If you want to run the same process manually without relying on Jenkins, here are the steps to manually execute the OWASP ZAP container and perform the scans using the ghcr.io/zaproxy/zaproxy:latest image.

**Manual Steps:**

1. **Pull the ZAP Docker Image**: First, you need to pull the latest ZAP Docker image from the GitHub Container Registry:

docker pull ghcr.io/zaproxy/zaproxy:latest

If you prefer to run the container without the UI (headless mode), you can run the container with the /bin/bash command:

docker run -dt --name zaproxy ghcr.io/zaproxy/zaproxy:latest /bin/bash

1. **Create a Working Directory** (Optional): It’s a good practice to create a directory for saving reports. You can do it inside the running container:

docker exec zaproxy mkdir -p /zap/wrk

1. **Run the ZAP Scan**: Depending on the scan type (Baseline, APIS, Full), you will run the respective ZAP scan script.
   * **Baseline Scan**:

docker exec zaproxy zap-baseline.py -t <https://www.openwriteup.com/> -r /zap/wrk/report.html

* + **API Scan**:

docker exec zaproxy zap-api-scan.py -t http://example.com/api -r /zap/wrk/report.html

* + **Full Scan**:

docker exec zaproxy zap-full-scan.py -t https://www.openwriteup.com/ -r /zap/wrk/report.html

Replace http://example.com with the URL of the web application you want to scan.

1. **Retrieve the Report**: After the scan completes, you can retrieve the generated report from the container by copying it to your local machine:
2. docker cp zaproxy:/zap/wrk/report.html ./zap\_report.html
3. **Stop and Remove the ZAP Container**: Once you are done with the scan, you can stop and remove the ZAP container to clean up:
4. docker stop zaproxy
5. docker rm zaproxy

**Additional Notes:**

* **Accessing ZAP UI**: If you started ZAP with the UI (-p 8080:8080), you can access the ZAP dashboard by going to http://localhost:8080 in your browser.
* **Saving Reports**: You can specify any directory on your local machine to store the reports. By default, the report will be saved in the /zap/wrk/ directory inside the container.
* **Scan Types**: Choose between Baseline, APIS, and Full based on your needs. Each type corresponds to different scan scripts, with Full being the most comprehensive and time-consuming.

Let me know if you need any more details on this process!

pipeline {

agent any

parameters {

choice(name: 'SCAN\_TYPE', choices: ['Baseline', 'APIS', 'Full'], description: 'Type of scan that will be performed.')

string(name: 'TARGET', defaultValue: 'http://example.com', description: 'Target URL to scan')

booleanParam(name: 'GENERATE\_REPORT', defaultValue: true, description: 'Whether to generate a scan report')

}

stages {

stage('Pull ZAP Docker Image') {

steps {

script {

echo 'Pulling ZAP Docker image from GitHub Container Registry...'

sh 'docker pull ghcr.io/zaproxy/zaproxy:latest'

}

}

}

stage('Run ZAP Container') {

steps {

script {

echo 'Running ZAP container...'

sh 'docker run -dt --name zaproxy ghcr.io/zaproxy/zaproxy:latest /bin/bash'

}

}

}

stage('Scan Target with ZAP') {

steps {

script {

def scan\_type = "${params.SCAN\_TYPE}"

def target = "${params.TARGET}"

if (scan\_type == 'Baseline') {

echo 'Running Baseline scan...'

sh """

docker exec zaproxy mkdir -p /zap/wrk

docker exec zaproxy zap-baseline.py -t $target -r /zap/wrk/report.html

"""

} else if (scan\_type == 'APIS') {

echo 'Running API scan...'

sh """

docker exec zaproxy zap-api-scan.py -t $target -r /zap/wrk/report.html

"""

} else if (scan\_type == 'Full') {

echo 'Running Full scan...'

sh """

docker exec zaproxy zap-full-scan.py -t $target -r /zap/wrk/report.html

"""

}

}

}

}

stage('Generate Report') {

when {

expression { return params.GENERATE\_REPORT }

}

steps {

script {

echo 'Copying report from ZAP container...'

sh 'docker cp zaproxy:/zap/wrk/report.html ./zap\_report.html'

archiveArtifacts artifacts: 'zap\_report.html', allowEmptyArchive: true

}

}

}

}

post {

always {

echo 'Cleaning up ZAP container...'

sh 'docker rm -f zaproxy'

}

success {

echo 'Scan completed successfully!'

}

failure {

echo 'Scan failed!'

}

}

}