**CAPSTONE PROJECT REPORT**

**TITLE:**

**Securing the Application Lifecycle**

<https://github.com/ScaleSec/vulnado.git>

**By:**

**Gowtham Attili**

**Mugundhan R**

**Securing the Application Lifecycle**

**Problem Statement:** Security breaches have been on the rise, and "TechSolutions" doesn't want to take any chances. Your task is to review their application for vulnerabilities, secure the development lifecycle, and implement security best practices in their DevOps processes.

**Dataset:** A sample web application with known vulnerabilities, available on GitHub. This simulates a typical business application with potential security loopholes.

<https://github.com/ScaleSec/vulnado.git>

**Project Steps:**

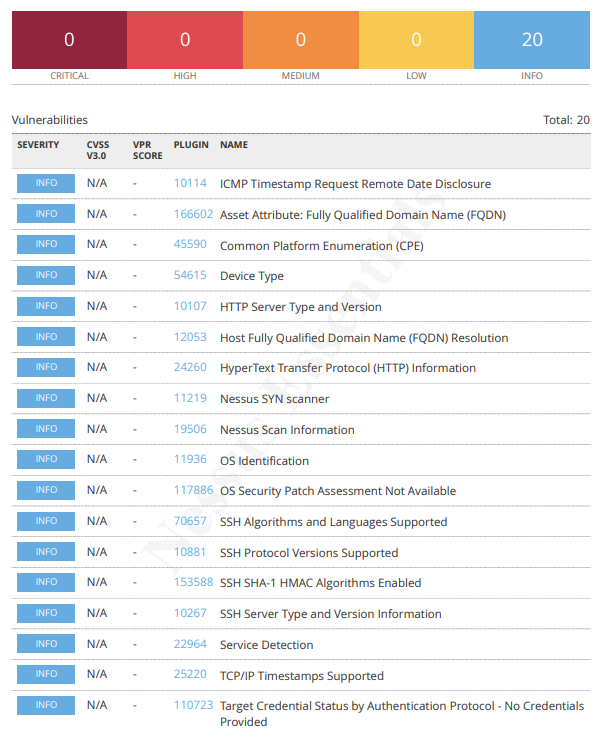
**STEP-1: Application Vulnerability Assessment**

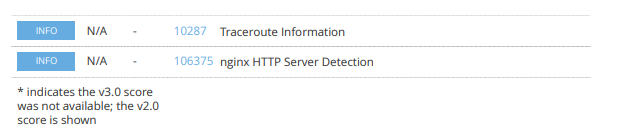
* Clone the vulnerable web application from github.
* Run the web application in local
* Download and install OWASP ZAP tool to check vulnerabilities of the application (<https://www.zaproxy.org/download/>) .
* Add the foxy proxy extension to the browser to identify the vulnerabilities of the application.
* Give the url where our application is running in the browser (<https://localhost:1337>).
* Do automated and manual scans for the application.
* It’ll automatically generates reports,alerts and vulnerabilities.

For better understanding on the vulnerabilities and how to secure the application we did scanning to our application using Nessus tool and got report of vulnerabilities.



**Fig 1: Vulnerability report by Nessus Tool**

****

****

**Fig 2: These are the vulnerabilities identified by Tenable Nessus Tool.**

**STEP-2: Container Security:**

* The Application has client as frontend , vulnado, internal-site as middleware and postgres database as backend.
* So we dockerized the application into four dockerfiles by using docker best practices.
* These are the dockerfiles created for Vulnado Application

Dockerfile for client:

FROM nginx:alpine

COPY . /usr/share/nginx/html

Dockerfile for vulnado:

FROM openjdk:8

RUN apt-get update && \

apt-get install build-essential maven default-jdk cowsay netcat -y && \

update-alternatives --config javac

COPY . .

CMD ["mvn", "spring-boot:run"]

Dockerfile for internal-site:

FROM nginx:alpine

COPY . /usr/share/nginx/html

Docker-compose file for vulnado application:

version: "3"

services:

vulnado:

build: .

ports:

- 8081:8081

links:

- db

- internal\_site

environment:

- PGPASSWORD=vulnado

- PGDATABASE=vulnado

- PGHOST=db:5432

- PGUSER=postgres

depends\_on:

- "db"

client:

build: client

ports:

- 1337:80

db:

image: postgres

environment:

- POSTGRES\_PASSWORD=vulnado

- POSTGRES\_DB=vulnado

internal\_site:

build: internal\_site

**STEP-3: DevSecOps Implementation:**

* Integrated OWASP Dependency check with the CI/CD pipeline to detect security vulnerabilities in dependencies.
* Integrated Snyk with the CI/CD pipeline to detect security vulnerabilities in dependencies and in our application.
* These two will generate vulnerability reports.

Steps to add owasp dependency check in jenkins pipeline :

1. Install OWASP Dependency Check plugin in the plugins section in manage jenkins.
2. Add the Owasp tool in the global tool configuration.
3. Add owasp dependency stage in the jenkins pipeline.

Owasp Dependency Check stage to be added in Jenkins Pipeline:

stage('OWASP Dependency-Check Vulnerabilities') {

steps {

dependencyCheck additionalArguments: '''

-o './'

-s './'

-f 'ALL'

--prettyPrint''', odcInstallation: 'owasp-zap'

dependencyCheckPublisher pattern: 'dependency-check-report.xml'

}

}

Steps to add Snyk security check in jenkins pipeline :

1. Install Snyk security plugin in the plugins section in manage jenkins.
2. Add the Snyk security scan in the global tool configuration.
3. Create an account in Snyk and generate API token and copy the token.
4. Go to the credential manager in jenkins and add the credentials as Snyk api token and give the credential id.
5. Add Snyk security scan stage in the jenkins pipeline using pipeline script generator.

Snyk security scan stage to be added in Jenkins Pipeline:

stage('Snyk Security') {

steps {

snykSecurity failOnError: false, failOnIssues: false, organisation: 'attiligowtham', projectName: 'vulnado', snykInstallation: 'snyk', snykTokenId: 'snyk-id', targetFile: 'pom.xml'

}

}

EKS Cluster Setup:

Create an EKS cluster in the same instance where you set up jenkins.

Following are the steps to create EKS Cluster:

* Install Kubectl

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin

kubectl version --client

* Install eksctl

curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

* Install AWS CLI

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo apt install unzip

sudo unzip awscliv2.zip

sudo ./aws/install

* Create IAM Role with following policies:

1. EC2 full access
2. VPC full access
3. IAM full access
4. Cloudformation access
5. Administrator Access
6. Attach it to the instance (Ec2 dashboard --> instance --> actions -->security --> modify IAM role --> select this role)

* Create EKS cluster using following command

eksctl create cluster --name cluster\_name \

--region aws\_region \

--node-type instance\_type \

--nodes-min 2 \

--nodes-max 2 \

--zones availability-zone1,availability-zone2

* Check whether the cluster is created or not using the following command

kubectl get nodes -o wide

Istio Setup in EKS Cluster:

Download Istio using the following documentation given below. <https://istio.io/latest/docs/setup/getting-started/>

Create and apply manifest files for all the services in our application in eks cluster by following commands:

* Go inside the istio folder

cd istio-1.19.0

* Create Istio ingress gateway to route traffic to our application.

vi gateway.yml

apiVersion: networking.istio.io/v1alpha3

kind: Gateway

metadata:

name: java-app

spec:

selector:

istio: ingressgateway # use istio default controller

servers:

- port:

number: 80

name: http

protocol: HTTP

hosts:

- "\*"

* Create Virtual Services to route traffic to ours destination i.e., client (frontend).

vi vs.yml

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: java-app-vs

spec:

hosts:

- "\*" # This will match any host

gateways:

- java-app # Reference to the Gateway created above

http:

- route:

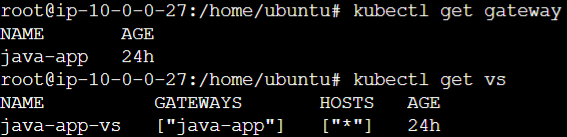
- destination:

host: client-service # The name of your service in Kubernetes

port:

number: 8080 # The port your service is listening on

* Check whether the gateway and virtual services are created or not



* Create manifest files for client,vulnado,internal-site and db.

Client.yml

vi client.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: client-deployment

spec:

replicas: 1

selector:

matchLabels:

app: client

template:

metadata:

labels:

app: client

spec:

containers:

- name: client

image: gowtham47/client:lastest

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: client-service

spec:

selector:

app: client

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

Vulnado.yml

vi vulnado.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: vulnado-deployment

spec:

replicas: 1

selector:

matchLabels:

app: vulnado

template:

metadata:

labels:

app: vulnado

spec:

containers:

- name: vulnado

image: gowtham47/vulnado

ports:

- containerPort: 8081

env:

- name: PGPASSWORD

value: vulnado

- name: PGDATABASE

value: vulnado

- name: PGHOST

value: db:5432

- name: PGUSER

value: postgres

Internal-site.yml

vi internal-site.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: internal-site-deployment

spec:

replicas: 1

selector:

matchLabels:

app: internal-site

template:

metadata:

labels:

app: internal-site

spec:

containers:

- name: internal-site

image: gowtham47/internal-site

Db.yml

vi db.yml

apiVersion: apps/v1

kind: Deployment

metadata:

name: db-deployment

spec:

replicas: 1

selector:

matchLabels:

app: db

template:

metadata:

labels:

app: db

spec:

containers:

- name: db

image: gowtham47/postgres

env:

- name: POSTGRES\_PASSWORD

value: vulnado

- name: POSTGRES\_DB

value: vulnado

* Apply all the manifest files using following commands

kubectl apply -f client.yml

kubectl apply -f vulnado.yml

kubectl apply -f internal-site.yml

kubectl apply -f db.yml

* Check whether the deployments are up and running

root@ip-10-0-0-27:/home/ubuntu# kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

client-deployment 1/1 1 1 24h

db-deployment 1/1 1 1 24h

internal-site-deployment 1/1 1 1 24h

vulnado-deployment 1/1 1 1 24h

Now Integrate the EKS Cluster with the Jenkins pipeline to automate the complete CI/CD pipeline.

**COMPLETE CI/CD PIPELINE FOR SECURING THE APPLICATION LIFE CYCLE.**

**pipeline {**

**agent any**

**tools{**

**jdk 'jdk'**

**maven 'maven'**

**}**

**environment {**

**aws\_region = "us-west-1" //Your aws region**

**eks\_cluster = "myeks" // Your cluster**

**}**

**stages {**

**stage('checkout') {**

**steps {**

**git branch: 'master', url: 'https://github.com/Gowtham-745/vulnado.git' //update your repo**

**}**

**}**

**stage('OWASP Dependency-Check') {**

**steps {**

**dependencyCheck additionalArguments: '''**

**-o './'**

**-s './'**

**-f 'ALL'**

**--prettyPrint''', odcInstallation: 'owasp'**

**dependencyCheckPublisher pattern: 'dependency-check-report.xml'**

**}**

**}**

**stage('Clean') {**

**steps {**

**// Get some code from a GitHub repository**

**git 'https://github.com/Gowtham-745/vulnado.git'**

**// To run Maven on a Windows agent, use**

**sh "mvn clean"**

**}**

**post {**

**// If Maven was able to run the tests, even if some of the test**

**// failed, record the test results and archive the jar file.**

**success {**

**echo 'Cleaning Project is Done'**

**}**

**}**

**}**

**stage('Compile') {**

**steps {**

**// Get some code from a GitHub repository**

**git 'https://github.com/Gowtham-745/vulnado.git'**

**// To run Maven on a Windows agent, use**

**sh "mvn compile"**

**}**

**post {**

**// If Maven was able to run the tests, even if some of the test**

**// failed, record the test results and archive the jar file.**

**success {**

**echo 'Compiling Project is Done'**

**}**

**}**

**}**

**stage('Snyk Security') {**

**steps {**

**snykSecurity failOnError: false, failOnIssues: false, organisation: 'attiligowtham', projectName: 'vulnado', snykInstallation: 'snyk', snykTokenId: 'snyk-id', targetFile: 'pom.xml'**

**//bat 'C:\\Users\\attil\\AppData\\Roaming\\npm\\node\_modules\\snyk\\wrapper\_dist\\snyk-win.exe test'**

**}**

**}**

**stage('Package') {**

**steps {**

**// Get some code from a GitHub repository**

**git 'https://github.com/Gowtham-745/vulnado.git'**

**// To run Maven on a Windows agent, use**

**sh "mvn -Dmaven.test.failure.ignore=true clean package"**

**}**

**post {**

**// If Maven was able to run the tests, even if some of the test**

**// failed, record the test results and archive the jar file.**

**success {**

**archiveArtifacts 'target/\*.jar'**

**}**

**}**

**}**

**stage("Docker Build & Push"){**

**steps{**

**script{**

**withDockerRegistry(credentialsId: 'docker-hub', toolName: 'docker') {**

**sh "docker tag vulnado\_vulnado gowtham47/vuln:latest "**

**sh "docker push gowtham47/vuln:latest "**

**sh "docker tag vulnado\_client gowtham47/clin:latest "**

**sh "docker push gowtham47/clin:latest "**

**sh "docker tag vulnado\_internal\_site gowtham47/ins:latest "**

**sh "docker push gowtham47/ins:latest "**

**sh "docker tag postgres gowtham47/d-b:latest "**

**sh "docker push gowtham47/d-b:latest "**

**}**

**}**

**}**

**}**

**stage('deploy') {**

**steps {**

**script {**

**sh "aws eks --region $aws\_region update-kubeconfig --name $eks\_cluster"**

**sh "kubectl get svc"**

**}**

**}**

**}**

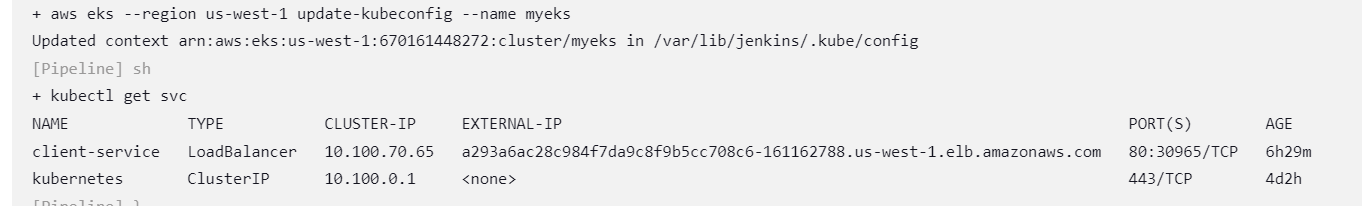
**}**

**}**

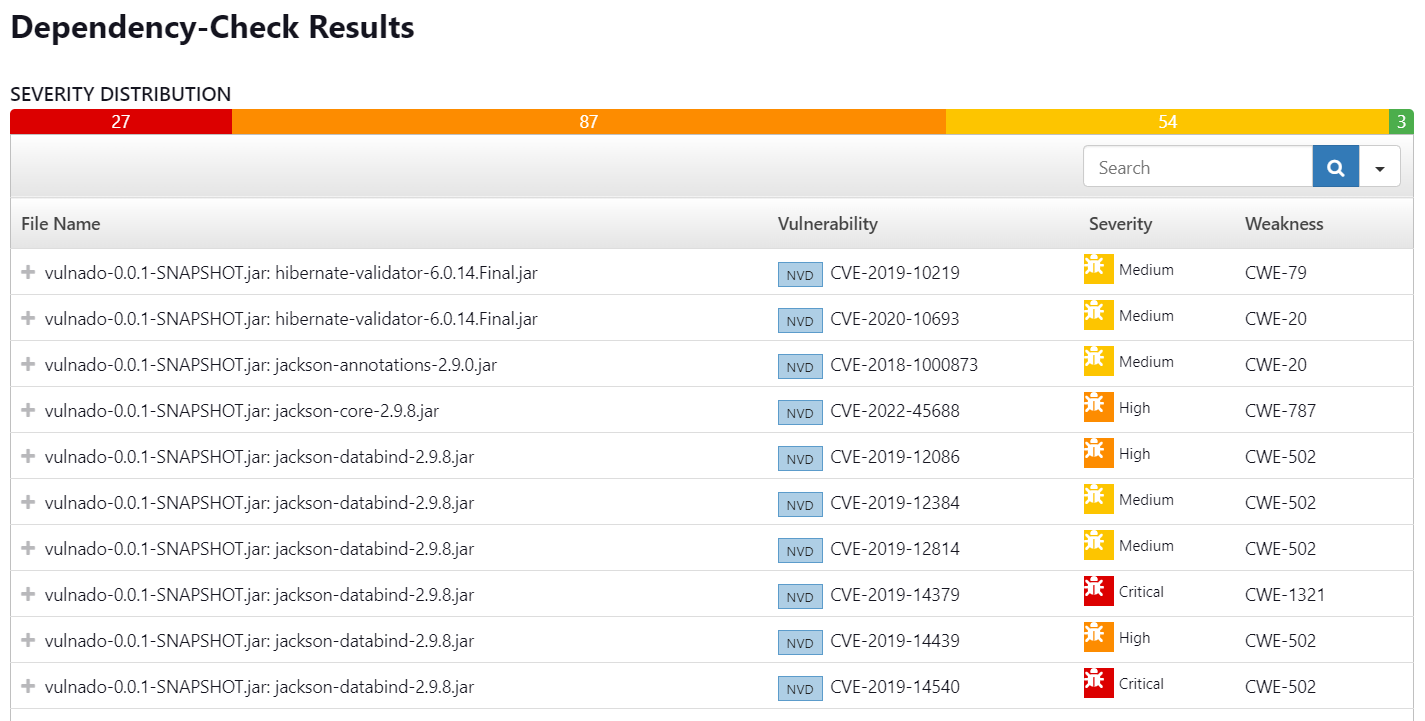
Build the pipeline to build, check for owasp dependency check and snyk security scan and to push the images to docker hub and deploy the application in the EKS cluster.

-> To access the application through the browser use the link from the output of the pipeline from the deployment stage.

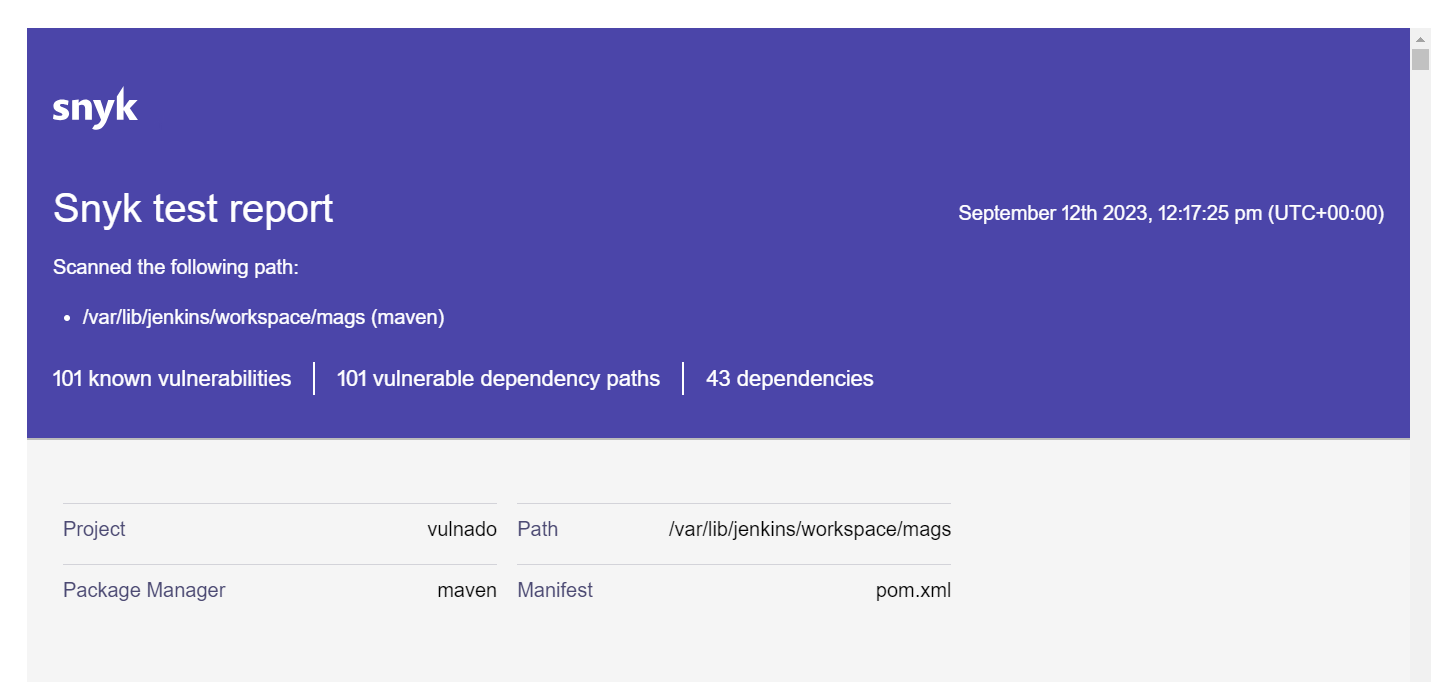
Console Output to access the application:



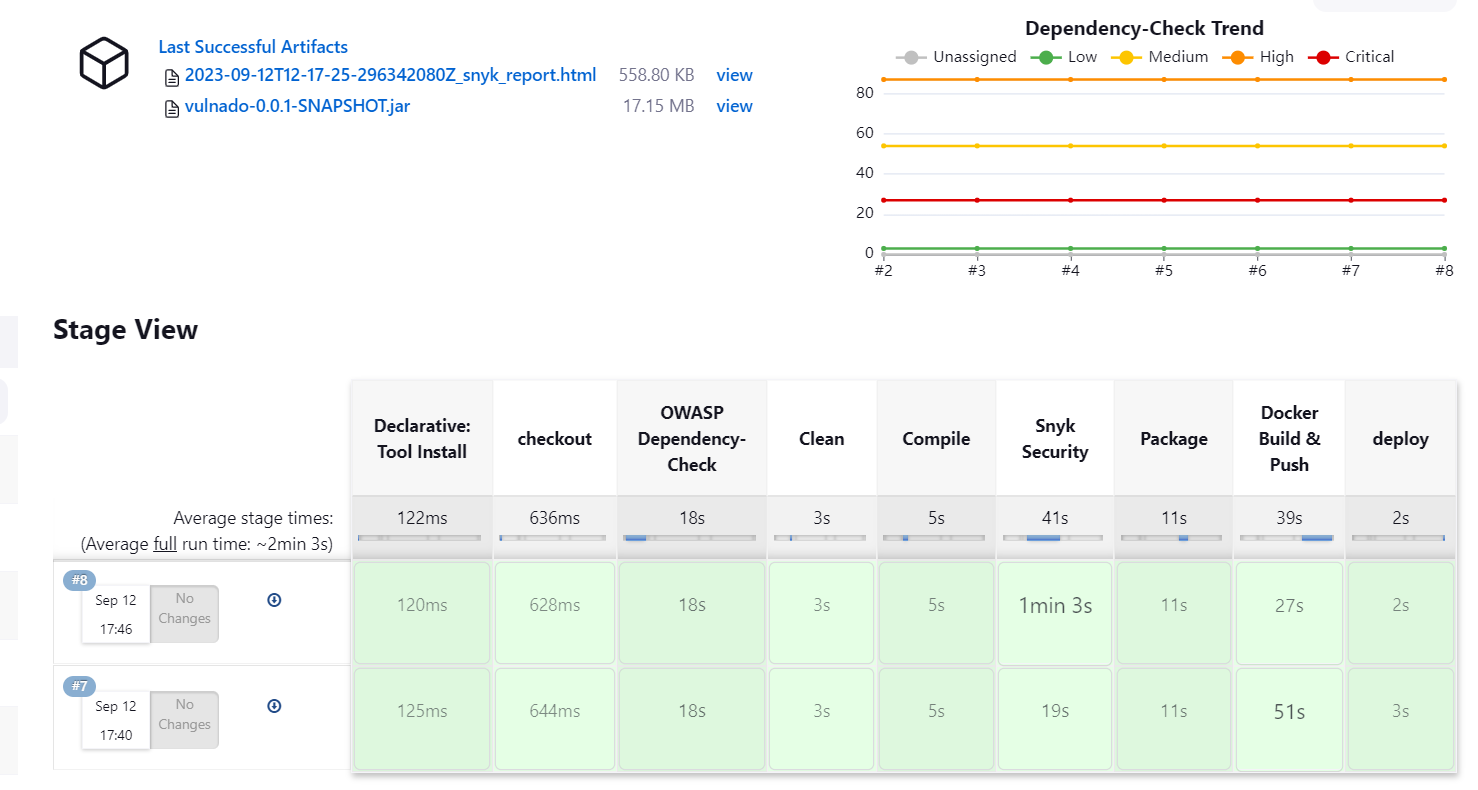
Console Output of Owasp Dependency Check Phase:



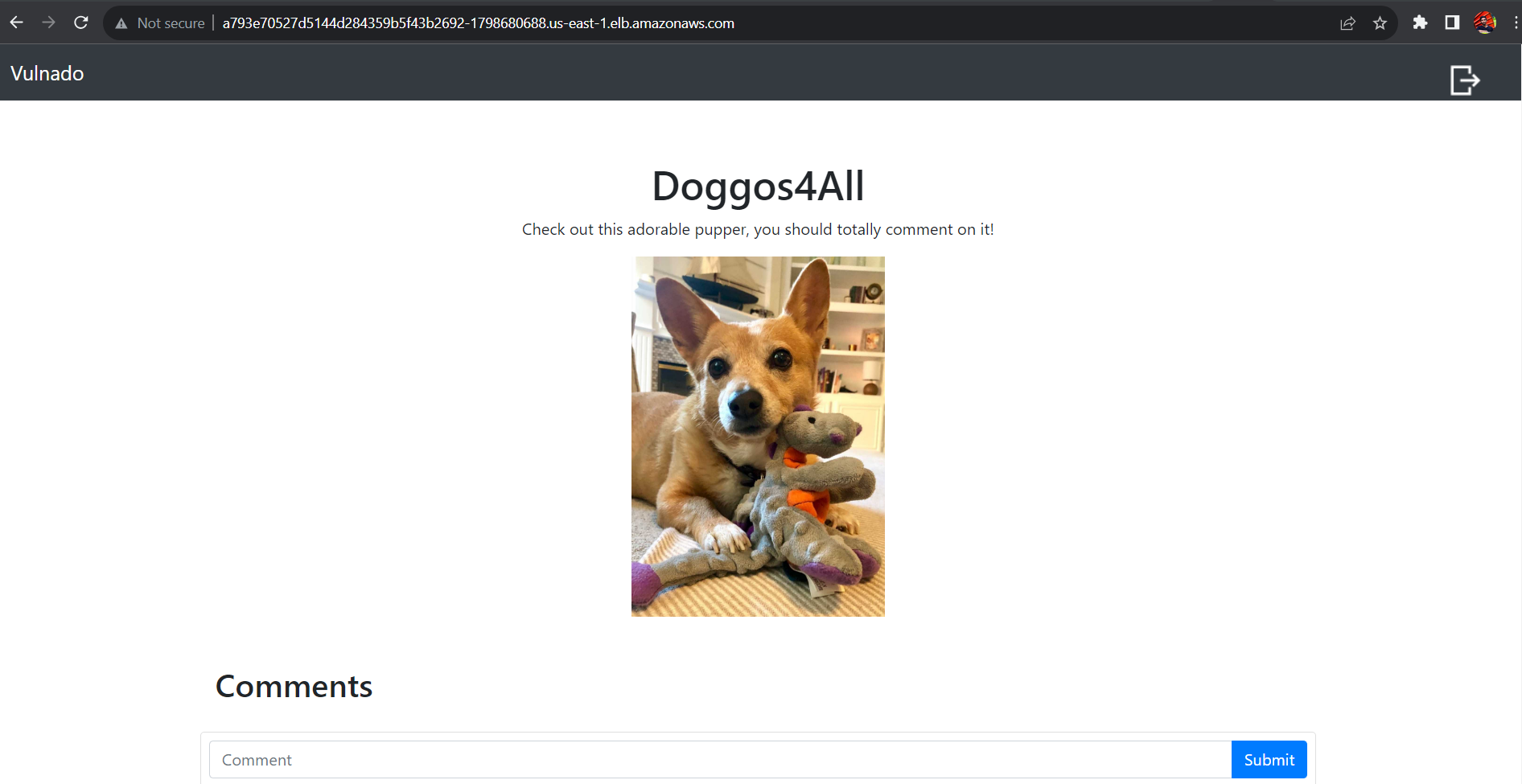
Console Output of Snyk security stage:



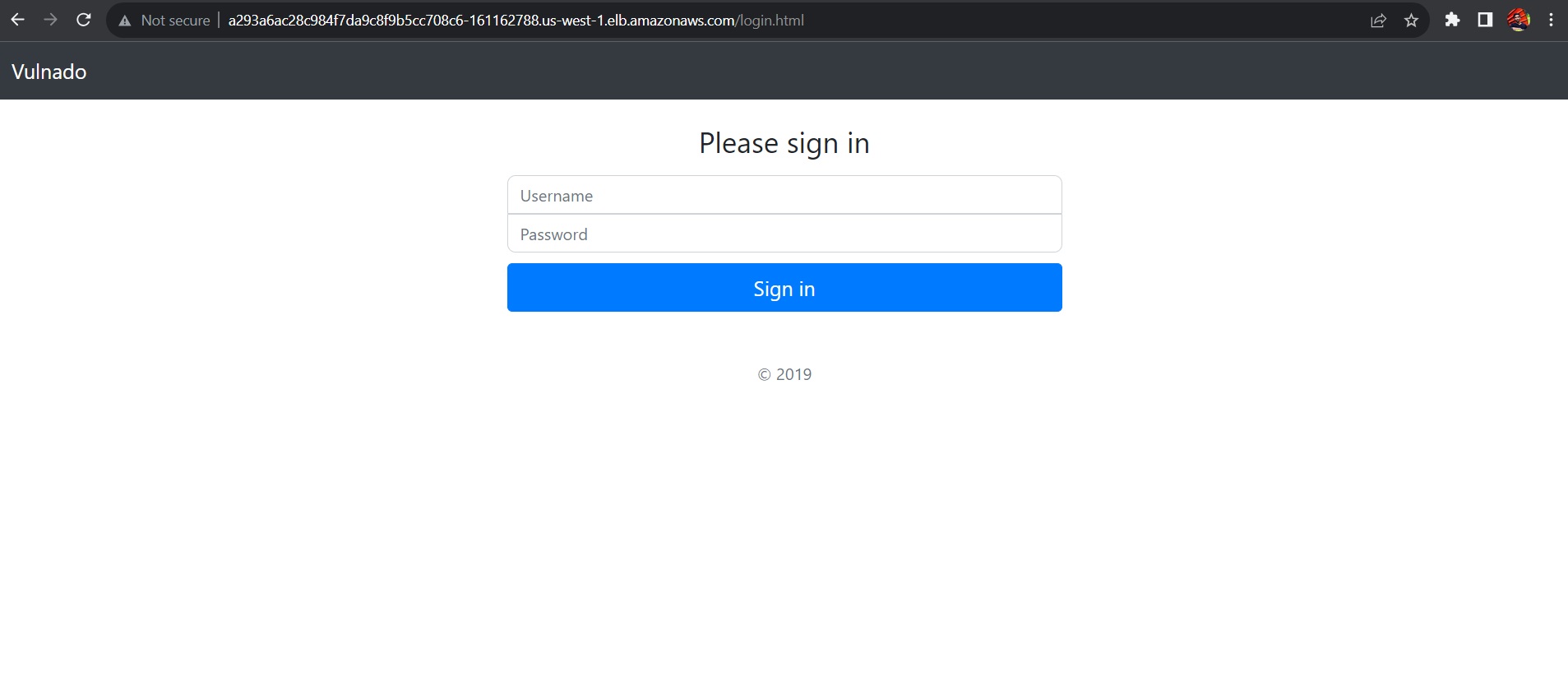
Build overview of pipeline:



Output:



Output:



**THE END**