**Juice Shop**

**SAST:**

Allow\_failure: true

Job is allowed to fail, since we only want to catch the vulnerabilities and don’t want to block the entire pipeline for it.

artifacts: always

it always generates artifacts even if the job is cancelled or failed

reports:

sast: semgrep-results.json

the file is declared as SAST job report so it can be stored and found in gitlab under the job called sast.

The file is formatted in .JSON so that we process it in our Vulnerability dashboard

expire\_in: 3 weeks

was done this way in case we don’t run our VD days or hours before the presentation it has enough time before the job artifacts expire. Since expired artifacts was not possible to pull from gitlab API

image: beshoynfarag/semgrep-juiceshop:latest

a costume docker image, uploaded on one of our team mates dockerhub. It includes the semgrep and the code for the juice shop ready for scanning.

semgrep --config p/ci --json -o semgrep-results.json juice-shop-master/ || true

semgrep: Runs the Semgrep tool.

--config p/ci: Tells Semgrep to use the predefined ruleset p/ci from the Semgrep Registry, optimized for CI pipelines (common security checks).

--json: Outputs the scan results in JSON format (machine-readable).

-o semgrep-results.json: Writes those results to a file called semgrep-results.json.

juice-shop-master/: Specifies the directory to scan — in this case, the source code of the Juice Shop app.

|| true: This ensures that even if Semgrep finds issues or returns a non-zero exit code, the job doesn't .

**Git-Leaks:**

gitleaks\_stage:

the job name and id in the pipeline

stage: sast\_stage:

the name of the stage that gitleaks is a part of

image: docker:24

services:

- docker:24-dind

The job uses docker image in the pipeline, uses docker image version 24

variables:

CI\_REGISTRY\_USER: beshoynfarag

CI\_REGISTRY\_PASSWORD: dckr\_pat\_H9qEvw8YgvWnAKPaAj6xTYCdUZk

DOCKER\_HOST: tcp://docker:2375

DOCKER\_TLS\_CERTDIR: ""

This declares variables that will be later on used for communication with the daemon and with dockerhub.

- docker pull zricethezav/gitleaks:latest

Pulls the latest git-leaks image from docker

- |docker run --rm -v "$CI\_PROJECT\_DIR":/src -w /src zricethezav/gitleaks:latestdetect --source=. --report-path=gitleaks-report.json || true

|| true at the end to make the job exists successfully no matter the scanning results

--report-path=gitleaks-report.json: Outputs results to a file.

artifacts: when: always

even if the job is canceled or failed always artifacts will be created.

expire\_in: 4 weeks

again for the same reason this one is stored for 4 weeks in case we don’t run the pipeline the artifacts does not expire, which will corrupt the .json and the vulnerability dashboard

**Dast:**

deploy:

stage: deploy

both the stage name and the job ID are called deploy

image: docker:latest

services:

- docker:dind

Uses the last docker image to run

Enables the docker to run in the pipeline

variables:

DOCKER\_HOST: tcp://docker:2375

DOCKER\_TLS\_CERTDIR: ""

CI\_REGISTRY: docker.io

CI\_REGISTRY\_USER: beshoynfarag

CI\_REGISTRY\_PASSWORD: dckr\_pat\_H9qEvw8YgvWnAKPaAj6xTYCdUZk

DOCKER\_HOST: tcp://docker:2375: Connects the Docker CLI to the DinD daemon over TCP (port 2375).

DOCKER\_TLS\_CERTDIR: "": Disables TLS for Docker (non-secure communication; okay in trusted environments).

CI\_REGISTRY: docker.io: Defines the Docker registry to use (Docker Hub).

CI\_REGISTRY\_USER and CI\_REGISTRY\_PASSWORD

script:

- echo "$CI\_REGISTRY\_PASSWORD" | docker login -u "$CI\_REGISTRY\_USER" --password-stdin docker.io

This line logs into dockerhub using the varibales declared before

- docker pull beshoynfarag/juice-shop:latest

Pulls the latest juiceshop image from dockerhub signed in to

- docker run -d --name juice-shop -p 3000:3000 beshoynfarag/juice-shop:latest

docker run  
This command starts a new container from a Docker image.

-d  
Runs the container in detached mode, meaning it runs in the background and doesn’t block the terminal (or pipeline job).This allows your CI job to continue without waiting for the container to stop.

--name juice-shop  
Assigns the container a friendly name: juice-shop. Naming containers makes it easier to manage them (e.g., stop or remove by name).

-p 3000:3000  
Maps port 3000 on the host (your CI runner or deployment server) to port 3000 inside the container. This means any traffic sent to the server’s port 3000 is forwarded to the container’s port 3000. Since Juice Shop runs on port 3000 inside the container, this exposes the app on port 3000 externally.

only:

- main

The job only executes when the changes are pushed to the main branch

dast\_juice\_shop\_stage:

stage: dast\_stage

allow\_failure: true

The job is named dast\_juice\_shop\_stage and runs in the dast\_stage of the pipeline.

allow\_failure: true means if this job fails or finds issues, it won't fail the entire pipeline (optional step).

image: docker:latest

services:

- docker:dind

Uses the Docker CLI image (docker:latest) to run Docker commands.

Runs Docker-in-Docker (docker:dind) as a service to launch containers inside the CI job.

variables:

DOCKER\_HOST: tcp://docker:2375

DOCKER\_TLS\_CERTDIR: ""

DOCKERHUB\_USERNAME: beshoynfarag

DOCKERHUB\_TOKEN: dckr\_pat\_H9qEvw8YgvWnAKPaAj6xTYCdUZk

DOCKER\_HOST and DOCKER\_TLS\_CERTDIR: Configure Docker CLI to connect to DinD without TLS.

DOCKERHUB\_USERNAME and DOCKERHUB\_TOKEN: Docker Hub credentials to pull images.

script:

- echo "$DOCKERHUB\_TOKEN" | docker login -u "$DOCKERHUB\_USERNAME" --password-stdin

Logs into dockerhub

- docker pull beshoynfarag/zap2docker-stable:latest

Pulls the latest OWASP ZAP Docker image from your Docker Hub repo (zap2docker-stable is a popular image for OWASP ZAP).

- |

docker run -v $(pwd):/zap/wrk/:rw beshoynfarag/zap2docker-stable \

zap-baseline.py -t http://10.97.7.109:3000 \

-r juice-dast-report.html \

-J juice-dast-report.json || true

-v $(pwd):/zap/wrk/:rw: Mounts the current directory into the container at /zap/wrk/ with read-write permission, so reports can be saved to the pipeline workspace.

Runs zap-baseline.py, a script that launches a baseline scan.

-t http://10.97.7.109:3000: The target URL for scanning — your Juice Shop app running at that IP and port.

-r juice-dast-report.html: Generates an HTML report.

-J juice-dast-report.json: Generates a JSON report.

|| true: Ensures the job doesn’t fail even if ZAP detects issues.

artifacts:

when: always

paths:

- juice-dast-report.html

- juice-dast-report.json

expire\_in: 3 weeks

it always generated artifacts even if the job is cancelled or failed

the artifacts expires in 3 week in case we don’t run the pipeline before the presentation.

The artifacts are generated in 2 forms the first is .json for the vulnerability dashboard and the other is .html which is a UI for dast-zap owasp

**Todoapp**

**Git-Leaks:**

The only difference is that it is allowed to fail.

The artifacts expiry length is not specified

**Dast:**

The difference is that it does not generate .json since we have no use of it in our vulnerability dashboard

The target port is 8000 instead of 3000

The job name called just dast-stage

**Deploy:**

This job uses host:container of 80:8000 instead of 3000:3000

The name of the container and the pulled image is todoapp

**SonarQube:**

sonar:

stage: sast\_stage

the job id is called sonar after the method chosen for it

it’s part of the stage called sast\_stage the same as gtileaks

image: maven:3.9.0-eclipse-temurin-17

maven is a tool special environment that is represented in an image form including the version 3.9.0 and the JDK 17

script:

- "curl --fail http://10.97.7.109:9000 || exit 1"

This line makes sure that sonarqube is set and ready to analyze my code for vulnerabilities

If sonarqube is not ready it exits with code 1

- cd ./todolist

Navigates to the todolist file where the java code is

- "mvn verify sonar:sonar -Dsonar.projectKey=todoapp-key -Dsonar.projectName=todoapp -Dsonar.projectVersion=1.0 -Dsonar.sources=src/main/java -Dsonar.tests=src/test/java -Dsonar.host.url=http://10.97.7.109:9000 -Dsonar.login=squ\_4853d36dd8d2fe229ba46e6fb9de41f6cc463495"

Now, it tells Maven to:

Check and build your code (verify).

Send the code to the Sonar server for analysis (sonar:sonar).

It also tells the server:

What your project is called (todoapp-key).

Where the code and tests live (src/main/java and src/test/java).

Where the Sonar server is.

How to log in securely (with a secret token).

- "curl -s -u squ\_4853d36dd8d2fe229ba46e6fb9de41f6cc463495: \"http://10.97.7.109:9000/api/issues/search?componentKeys=todoapp-key\" -o target/sonar-issues.json"

After the scanning is done this line requests the results from our sonarqube

artifacts:

paths:

- "todolist/target/sonar-issues.json"

expire\_in: 2 hrs

the artifacts are generated after the job is done and stored as .json as well as it can be found on sonarqube UI the expiry for the .json is 2 hours

unlike our juice shop this one will not generate artifacts if cancelled or failed

for the UI just read the technical report the HMI phase

for the documentation just read your parts