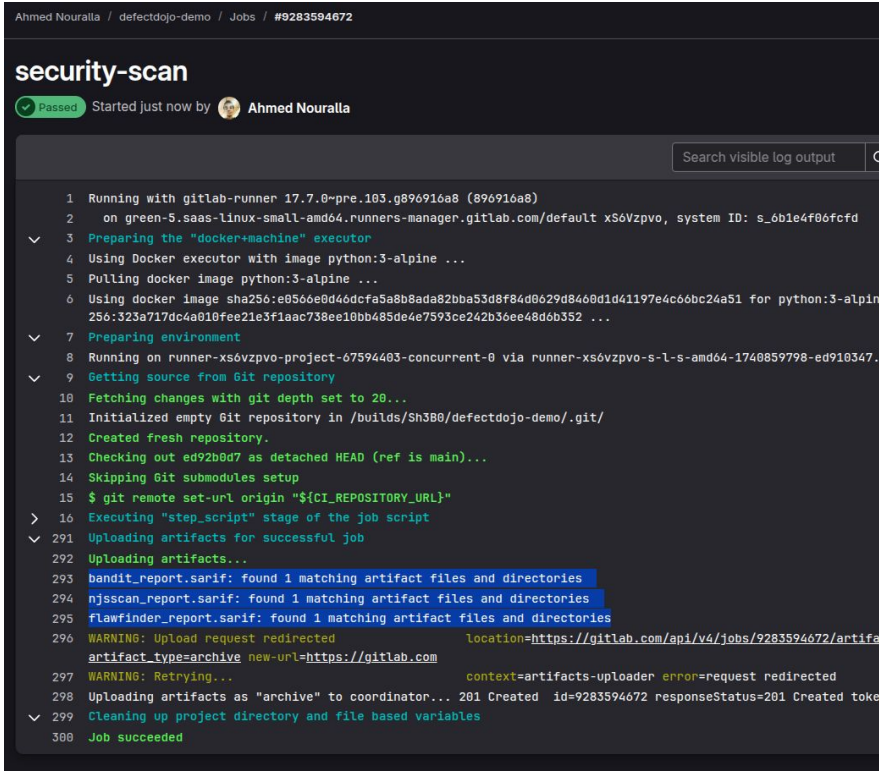
gitlab-ci - GitLab CI Configuration file (ci.json)

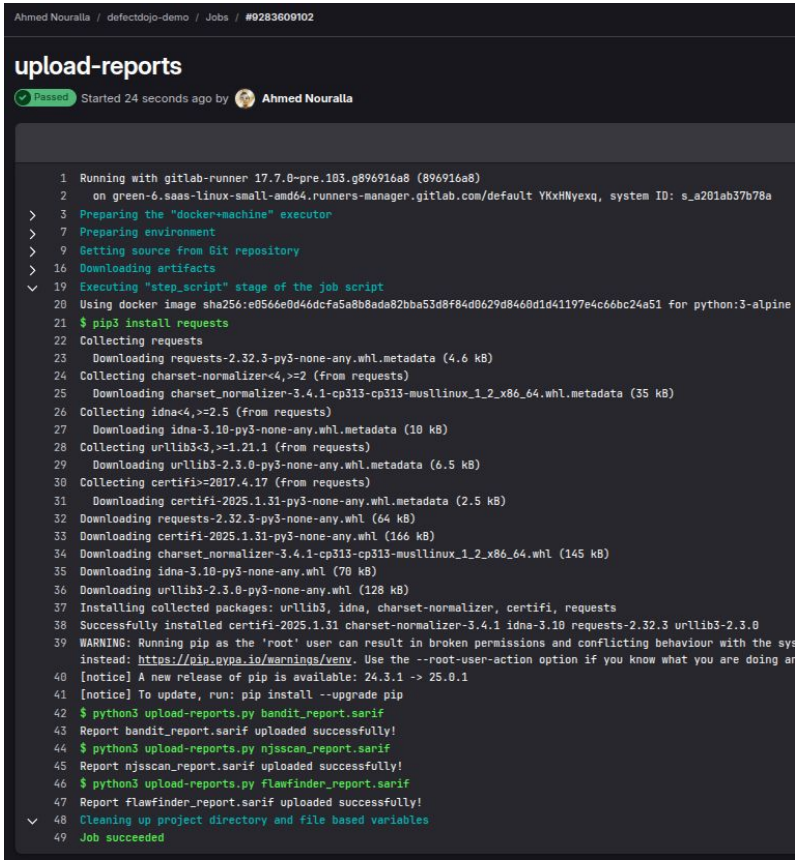
```yaml
stages:
  - security-scan
  - upload-reports

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

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
```

upload-reports.py > ...

```python
import requests
from os import environ
from sys import argv

headers = {
    'Authorization': f'Token {environ["DD_API_TOKEN"]}'
}

url = 'http://defectdojo.duckdns.org/api/v2/import-scan/'

data = {
    'active': True,
    'verified': True,
    'scan_type': 'SARIF',
    'auto_create_context': True,
    'product_name': 'demo',
    'engagement_name': 'CI Engagement',
    'test_title': 'CI Scan',
}

files = {
    'file': open(argv[1], 'r')
}

res = requests.post(url, headers=headers, data=data, files=files)

if res.status_code == 201:
    print(f"Report {argv[1]} uploaded successfully!")
else:
    print(f"Report {argv[1]} failed to upload.\n{res.content}")
```

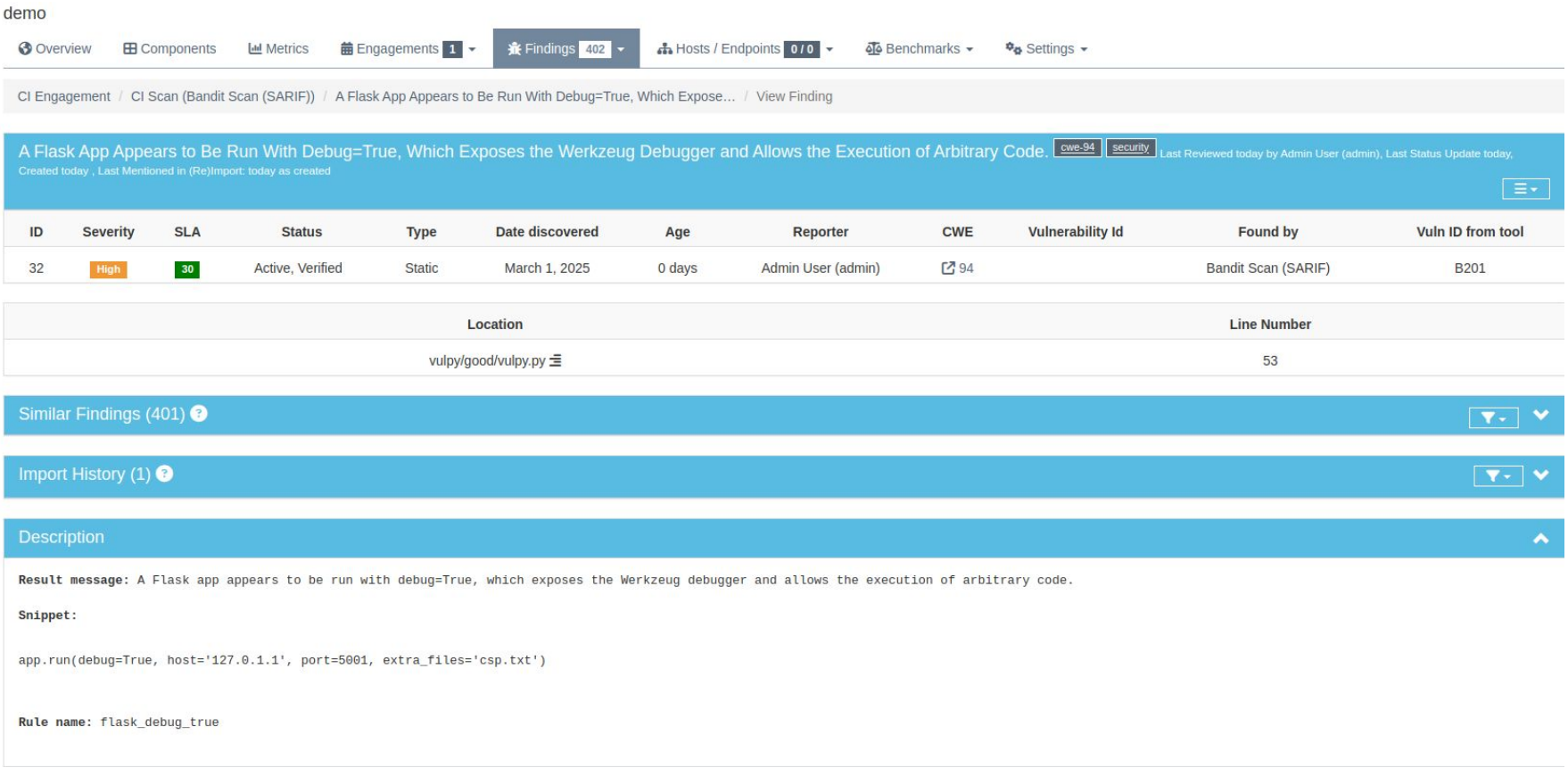
Results: Automatic Scans and Imported Findings





Findings are shown in DefectDojo UI

Inspecting a Finding



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://www.microsoft.com/en-us/security/business/security-101/what-is-vulnerability-management
- 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://docs.oasis-open.org/sarif/sarif/v2.0/csprd01/sarif-v2.0-csprd01.html
- https://docs.gitlab.com/ci/quick_start/
- 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