Practical_Notes_DevSecOps

Here are some notes and commands used during the workshop.

Created with ❤️ by **NAJIM Ayoub** for **ENSA KHOURIBGA** students.

SCA — Analyse des dépendances

First we clone the repo of our vulnerable project. “WebGoat”

```
git clone https://github.com/WebGoat/WebGoat.git
```

Let’s download the OwaspDependencyCheck using the wget command:

```
wget https://github.com/jeremylong/DependencyCheck/releases/download/v7.4.4/dependency
-check-7.4.4-release.zip
```

After unziping the “dependency-check-7.4.4-release.zip”, we scan our repo WebGoat using the command below :

```
cd dependensy-check-7.4.4/bin
./dependency-check.sh --scan ~/WebGoat/ --format JSON --out ~/WebGoat/report_owasp_dep
endency_check.json
```

We use the following command to show the file content :

```
cat ~/WebGoat/report_owasp_dependency_check.json | jq .
```

SAST — Secret Scan

First we clone the scanner repo. “repo-supervisor”

```
git clone https://github.com/auth0/repo-supervisor.git
```

Because the tool needs nodejs and npm as requirements, we install npm and nodejs as below :

```
sudo apt update
sudo apt install nodejs@14
sudo apt install npm
```

Then we run the commands :

```
cd repo-supervisor
npm ci && npm run build
JSON_OUTPUT=1 node ./dist/cli.js ~/WebGoat/ >> ~/WebGoat/repo-supervisor_output.json
```

DAST

We use Dastardly as below to test our target : http://testphp.vulnweb.com

```
docker run --user $(id -u) --rm -v $(pwd):/dastardly -e \
DASTARDLY_TARGET_URL=http://testphp.vulnweb.com/ -e \
DASTARDLY_OUTPUT_FILE=/dastardly/dastardly-report.xml \
public.ecr.aws/portswigger/dastardly:latest
```

Same for OWASP ZAP.

```
docker run --user $(id -u):$(id -g) -w /zap -v $(pwd):/zap/wrk:rw --rm owasp/zap2docke
r-stable:2.10.0 zap-baseline.py -t http://testphp.vulnweb.com/ -d -x zap-output.xml
```

CI/CD Pipeline

```
---
image: docker:latest # To run all jobs in this pipeline, use a latest docker image

services:
  - docker:dind

stages:
  - build
  - test
  - release
  - deploy
  - operate
  - prod

sca:
  stage: build
  script:
    # We are going to pull the owasp/dependency-check image
    - docker pull owasp/dependency-check
    # Let's run the scan
    - docker run --rm -v $(pwd):/src owasp/dependency-check --scan /src --format JSON
 --out owasp_dependency_check.json
  artifacts:
    paths: [owasp_dependency_check.json]
    when: always
  allow_failure: true

sast-secrets-scanning:
  script:
    - docker run -it --rm -v $(pwd):/opt/scan_me repo-supervisor /bin/bash -c "source
 ~/.bashrc && JSON_OUTPUT=1 node /opt/repo-supervisor/dist/cli.js /opt/scan_me" >> rep
o-supervisor_output.json
  artifacts:
    paths: [repo-supervisor_output.json]
    when: always
  allow_failure: true

dast-dastardly:
  stage: test
  script:
    - docker run --user $(id -u) --rm -v $(pwd):/dastardly -e
      DASTARDLY_TARGET_URL=http://testphp.vulnweb.com/ -e
      DASTARDLY_OUTPUT_FILE=/dastardly/dastardly-report.xml
      public.ecr.aws/portswigger/dastardly:latest
  artifacts:
    paths: [dastardly-report.xml]
    when: always
  allow_failure: true

dast-owasp-zap:
  stage: test
  #before_script:
  #  - apk add py-pip py-requests
  script:
    - docker run --user $(id -u):$(id -g) -w /zap -v $(pwd):/zap/wrk:rw --rm owasp/zap
2docker-stable:2.10.0 zap-baseline.py -t http://testphp.vulnweb.com/ -d -x zap-output.
xml
  #after_script:
  #  - python3 upload-results.py --host $DOJO_HOST --api_key $DOJO_API_TOKEN --engagem
ent_id 3 --product_id 1 --username "admin" --result_file zap-output.xml --scanner "ZAP
Scan"
  artifacts:
    paths: [zap-output.xml]
    when: always
  allow_failure: true
```