**OWASP Workflow**

**Overview:**

This GitHub Actions workflow automates the process of performing OWASP ZAP (Zed Attack Proxy) security scans on both the frontend and backend applications as part of the CI/CD pipeline. The purpose is to identify potential security vulnerabilities in both parts of the application and generate reports for further analysis.

**Key Concepts:**

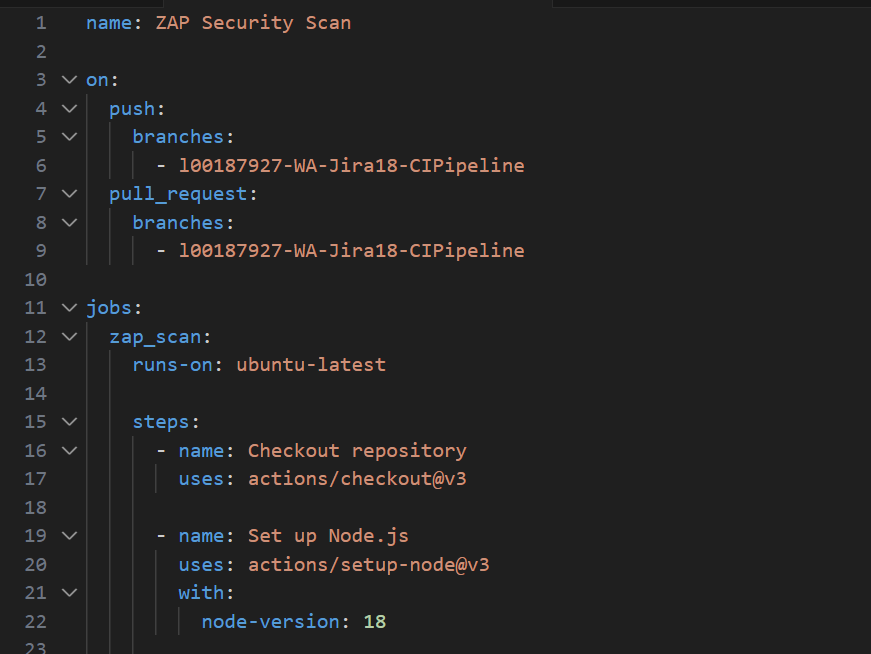
The key concept of this workflow is to automate security testing using OWASP ZAP in a CI/CD pipeline to scan both the frontend and backend applications for vulnerabilities. It installs dependencies, starts the applications, and runs a baseline security scan to identify potential security issues. The results are then uploaded as artifacts in multiple formats (JSON, Markdown, HTML) for review and analysis.

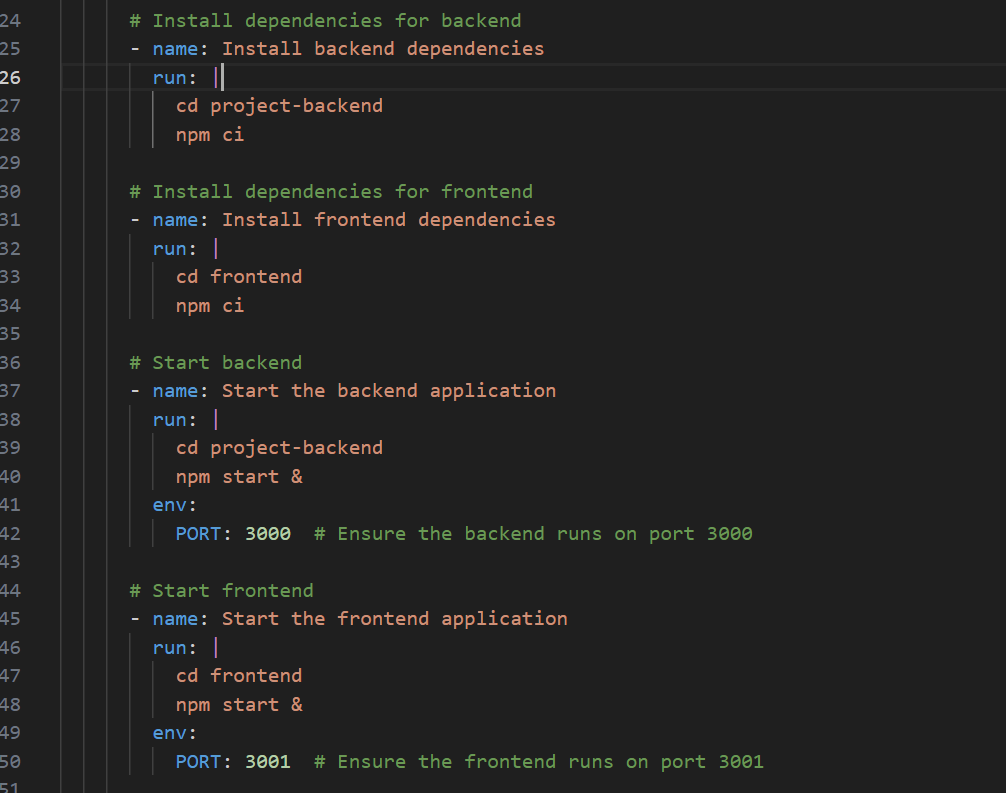
**Why OWASP ZAP:**

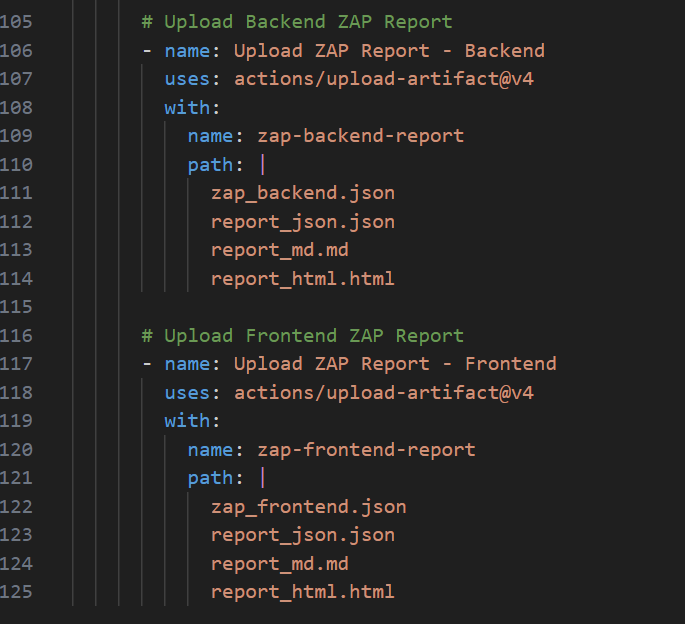
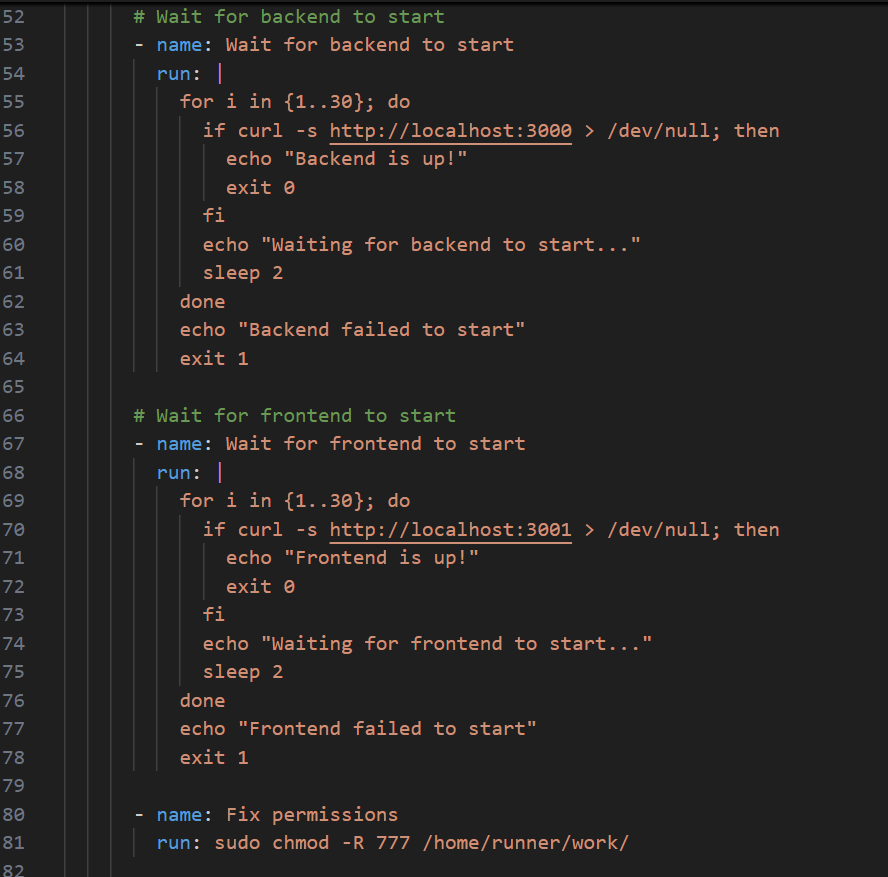
1. Automated Security Testing: OWASP ZAP provides automated scanning to identify common vulnerabilities like SQL injection, XSS, and misconfigurations.
2. Open Source: It's free, open-source, and widely trusted in the security community, making it accessible for all types of projects.
3. CI/CD Integration: Integrating ZAP into the CI/CD pipeline helps catch vulnerabilities early in development, ensuring proactive security and reducing deployment risks.

**Workflow:**

Location: .github/workflows/ZAP.yml







1. **Workflow Metadata**

name: ZAP Security Scan

* The workflow is named "ZAP Security Scan" to indicate its purpose.

1. **Trigger Events**

on:

push:

branches:

- l00187927-WA-Jira18-CIPipeline

pull\_request:

branches:

- l00187927-WA-Jira18-CIPipeline

The workflow is triggered when:

* A push is made to the l00187927-WA-Jira18-CIPipeline branch.
* A pull request is opened for the same branch.

1. **Jobs Section**

jobs:

zap\_scan:

runs-on: ubuntu-latest

* The job zap\_scan runs on Ubuntu as the operating system.

1. **Steps Section**

The workflow executes several steps in sequence.

Step 1: Checkout Repository

- name: Checkout repository

uses: actions/checkout@v3

* This pulls the latest code from the repository into the GitHub Actions runner.

Step 2: Set up Node.js

- name: Set up Node.js

uses: actions/setup-node@v3

with:

node-version: 18

* Installs Node.js v18, which is required to run the backend and frontend applications.

Step 3: Install Dependencies

- name: Install backend dependencies

run: |

cd project-backend

npm ci

* Installs the backend dependencies using npm ci (clean install).

- name: Install frontend dependencies

run: |

cd frontend

npm ci

* Installs the frontend dependencies.

Step 4: Start Applications

- name: Start the backend application

run: |

cd project-backend

npm start &

env:

PORT: 3000

* Starts the backend service in the background (&) on port 3000.

- name: Start the frontend application

run: |

cd frontend

npm start &

env:

PORT: 3001

* Starts the frontend service in the background (&) on port 3001.

Step 5: Wait for Applications to Start

- name: Wait for backend to start

run: |

for i in {1..30}; do

if curl -s http://localhost:3000 > /dev/null; then

echo "Backend is up!"

exit 0

fi

echo "Waiting for backend to start..."

sleep 2

done

echo "Backend failed to start"

exit 1

* This step checks if the backend is running by making a curl request to http://localhost:3000 in a loop for 60 seconds (30 attempts, 2 seconds apart).

- name: Wait for frontend to start

run: |

for i in {1..30}; do

if curl -s http://localhost:3001 > /dev/null; then

echo "Frontend is up!"

exit 0

fi

echo "Waiting for frontend to start..."

sleep 2

done

echo "Frontend failed to start"

exit 1

* Similar to the backend, this ensures the frontend is running on port 3001.

Step 6: Fix Permissions

- name: Fix permissions

run: sudo chmod -R 777 /home/runner/work/

* Sets full permissions (777) to avoid permission errors in subsequent steps.

1. **Run ZAP Security Scan**

Step 7: ZAP Scan for Backend

- name: Run ZAP Baseline Scan - Backend

uses: zaproxy/action-baseline@v0.7.0

with:

target: 'http://localhost:3000'

docker\_name: 'ghcr.io/zaproxy/zaproxy:stable'

cmd\_options: '-T 120 -J zap\_backend.json'

fail\_action: false

* Runs OWASP ZAP Baseline Scan on the backend (http://localhost:3000).
* Uses the ZAP Docker image (zaproxy:stable).
* Sets a timeout of 120 seconds (-T 120).
* Outputs results as zap\_backend.json.
* fail\_action: false ensures that the scan does not fail the workflow.

Step 8: ZAP Scan for Frontend

- name: Run ZAP Baseline Scan - Frontend

uses: zaproxy/action-baseline@v0.7.0

with:

target: 'http://localhost:3001'

docker\_name: 'ghcr.io/zaproxy/zaproxy:stable'

cmd\_options: '-T 120 -J zap\_frontend.json'

fail\_action: false

* Scans the frontend application for security vulnerabilities.
* Generates a JSON report (zap\_frontend.json).
* Ensures security testing is automated within the CI/CD pipeline.

1. **Debug - List Generated Files**

- name: Debug - List generated files

run: ls -la

* Lists all files to verify the reports exist.

1. **Upload Security Reports**

For backend

- name: Upload ZAP Report - Backend

uses: actions/upload-artifact@v4

with:

name: zap-backend-report

path: |

zap\_backend.json

report\_json.json

report\_md.md

report\_html.html

* Uploads the scan reports so they can be reviewed later.
* Artifacts are stored in GitHub for easy access and download.
* The scan reports give insights into vulnerabilities detected in the backend application.

For Frontend:

- name: Upload ZAP Report - Frontend

uses: actions/upload-artifact@v4

with:

name: zap-frontend-report

path: |

zap\_frontend.json

report\_json.json

report\_md.md

report\_html.html

* Uploads ZAP Reports: This step uploads the ZAP scan reports for the frontend application as artifacts.
* Artifact Naming: The uploaded artifact is named zap-frontend-report for easy identification.
* Files Included: It uploads multiple report files: zap\_frontend.json, report\_json.json, report\_md.md, and report\_html.html for different formats of the scan results (JSON, Markdown, and HTML).