LAPORAN AKHIR UAS DEVSECOPS
PENGAMANAN APLIKASI OWASP JUICE SHOP DENGAN PENDEKATAN DEVSECOPS

Dosen Pembimbing:

Khamarudin Syarif

Mata Kuliah Development Security Operation (DevSecOps)

Disusun Oleh:

Zico Nakano Morientes

88032023006

PROGRAM STUDI REKAYASA KEAMANAN SIBER

POLITEKNIK BHAKTI SEMESTA

2025

1. PENDAHULUAN

Perkembangan teknologi informasi dan meningkatnya penggunaan aplikasi berbasis web telah mendorong organisasi untuk mengembangkan sistem secara cepat dan berkelanjutan. Namun, kecepatan pengembangan tersebut sering kali tidak diimbangi dengan penerapan aspek keamanan yang memadai. Akibatnya, banyak aplikasi web rentan terhadap berbagai ancaman keamanan seperti kebocoran data, serangan terhadap mekanisme autentikasi, eksploitasi kerentanan dependensi, hingga gangguan layanan.

Pendekatan pengembangan tradisional yang memisahkan proses pengembangan, operasional, dan keamanan terbukti kurang efektif dalam menghadapi tantangan keamanan modern. Untuk mengatasi permasalahan tersebut, diperlukan pendekatan DevSecOps, yaitu integrasi keamanan ke dalam seluruh siklus pengembangan perangkat lunak (Secure Software Development Life Cycle) sejak tahap perencanaan hingga aplikasi berjalan di lingkungan operasional. Dengan pendekatan ini, keamanan tidak lagi menjadi proses terpisah, melainkan bagian yang terintegrasi dalam setiap tahapan pengembangan dan operasional aplikasi.

Pada proyek ini, aplikasi OWASP Juice Shop digunakan sebagai objek implementasi dan pengujian karena aplikasi tersebut merepresentasikan berbagai kerentanan keamanan yang umum ditemukan pada aplikasi web nyata. OWASP Juice Shop dirancang sebagai aplikasi pembelajaran, sehingga sangat sesuai untuk mensimulasikan penerapan DevSecOps, pengujian keamanan otomatis, pengamanan container, monitoring, serta penanganan insiden keamanan dalam lingkungan yang terkontrol.

Tujuan dari proyek ini adalah untuk menerapkan pendekatan DevSecOps secara menyeluruh pada aplikasi OWASP Juice Shop, mulai dari threat modeling, implementasi CI/CD pipeline, security testing otomatis, Docker hardening, monitoring dan observabilitas, hingga incident handling. Melalui proyek ini diharapkan dapat diperoleh pemahaman praktis mengenai bagaimana keamanan dapat diintegrasikan secara sistematis dalam proses pengembangan aplikasi modern serta pentingnya kolaborasi antara developer, operator, dan security engineer dalam membangun aplikasi yang aman dan andal.

2. LATAR BELAKANG

Perkembangan aplikasi berbasis web yang semakin pesat menuntut proses pengembangan perangkat lunak yang cepat, fleksibel, dan berkelanjutan. Namun, dalam praktiknya, percepatan proses pengembangan sering kali tidak diimbangi dengan penerapan keamanan yang memadai. Kondisi ini menyebabkan banyak aplikasi web memiliki celah keamanan yang dapat dimanfaatkan oleh pihak tidak bertanggung jawab, seperti serangan terhadap autentikasi, eksploitasi kerentanan dependensi, hingga kebocoran data pengguna.

Pendekatan pengembangan tradisional yang memisahkan peran pengembang, operator, dan tim keamanan terbukti kurang efektif dalam menghadapi ancaman keamanan modern. Keamanan sering kali baru diperhatikan pada tahap akhir pengembangan atau setelah aplikasi berjalan di lingkungan operasional, sehingga potensi risiko sudah terlanjur muncul. Oleh karena itu, diperlukan pendekatan yang mampu mengintegrasikan keamanan ke dalam seluruh siklus pengembangan aplikasi sejak tahap awal.

DevSecOps hadir sebagai pendekatan yang menggabungkan aspek pengembangan (Development), operasional (Operations), dan keamanan (Security) dalam satu alur kerja yang terintegrasi. Melalui DevSecOps, keamanan diterapkan secara berkelanjutan melalui praktik Secure SDLC, automasi pengujian keamanan, pengamanan infrastruktur dan container, monitoring, serta penanganan insiden. Pendekatan ini memungkinkan deteksi dan mitigasi risiko dilakukan lebih awal dan lebih konsisten.

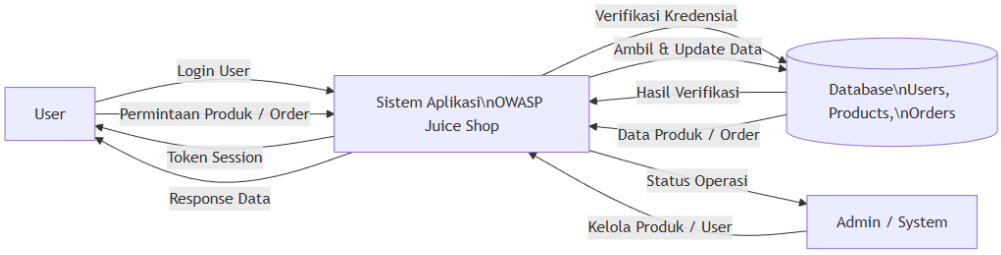
OWASP Juice Shop dipilih sebagai objek pada proyek ini karena aplikasi tersebut dirancang secara khusus untuk merepresentasikan berbagai kerentanan keamanan yang umum ditemukan pada aplikasi web nyata. Dengan menggunakan OWASP Juice Shop, penerapan DevSecOps dapat disimulasikan secara aman dalam lingkungan lokal, sehingga memungkinkan pembelajaran praktis mengenai pengujian keamanan, monitoring, dan incident handling tanpa risiko terhadap sistem produksi. Latar belakang inilah yang mendasari pelaksanaan proyek ini sebagai upaya untuk memahami dan menerapkan konsep DevSecOps secara komprehensif dalam pengamanan aplikasi web.

3. ARSITEKTUR SISTEM

Arsitektur sistem OWASP Juice Shop pada proyek ini menggunakan pendekatan berbasis container. Aplikasi dijalankan menggunakan Docker pada mesin lokal dengan sistem operasi Windows. Pengguna mengakses aplikasi melalui web browser, yang kemudian berkomunikasi dengan aplikasi Juice Shop yang berjalan di dalam container Docker. Aplikasi tersebut menangani proses autentikasi, pengelolaan data produk, serta interaksi dengan database internal aplikasi.

Selain komponen utama aplikasi, arsitektur sistem juga mencakup komponen pendukung keamanan dan observabilitas, seperti pipeline CI/CD di GitHub Actions, tool security testing (SAST, SCA, DAST), serta sistem monitoring dan logging. Arsitektur ini dirancang untuk mencerminkan lingkungan pengembangan modern yang menerapkan prinsip DevSecOps secara terintegrasi.

3.1. Diagram Arsitektur Sistem OWASP Juice Shop



Gambar 3.1 – Data Flow Diagram

Data Flow Diagram (DFD) Level 0 pada OWASP Juice Shop menggambarkan alur data utama antara pengguna, sistem aplikasi, database, dan admin. Pengguna melakukan proses login dan mengirimkan permintaan data ke sistem aplikasi. Sistem memverifikasi kredensial pengguna ke database dan mengelola session token yang dikirimkan kembali ke pengguna. Sistem juga melakukan proses pengambilan dan pembaruan data produk serta transaksi ke database. Admin berinteraksi dengan sistem untuk melakukan pengelolaan data aplikasi. Diagram ini digunakan sebagai dasar dalam proses threat modeling dan analisis risiko keamanan.

3.2. Penjelasan Alur Data Sistem (Sesuai Diagram DFD)

1. **User → Sistem Aplikasi OWASP Juice Shop**

   Alur ini menggambarkan interaksi awal antara pengguna dan sistem. User mengirimkan data login berupa username dan password ke sistem aplikasi untuk proses autentikasi. Selain itu, user juga mengirimkan permintaan untuk melihat daftar produk, melakukan pemesanan, atau aktivitas lain yang tersedia pada aplikasi. Seluruh permintaan tersebut diproses oleh sistem sesuai dengan hak akses pengguna.

2. **Sistem Aplikasi OWASP Juice Shop → User**

   Setelah permintaan diproses, sistem mengirimkan respon kembali kepada user. Jika proses login berhasil, sistem akan mengirimkan token session yang digunakan sebagai identitas pengguna selama sesi berlangsung. Untuk permintaan lainnya, sistem mengembalikan data berupa informasi produk, status pemesanan, atau pesan hasil proses yang dilakukan oleh user.

3. **Sistem Aplikasi OWASP Juice Shop ↔ Database**

   Pada alur ini, sistem melakukan komunikasi dua arah dengan database. Sistem mengirimkan permintaan verifikasi kredensial user saat proses login berlangsung. Selain itu, sistem juga melakukan pengambilan, penyimpanan, dan pembaruan data yang berkaitan dengan pengguna, produk, serta transaksi pemesanan. Database mengembalikan hasil verifikasi dan data yang dibutuhkan oleh sistem untuk diproses lebih lanjut.

4. **Admin → Sistem Aplikasi OWASP Juice Shop**

   Admin berinteraksi dengan sistem untuk melakukan manajemen aplikasi, seperti menambah, mengubah, atau menghapus data produk, mengelola akun pengguna, serta melakukan konfigurasi sistem. Akses admin bersifat terbatas dan memiliki hak istimewa dibandingkan user biasa.

5. **Sistem Aplikasi OWASP Juice Shop → Admin**

   Sistem memberikan umpan balik kepada admin berupa status dan hasil dari setiap operasi yang dilakukan. Informasi ini dapat berupa konfirmasi perubahan data, notifikasi keberhasilan atau kegagalan proses, serta informasi lain yang berkaitan dengan pengelolaan aplikasi.

4. SECURE SDLC & THREAT MODELING

Penerapan Secure SDLC dilakukan dengan memasukkan aspek keamanan pada setiap fase pengembangan aplikasi. Pada tahap perencanaan, dilakukan identifikasi aset penting seperti akun pengguna, mekanisme autentikasi, database aplikasi, API endpoint, dan session token. Aset-aset ini menjadi fokus utama dalam proses threat modeling.

Threat modeling dilakukan menggunakan metode STRIDE untuk mengidentifikasi jenis ancaman seperti spoofing, tampering, repudiation, information disclosure, denial of service, dan elevation of privilege. Setiap ancaman kemudian dianalisis tingkat risikonya menggunakan metode DREAD yang menilai dampak kerusakan, kemudahan eksploitasi, serta jumlah pengguna yang terdampak. Hasil threat modeling ini digunakan sebagai dasar dalam menentukan prioritas pengamanan dan pengujian keamanan pada tahap selanjutnya.

4.1. Identifikasi Aset Penting

Berdasarkan analisis sistem, berikut adalah aset-aset penting yang perlu diamankan pada aplikasi OWASP Juice Shop:

No	Aset	Deskripsi
1	User Account	Data akun pengguna termasuk username dan profile
2	Authentication Mechanism	Proses login dan validasi kredensial
3	Database	Penyimpanan data user, produk, dan transaksi
4	API Endpoint	Endpoint backend yang menangani request aplikasi
5	Session / Token	Informasi sesi pengguna setelah login

4.2. Threat Modeling dengan STRIDE

Metode STRIDE digunakan untuk mengidentifikasi jenis ancaman yang mungkin terjadi terhadap aset-aset penting aplikasi.

Aset	Ancaman	Kategori STRIDE	Deskripsi Ancaman
User Account	Brute force login	Spoofing	Penyerang mencoba mengambil alih akun
Authentication Mechanism	Credential interception	Information Disclosure	Kredensial dapat bocor saat transmisi
Database	Manipulasi data	Tampering	Perubahan data tanpa otorisasi
API Endpoint	Flooding request	Denial of Service	Layanan menjadi tidak tersedia
Session / Token	Session hijacking	Elevation of Privilege	Penyerang memperoleh akses lebih tinggi

4.3. Analisis Risiko dengan DREAD

Setiap ancaman yang telah diidentifikasi kemudian dianalisis tingkat risikonya menggunakan metode DREAD.

Threat	D	R	E	A	D	Risk Level
Brute Force Login	8	9	8	7	8	Tinggi
Data Tampering	9	7	7	8	6	Tinggi
DoS Attack	7	8	8	9	7	Tinggi
Session Hijacking	9	6	6	7	6	Sedang
Credential Leak	8	7	6	6	7	Sedang

Hasil analisis DREAD menunjukkan bahwa serangan brute force login, manipulasi data, dan denial of service memiliki tingkat risiko tinggi sehingga perlu menjadi prioritas utama dalam penerapan kontrol keamanan dan pengujian pada tahap selanjutnya.

5. CI/CD Pipeline

Pipeline CI/CD diimplementasikan menggunakan GitHub Actions untuk memastikan proses build dan pengujian keamanan berjalan otomatis setiap kali terjadi perubahan kode. Pipeline dirancang dengan tahapan berurutan yang mencerminkan praktik DevSecOps, dimulai dari pengambilan kode (code), analisis keamanan statis (SAST), analisis dependensi (SCA), proses build aplikasi, pemindaian keamanan container, hingga pengujian keamanan dinamis (DAST).

Setiap tahapan pipeline dijalankan secara otomatis dan menghasilkan output yang dapat digunakan untuk mengevaluasi kondisi keamanan aplikasi. Pipeline ini memungkinkan deteksi kerentanan dilakukan lebih awal sehingga risiko dapat diminimalkan sebelum aplikasi dijalankan di lingkungan produksi.

5.1. Tahapan CI/CD Pipeline

1. **Code (Source Code Management)**

   Tahap ini merupakan proses pengambilan source code dari repository GitHub. Setiap perubahan kode menjadi pemicu otomatis bagi pipeline untuk berjalan, sehingga memastikan tidak ada perubahan yang lolos tanpa melalui pemeriksaan keamanan.

2. **SAST (Static Application Security Testing)**

   Pada tahap ini dilakukan analisis keamanan statis terhadap kode dan dependensi aplikasi menggunakan tool seperti npm audit. Tujuannya adalah untuk mendeteksi kerentanan keamanan sejak awal tanpa menjalankan aplikasi.

3. **SCA (Software Composition Analysis)**

   Tahap SCA berfokus pada pemeriksaan kerentanan yang berasal dari library dan dependency pihak ketiga. Tool seperti Trivy digunakan untuk mengidentifikasi vulnerability pada komponen yang digunakan oleh aplikasi.

4. **Build**

   Setelah proses analisis statis selesai, aplikasi dibangun menjadi image Docker. Tahap ini memastikan bahwa aplikasi dapat dikompilasi dan dijalankan dengan baik sebelum dilanjutkan ke tahap pengujian lanjutan.

5. **Container Scan**

   Image Docker hasil build kemudian dipindai menggunakan Trivy untuk mendeteksi kerentanan pada sistem operasi container, package, dan konfigurasi

image. Tahap ini penting untuk memastikan keamanan lingkungan runtime aplikasi.

6. **DAST (Dynamic Application Security Testing)**

   Tahap terakhir adalah pengujian keamanan dinamis menggunakan OWASP ZAP terhadap aplikasi yang sedang berjalan. Pengujian ini mensimulasikan serangan dari sisi eksternal untuk mendeteksi kerentanan seperti XSS, SQL Injection, dan konfigurasi yang tidak aman.

5.2. Implementasi Pipeline

Pipeline diimplementasikan dalam sebuah file konfigurasi GitHub Actions dengan format YAML yang disimpan pada direktori:

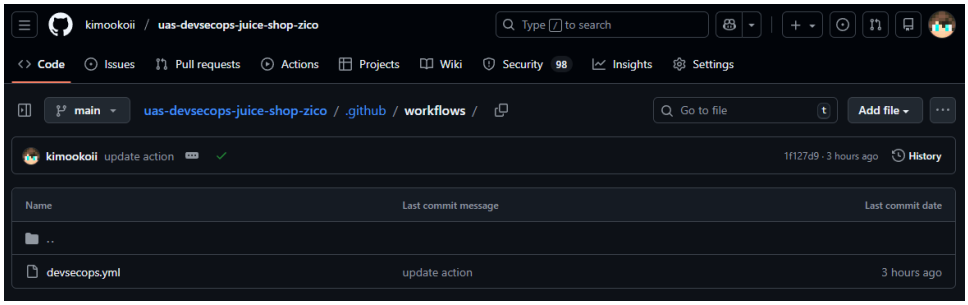
.github/workflows/devsecops.yml

File ini berisi definisi job dan langkah-langkah pipeline yang dijalankan secara otomatis oleh GitHub Actions.

5.3. Output CI/CD Pipeline

Output yang dihasilkan dari tahap CI/CD Pipeline meliputi:

- File konfigurasi pipeline



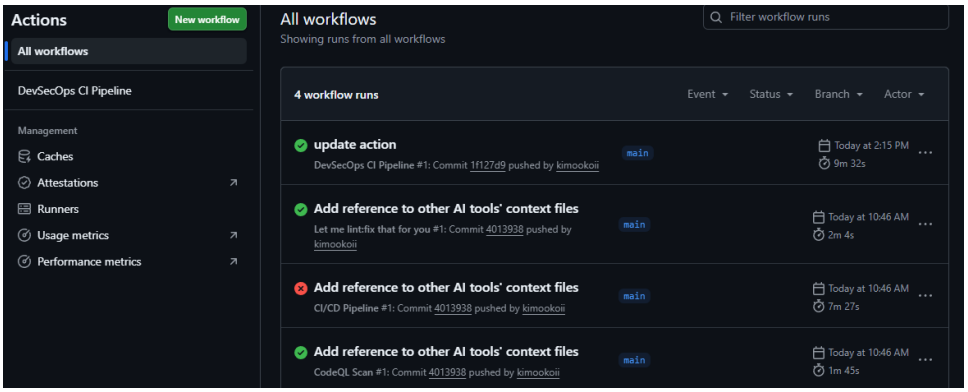
Gambar 5.1 – File konfigurasi pipeline pada GitHub

```yaml
name: DevSecOps CI Pipeline

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

permissions:
  contents: read
  checks: write
  issues: write

jobs:
  devsecops:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Source Code
      uses: actions/checkout@v4

    # SAST - npm audit
    - name: Install Dependencies
      run: npm install

    - name: SAST - npm audit
      run: |
        npm audit --audit-level=high || true
```

```yaml
# SCA - Trivy FS Scan
- name: Install Trivy
  run: |
    sudo apt-get update
    sudo apt-get install -y curl
    curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sudo sh -s -- -b /usr/local/bin

- name: SCA - Trivy File System Scan
  run: |
    trivy fs . --severity HIGH,CRITICAL || true

# Build Docker Image
- name: Build Docker Image
  run: |
    docker build -t juice-shop:ci .

# Container Scan - Trivy
- name: Container Image Scan
  run: |
    trivy image juice-shop:ci --severity HIGH,CRITICAL || true

# Run Juice Shop Container
- name: Run Juice Shop
  run: |
    docker run -d -p 3000:3000 --name juice-shop-ci juice-shop:ci
    sleep 30

# DAST - OWASP ZAP
- name: DAST - OWASP ZAP
  run: |
    docker run --rm \
      --network host \
      zaproxy/zap-stable \
      zap-baseline.py \
      -t http://localhost:3000 \
      -I || true
```

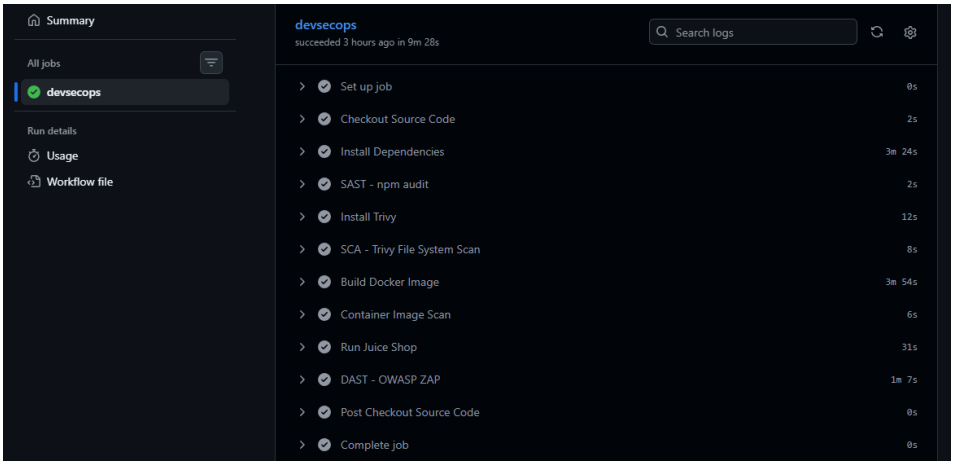
Gambar 5.2 – Isi file konfigurasi pipeline

- Status pipeline berhasil atau gagal pada setiap tahapan



Gambar 5.3 – Status konfigurasi pipeline pada GitHub

- Log eksekusi masing masing tahap keamanan



Gambar 5.4 – Log eksekusi pipeline

6. SECURITY TESTING

Pada tahap ini dilakukan pengujian keamanan otomatis terhadap aplikasi OWASP Juice Shop sebagai bagian dari implementasi DevSecOps. Security testing bertujuan untuk mengidentifikasi kerentanan keamanan baik pada source code, dependensi aplikasi, container, maupun perilaku aplikasi saat dijalankan. Pengujian dilakukan menggunakan beberapa jenis tools yang mewakili pendekatan SAST, SCA, dan DAST, sehingga memberikan cakupan pengujian yang menyeluruh.

Seluruh pengujian keamanan dijalankan secara nyata dan hasilnya dianalisis untuk mengetahui tingkat risiko yang mungkin berdampak pada keamanan aplikasi. Meskipun tidak semua kerentanan diperbaiki, hasil pengujian ini menjadi dasar evaluasi keamanan.

6.1. Jenis dan Tahapan Security Testing

1. **Static Application Security Testing (SAST)**

   Pengujian SAST dilakukan menggunakan tool npm audit untuk menganalisis source code dan dependensi aplikasi tanpa menjalankan aplikasi. Pengujian ini bertujuan untuk mendeteksi kerentanan yang berasal dari penggunaan library atau konfigurasi kode yang tidak aman sejak tahap awal pengembangan.

2. **Software Composition Analysis (SCA)**

   SCA dilakukan untuk mengidentifikasi kerentanan pada komponen pihak ketiga dan dependensi yang digunakan oleh aplikasi. Tool Trivy digunakan untuk memindai dependensi dan package yang terintegrasi dalam aplikasi OWASP Juice Shop. Tahap ini penting karena banyak kerentanan modern berasal dari library eksternal yang sudah usang atau rentan.

3. **Dynamic Application Security Testing (DAST)**

   Pengujian DAST dilakukan menggunakan OWASP ZAP dengan melakukan pemindaian terhadap aplikasi yang sedang berjalan. Pengujian ini mensimulasikan serangan dari sisi eksternal untuk mengidentifikasi kerentanan seperti Cross-Site Scripting (XSS), SQL Injection, serta kesalahan konfigurasi keamanan pada web.

6.2. Hasil dan Analisis Pengujian

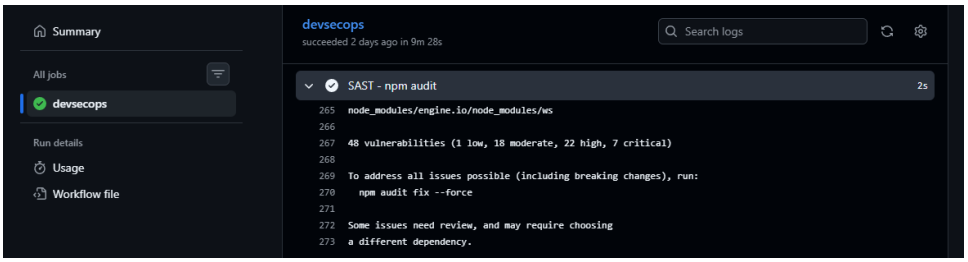
Hasil pengujian keamanan menunjukkan adanya beberapa kerentanan dengan tingkat risiko Critical dan High, khususnya yang berasal dari dependensi pihak ketiga dan konfigurasi aplikasi. Temuan-temuan tersebut dianalisis berdasarkan tingkat dampak dan

kemungkinan eksploitasi terhadap sistem. Kerentanan dengan risiko tinggi menjadi perhatian utama untuk tahap pengamanan lanjutan, seperti hardening container dan penerapan security gate pada pipeline.

6.3. Output Security Testing

Output yang dihasilkan pada tahap security testing meliputi:

1. **Hasil npm audit (SAST)**



Gambar 6.1 – Hasil SAST - npm audit

Berdasarkan hasil pengujian SAST menggunakan npm audit, ditemukan total 48 kerentanan, yang terdiri dari 7 Critical dan 22 High. Kerentanan tersebut sebagian besar berasal dari dependensi pihak ketiga yang digunakan oleh aplikasi OWASP Juice Shop. Berikut adalah tiga temuan utama dengan tingkat risiko tertinggi.

1. **Kerentanan Sandbox Escape pada vm2 (Critical)**

   Kerentanan pada library vm2 memungkinkan terjadinya sandbox escape yang dapat dieksploitasi oleh penyerang untuk mengeksekusi kode di luar lingkungan sandbox. Jika berhasil dimanfaatkan, kerentanan ini berpotensi memberikan akses penuh ke sistem host aplikasi, sehingga memungkinkan pencurian data sensitif, eksekusi perintah berbahaya, hingga pengambilalihan aplikasi secara keseluruhan.

   Mitigasi:

   - Memperbarui dependensi ke versi vm2 yang telah diperbaiki
   - Menghapus fitur atau modul yang bergantung pada vm2 jika tidak diperlukan
   - Membatasi hak akses container (non-root, isolation)

2. **Prototype Pollution pada lodash (Critical)**

   Library lodash memiliki kerentanan prototype pollution yang memungkinkan penyerang memanipulasi struktur objek JavaScript. Eksploitasi kerentanan ini dapat menyebabkan perubahan alur logika aplikasi, perilaku sistem yang tidak

semestinya, kebocoran data, serta berpotensi menjadi pintu masuk bagi serangan lanjutan seperti remote code execution.

Mitigasi:

- Memperbarui lodash ke versi terbaru yang aman
- Menghindari penggunaan fungsi lodash yang rentan
- Menambahkan validasi input secara ketat

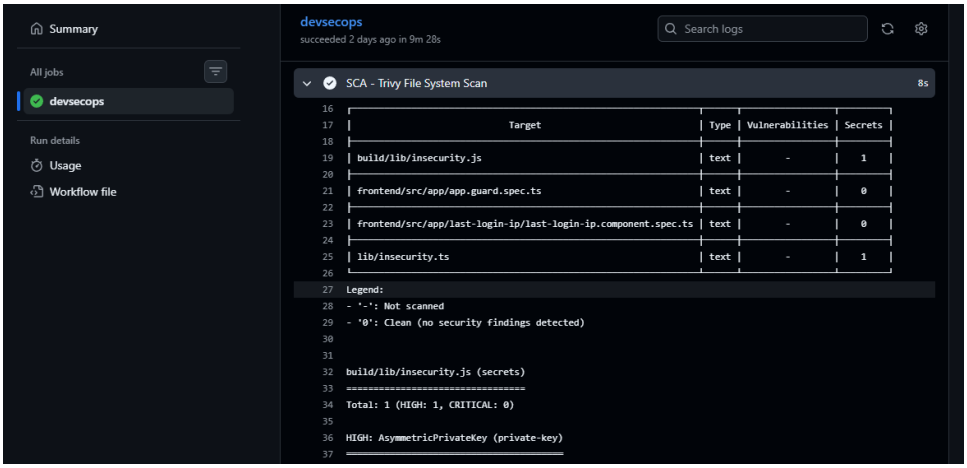
3. Uncontrolled Resource Consumption pada braces (High)

Kerentanan pada library braces dapat menyebabkan konsumsi resource yang berlebihan akibat pemrosesan pola input tertentu. Kondisi ini dapat dimanfaatkan oleh penyerang untuk melakukan serangan Denial of Service (DoS), yang berdampak pada penurunan performa aplikasi atau bahkan membuat layanan tidak dapat diakses oleh pengguna yang sah.

Mitigasi:

- Memperbarui dependensi terkait (braces, micromatch)
- Membatasi ukuran dan kompleksitas input pengguna
- Mengaktifkan rate limiting pada endpoint yang rentan

2. Hasil scan Trivy (SCA)



Gambar 6.2 – Hasil SCA - Trivy

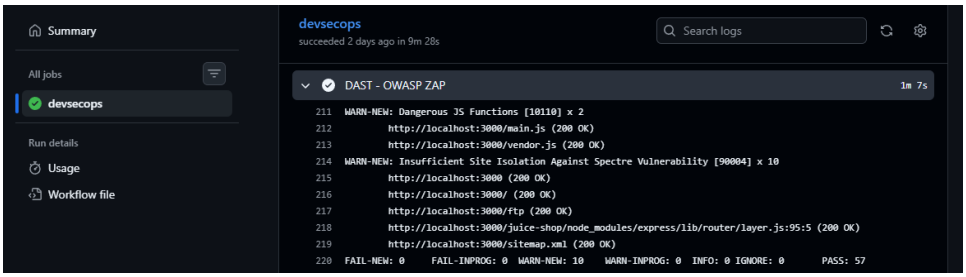
Berdasarkan hasil Software Composition Analysis (SCA), tidak ditemukan kerentanan dengan tingkat Critical, namun teridentifikasi kerentanan tingkat High berupa penyimpanan Asymmetric Private Key (RSA Private Key) secara langsung di dalam source code aplikasi. Temuan ini muncul pada file build/lib/insecurity.js dan lib/insecurity.ts, yang menunjukkan adanya secret sensitif yang tertanam (hardcoded) dalam kode sumber.

Keberadaan private key di dalam repository kode berisiko tinggi karena dapat dimanfaatkan oleh pihak tidak berwenang untuk melakukan pemalsuan token, manipulasi autentikasi, dan akses ilegal ke sistem backend apabila kode tersebut bocor atau diakses oleh penyerang.

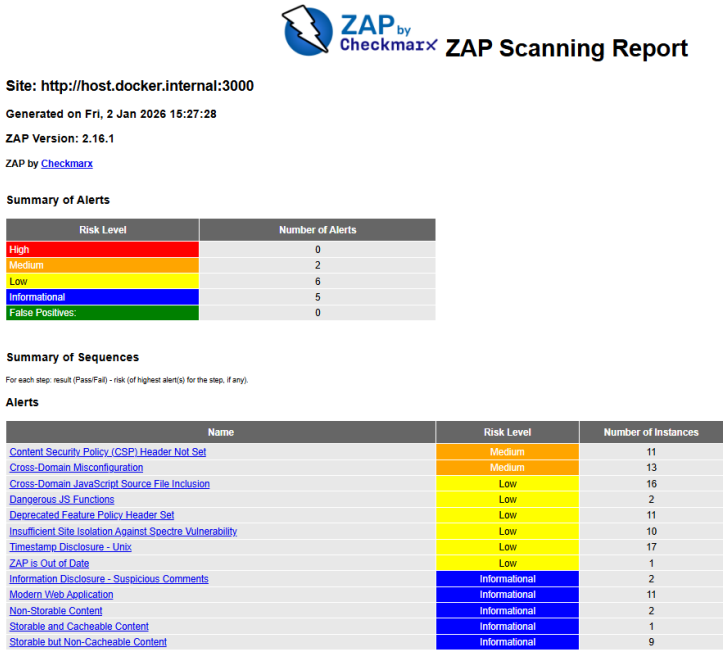
Mitigasi:

- Hapus private key dari source code. Private key tidak boleh disimpan langsung di dalam kode aplikasi atau repository.
- Simpan key sensitif menggunakan environment variable atau secret management tool agar tidak terekspos di kode.
- Segera lakukan rotasi (generate ulang) key yang terlanjur terekspos.

3. Laporan pemindaian OWASP ZAP (DAST)



Gambar 6.3 – Hasil DAST - OWASP ZAP



Gambar 6.4 – File hasil OWASP ZAP (zap_report.html)

Hasil pemindaian DAST menunjukkan tidak adanya kerentanan kritis yang dapat dieksploitasi secara langsung, namun ditemukan beberapa kelemahan konfigurasi

keamanan dengan tingkat risiko High. Meskipun OWASP ZAP tidak mencatat temuan kategori FAIL, sejumlah temuan WARN berpotensi dimanfaatkan oleh penyerang sehingga diperlukan penerapan header keamanan, pembatasan resource lintas domain, serta praktik pengkodean JavaScript yang lebih aman untuk meningkatkan keamanan aplikasi.

1. **Content Security Policy (CSP) Header Not Set — High**

   Hasil pemindaian DAST menunjukkan bahwa aplikasi belum menerapkan header Content-Security-Policy (CSP) pada beberapa endpoint utama. Kondisi ini memungkinkan browser memuat dan mengeksekusi skrip dari sumber yang tidak dibatasi, sehingga meningkatkan risiko serangan Cross-Site Scripting (XSS) dan penyisipan skrip berbahaya. Untuk mengurangi risiko tersebut, disarankan menambahkan konfigurasi header CSP yang membatasi sumber skrip, style, dan resource hanya dari domain yang terpercaya.

2. **Cross-Domain JavaScript Source File Inclusion — High**

   Aplikasi ditemukan memuat file JavaScript dari domain eksternal tanpa pembatasan keamanan yang memadai. Jika domain pihak ketiga tersebut berhasil dikompromikan, penyerang dapat menyisipkan kode JavaScript berbahaya yang akan dieksekusi di sisi pengguna. Mitigasi yang disarankan adalah membatasi pemuatan skrip lintas domain, menerapkan Subresource Integrity (SRI), serta hanya mengizinkan domain eksternal yang benar-benar dibutuhkan.

3. **Dangerous JavaScript Functions Detected — High**

   Pemindaian juga menemukan penggunaan fungsi JavaScript berisiko seperti eval() pada file aplikasi, yang dapat dimanfaatkan untuk mengeksekusi kode arbitrer apabila input tidak divalidasi dengan baik. Penggunaan fungsi tersebut meningkatkan risiko serangan XSS dan manipulasi kode di sisi klien. Untuk mengurangi risiko ini, pengembang disarankan menghindari penggunaan fungsi JavaScript berbahaya, menerapkan validasi input yang ketat, dan menggunakan metode pemrosesan data yang lebih aman.

7. DOCKER & IAC SECURITY

Pada tahap ini dilakukan evaluasi dan peningkatan keamanan container dengan membuat dua versi Dockerfile, yaitu versi awal yang masih belum aman dan versi yang telah di-hardening. Dockerfile insecure menggunakan konfigurasi default, base image berukuran besar, serta menjalankan aplikasi sebagai user root. Pada Dockerfile hardened, dilakukan perbaikan dengan menggunakan base image yang lebih minimal, menjalankan aplikasi sebagai non-root user, dan menerapkan best practice keamanan container.

Dockerfile versi insecure menggunakan konfigurasi default, base image berukuran besar, serta menjalankan aplikasi dengan hak akses user root. Konfigurasi tersebut berpotensi meningkatkan risiko keamanan apabila container berhasil dikompromikan, karena penyerang dapat memperoleh akses penuh ke dalam container.

Sebagai perbaikan, Dockerfile versi hardened dibuat dengan menggunakan base image yang lebih minimal untuk mengurangi attack surface, menjalankan aplikasi menggunakan user non-root, serta menerapkan praktik keamanan seperti pembatasan permission dan penghapusan package yang tidak diperlukan. Pendekatan ini bertujuan untuk meminimalkan dampak apabila terjadi eksploitasi terhadap aplikasi.

7.1. Perbandingan Dockerfile Insecure dan Hardened

Dockerfile Insecure (Sebelumnya)

Dockerfile

```
FROM node:22 AS installer
COPY . /juice-shop
WORKDIR /juice-shop
RUN npm i -g typescript ts-node
RUN npm install --omit=dev --unsafe-perm
RUN npm dedupe --omit=dev
RUN rm -rf frontend/node_modules
RUN rm -rf frontend/.angular
RUN rm -rf frontend/src/assets
RUN mkdir logs
RUN chown -R 65532 logs
RUN chgrp -R 0 ftp/ frontend/dist/ logs/ data/ i18n/
RUN chmod -R g=u ftp/ frontend/dist/ logs/ data/ i18n/
RUN rm data/chatbot/botDefaultTrainingData.json || true
RUN rm ftp/legal.md || true
RUN rm i18n/*.json || true

ARG CYCLONEDX_NPM_VERSION=latest
RUN npm install -g @cyclonedx/cyclonedx-npm@$CYCLONEDX_NPM_VERSION
RUN npm run sbom

FROM gcr.io/distroless/nodejs22-debian12
ARG BUILD_DATE
ARG VCS_REF
LABEL maintainer="Bjoern Kimminich <bjoern.kimminich@owasp.org>" \
    org.opencontainers.image.title="OWASP Juice Shop" \
    org.opencontainers.image.description="Probably the most modern and sophisticated insecure web
application" \
```

```
    org.opencontainers.image.authors="Bjoern Kimminich <bjoern.kimminich@owasp.org>" \
    org.opencontainers.image.vendor="Open Worldwide Application Security Project" \
    org.opencontainers.image.documentation="https://help.owasp-juice.shop" \
    org.opencontainers.image.licenses="MIT" \
    org.opencontainers.image.version="19.1.1" \
    org.opencontainers.image.url="https://owasp-juice.shop" \
    org.opencontainers.image.source="https://github.com/juice-shop/juice-shop" \
    org.opencontainers.image.revision=$VCS_REF \
    org.opencontainers.image.created=$BUILD_DATE
WORKDIR /juice-shop
COPY --from=installer --chown=65532:0 /juice-shop .
USER 65532
EXPOSE 3000
CMD ["/juice-shop/build/app.js"]
```

Gambar 7.1 – Isi file DockerFile (insecure)

- Base image builder besar (node:22)
- Banyak tool build tertinggal (typescript, ts-node)
- Copy seluruh source ke runtime
- Attack surface lebih luas
- Sulit diterapkan security gate ketat

Dockerfile Secure (Hardened)

Dockerfile

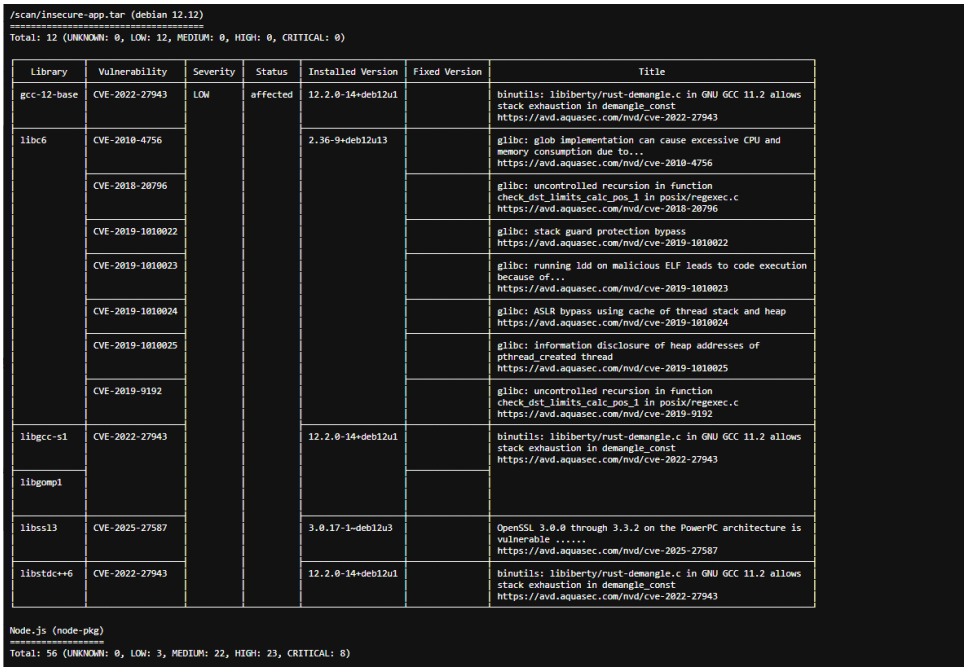
```
# Stage 1: Builder
FROM node:22-alpine AS builder

# Install git (needed by npm)
RUN apk add --no-cache git
WORKDIR /juice-shop

# Copy FULL source first (required by Juice Shop)
COPY . .

# Install production dependencies
RUN npm install --omit=dev \
    && npm cache clean --force

# Stage 2: Runtime (Hardened)
FROM gcr.io/distroless/nodejs22-debian12

ARG BUILD_DATE
ARG VCS_REF

LABEL org.opencontainers.image.title="OWASP Juice Shop (Hardened)" \
      org.opencontainers.image.created=$BUILD_DATE \
      org.opencontainers.image.revision=$VCS_REF

# Non-root user
USER 65532
WORKDIR /juice-shop

# Copy only runtime artifacts
COPY --from=builder --chown=65532:0 /juice-shop/build ./build
COPY --from=builder --chown=65532:0 /juice-shop/node_modules ./node_modules
COPY --from=builder --chown=65532:0 /juice-shop/config ./config
COPY --from=builder --chown=65532:0 /juice-shop/data ./data
COPY --from=builder --chown=65532:0 /juice-shop/i18n ./i18n

EXPOSE 3000
CMD ["build/app.js"]
```

Gambar 7.2 – Isi file DockerFile (secure)

1. Base image builder minimal (node:22-alpine)
2. Runtime menggunakan distroless
3. Hanya file penting yang dicopy
4. Tidak ada shell di runtime container
5. Aplikasi berjalan sebagai non-root user
6. Lebih sedikit CVE saat scan Trivy

7.2. Ringkasan Hasil Scan Trivy

Dockerfile Insecure (Sebelumnya)



/scan/insecure-app.tar (debian 12.12)
==========
Total: 12 (UNKNOWN: 0, LOW: 12, MEDIUM: 0, HIGH: 0, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
gcc-12-base	CVE-2022-27943	LOW	affected	12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943
libc6	CVE-2010-4756			2.36-9+deb12u13		glibc: glob implementation can cause excessive CPU and memory consumption due to... https://avd.aquasec.com/nvd/cve-2010-4756
	CVE-2018-20796					glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c https://avd.aquasec.com/nvd/cve-2018-20796
	CVE-2019-1010022					glibc: stack guard protection bypass https://avd.aquasec.com/nvd/cve-2019-1010022
	CVE-2019-1010023					glibc: running ldd on malicious ELF leads to code execution because of... https://avd.aquasec.com/nvd/cve-2019-1010023
	CVE-2019-1010024					glibc: ASLR bypass using cache of thread stack and heap https://avd.aquasec.com/nvd/cve-2019-1010024
	CVE-2019-1010025					glibc: information disclosure of heap addresses of pthread_created thread https://avd.aquasec.com/nvd/cve-2019-1010025
	CVE-2019-9192					glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c https://avd.aquasec.com/nvd/cve-2019-9192
libgcc-s1	CVE-2022-27943			12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943
libgomp1						
libss13	CVE-2025-27587			3.0.17-1~deb12u3		OpenSSL 3.0.0 through 3.3.2 on the PowerPC architecture is vulnerable https://avd.aquasec.com/nvd/cve-2025-27587
libstdc++6	CVE-2022-27943			12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943

Node.js (node-pkg)
==========
Total: 56 (UNKNOWN: 0, LOW: 3, MEDIUM: 22, HIGH: 23, CRITICAL: 8)

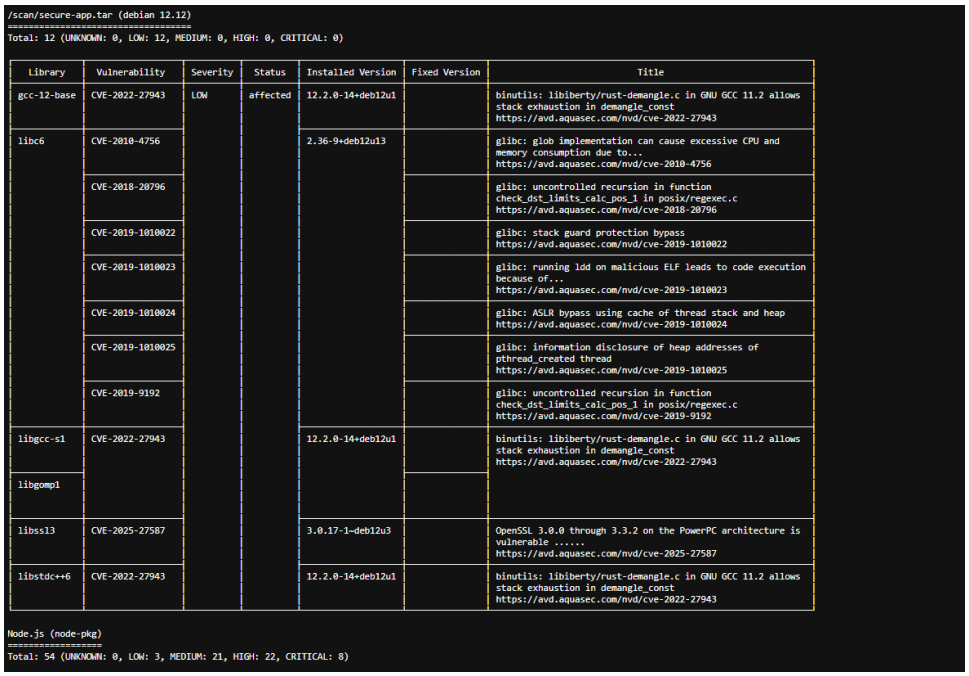
Gambar 7.3 – Hasil scan Trivy (insecure)

Hasil pemindaian Trivy terhadap image Dockerfile insecure menunjukkan bahwa pada layer sistem operasi (Debian 12.12) ditemukan 12 kerentanan dengan tingkat Low dan tidak ditemukan kerentanan dengan tingkat Medium, High, maupun Critical. Kerentanan tersebut berasal dari package bawaan sistem seperti glibc, gcc, dan libssl, yang umumnya merupakan risiko residual dari base image.

Namun, pada layer Node.js dependencies, ditemukan total 56 kerentanan, yang terdiri dari 8 Critical, 23 High, 22 Medium, dan 3 Low. Kerentanan kritis dan tinggi terutama berasal dari library pihak ketiga seperti jsonwebtoken, lodash, vm2, crypto-js, dan marsdb, yang berpotensi menyebabkan dampak serius seperti authentication bypass, prototype pollution, command injection, dan sandbox escape. Temuan ini menunjukkan bahwa meskipun base image relatif aman, dependensi aplikasi pada

Dockerfile insecure memiliki tingkat risiko keamanan yang tinggi dan memerlukan hardening lebih lanjut.

Dockerfile Secure (Hardened)



/scan/secure-app.tar (debian 12.12)
==========
Total: 12 (UNKNOWN: 0, LOW: 12, MEDIUM: 0, HIGH: 0, CRITICAL: 0)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
gcc-12-base	CVE-2022-27943	LOW	affected	12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943
libc6	CVE-2010-4756			2.36-9+deb12u13		glibc: glob implementation can cause excessive CPU and memory consumption due to... https://avd.aquasec.com/nvd/cve-2010-4756
	CVE-2018-20796					glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c https://avd.aquasec.com/nvd/cve-2018-20796
	CVE-2019-1010022					glibc: stack guard protection bypass https://avd.aquasec.com/nvd/cve-2019-1010022
	CVE-2019-1010023					glibc: running ldd on malicious ELF leads to code execution because of... https://avd.aquasec.com/nvd/cve-2019-1010023
	CVE-2019-1010024					glibc: ASLR bypass using cache of thread stack and heap https://avd.aquasec.com/nvd/cve-2019-1010024
	CVE-2019-1010025					glibc: information disclosure of heap addresses of pthread_created thread https://avd.aquasec.com/nvd/cve-2019-1010025
	CVE-2019-9192					glibc: uncontrolled recursion in function check_dst_limits_calc_pos_1 in posix/regexec.c https://avd.aquasec.com/nvd/cve-2019-9192
libgcc-s1	CVE-2022-27943			12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943
libgomp1						
libssl3	CVE-2025-27587			3.0.17-1~deb12u3		OpenSSL 3.0.0 through 3.3.2 on the PowerPC architecture is vulnerable https://avd.aquasec.com/nvd/cve-2025-27587
libstdc++6	CVE-2022-27943			12.2.0-14+deb12u1		binutils: libiberty/rust-demangle.c in GNU GCC 11.2 allows stack exhaustion in demangle_const https://avd.aquasec.com/nvd/cve-2022-27943

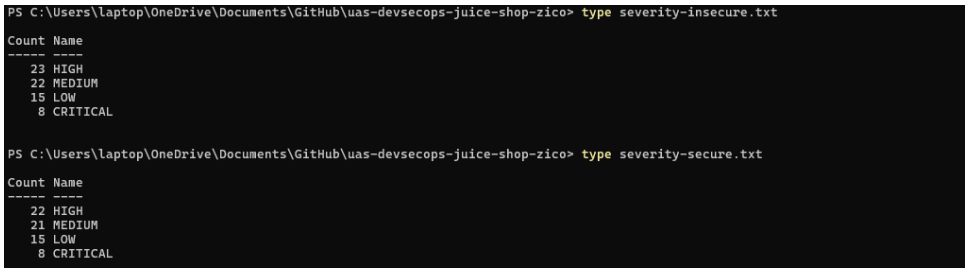
Node.js (node-pkg)
==========
Total: 54 (UNKNOWN: 0, LOW: 3, MEDIUM: 21, HIGH: 22, CRITICAL: 8)

Gambar 7.4 – Hasil scan Trivy (hardened)

Hasil pemindaian Trivy terhadap image Dockerfile hardened menunjukkan bahwa pada layer sistem operasi (Debian 12.12) masih ditemukan 12 kerentanan dengan tingkat Low, yang berasal dari package bawaan base image seperti glibc, gcc, dan libssl. Tidak ditemukan kerentanan dengan tingkat Medium, High, maupun Critical pada layer sistem operasi, sehingga risiko dari sisi base image tergolong rendah dan.

Pada layer Node.js dependencies, hasil scan menunjukkan total 54 kerentanan, yang terdiri dari 8 Critical, 22 High, 21 Medium, dan 3 Low. Kerentanan yang terdeteksi masih didominasi oleh library pihak ketiga seperti jsonwebtoken, lodash, vm2, crypto-js, dan marsdb. Temuan ini menunjukkan bahwa hardening pada Dockerfile berfokus pada pengurangan attack surface container dan bukan pada pembaruan dependensi aplikasi, sehingga sebagian besar risiko aplikasi tetap berasal dari komponen library yang memang sengaja rentan pada OWASP Juice Shop.

7.3. Perbedaan Hasil Scan Trivy: Insecure vs Hardened



Gambar 7.5 – Hasil perbedaan scan Trivy (insecure vs hardened)

Aspek	Dockerfile Insecure	Dockerfile Hardened
Kerentanan OS	12 Low	12 Low
Kerentanan Node.js	56 total	54 total
Critical	8	8
High	23	22
Medium	22	21
Low	3	3
Attack Surface	Lebih luas	Lebih kecil
User Runtime	Berpotensi root	Non-root
File Runtime	Semua source	Hanya file penting
Kesiapan Security Gate	Terbatas	Lebih siap

7.4. Kesimpulan Perbedaan

Dockerfile hardened tidak secara signifikan menurunkan jumlah kerentanan pada dependensi aplikasi, namun berhasil mengurangi attack surface container, meningkatkan isolasi runtime, dan memperbaiki postur keamanan lingkungan eksekusi. Hal ini menunjukkan bahwa hardening container berfungsi sebagai lapisan mitigasi tambahan, sementara perbaikan kerentanan aplikasi tetap memerlukan pembaruan dan pengelolaan dependency secara terpisah.

8. MONITORING

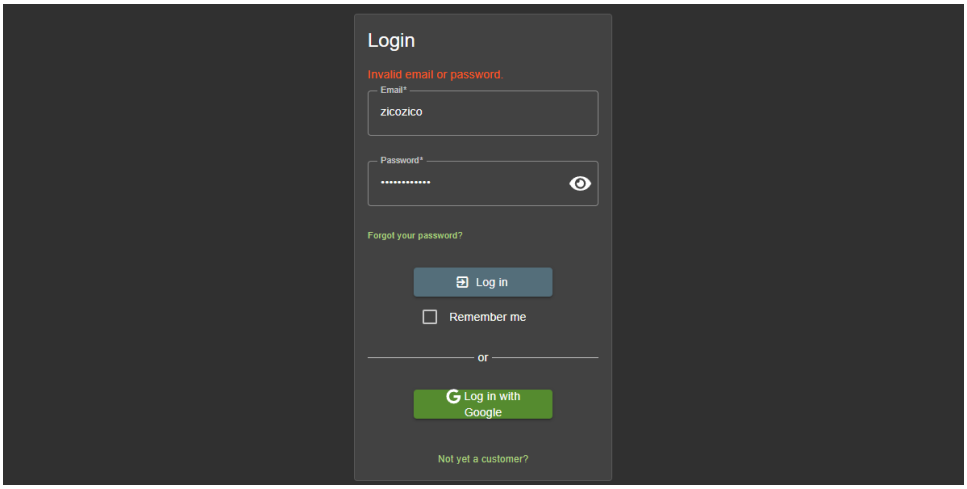
Monitoring dan observabilitas diterapkan untuk memastikan aplikasi OWASP Juice Shop dapat dipantau secara real-time selama beroperasi. Penerapan monitoring bertujuan untuk meningkatkan visibilitas terhadap kondisi sistem, mendeteksi anomali lebih dini, serta mendukung proses analisis ketika terjadi gangguan performa atau insiden keamanan. Pada tahap ini, monitoring difokuskan pada dua aspek utama, yaitu logging dan metrics.

8.1. Implementasi Logging

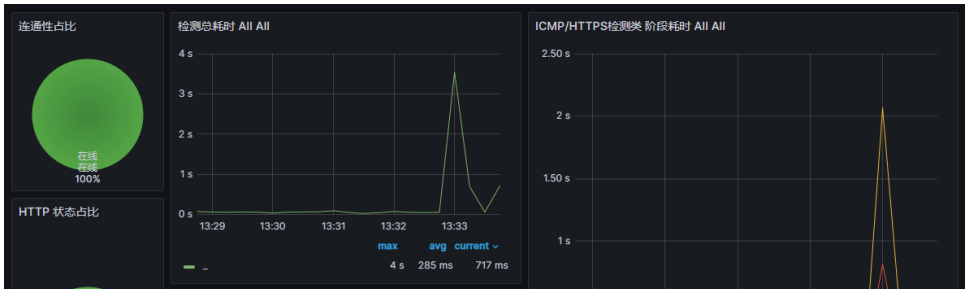
Logging digunakan untuk mencatat aktivitas penting yang terjadi pada aplikasi, seperti percobaan login gagal, error aplikasi, dan aktivitas mencurigakan lainnya. Logging dapat diimplementasikan menggunakan mekanisme log bawaan aplikasi atau diintegrasikan dengan stack logging seperti ELK Stack (Elasticsearch, Logstash, Kibana). Log yang dikumpulkan digunakan untuk melakukan analisis kejadian dan mendukung proses identifikasi insiden keamanan. Jenis log yang diamati antara lain:

1. **Skenario 1: Percobaan login gagal secara berulang**

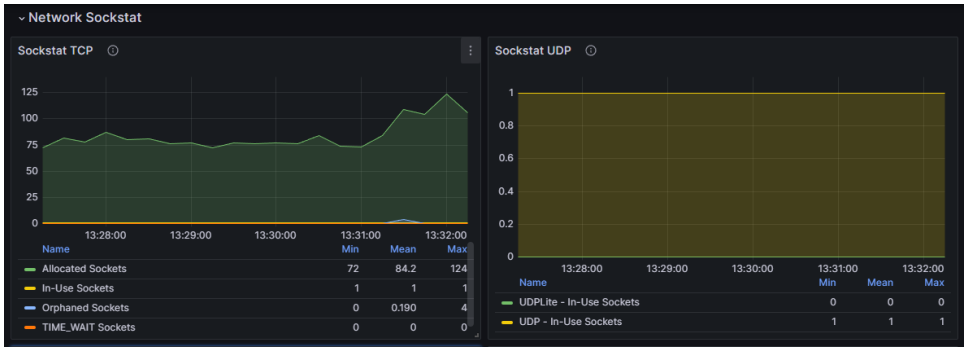
   Pengujian dilakukan dengan mengakses aplikasi OWASP Juice Shop melalui alamat http://127.0.0.1:3000. Selanjutnya, dilakukan beberapa kali percobaan login menggunakan email dan password yang tidak valid untuk mensimulasikan kondisi login gagal berulang. Selama proses tersebut, sistem monitoring dibiarkan berjalan untuk merekam perubahan kondisi aplikasi dan performa layanan secara real-time.



Gambar 8.1 – Form login untuk target percobaan login gagal

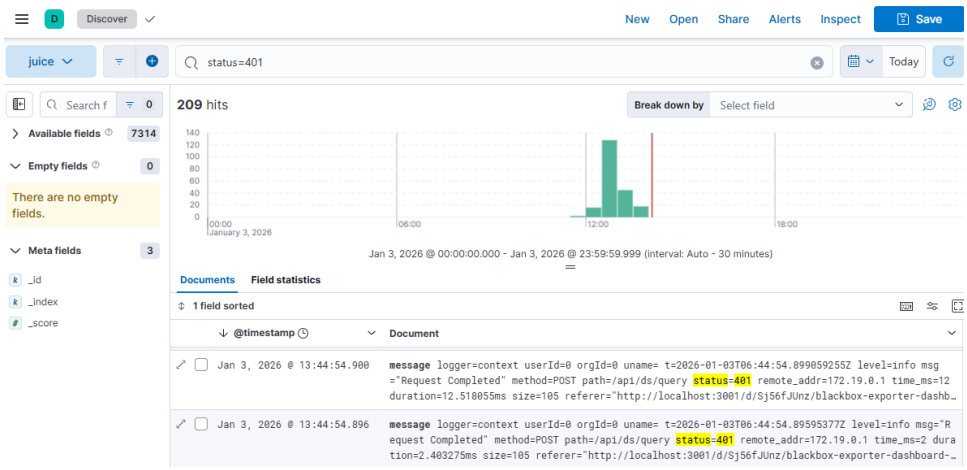


Gambar 8.2 – Dashboard Grafana



Gambar 8.3 – Dashboard Grafana

Selama pengujian berlangsung, dashboard Grafana yang diamati meliputi HTTP Status/Availability dan Response Time (Latency). Hasil monitoring menunjukkan bahwa aplikasi tetap berada dalam kondisi aktif (UP) meskipun terjadi login gagal berulang. Namun, terdapat peningkatan pada waktu respons (latency) yang menandakan adanya kenaikan jumlah permintaan ke aplikasi. Hal ini menunjukkan bahwa aktivitas login yang berulang dapat mempengaruhi performa aplikasi meskipun tidak menyebabkan layanan mengalami downtime.

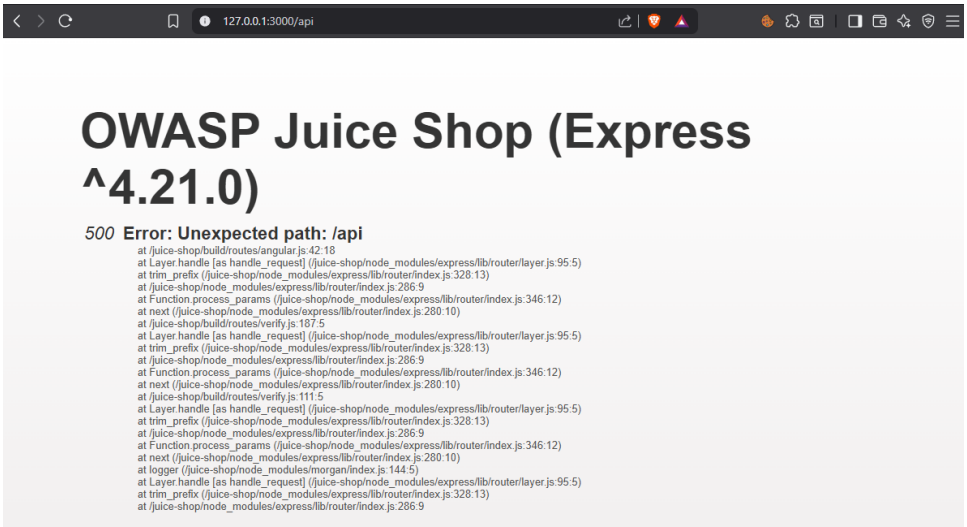


Gambar 8.4 – Elastic Discover

Berdasarkan hasil monitoring log pada Elastic Discover, ditemukan beberapa permintaan dengan status HTTP 401. Status ini menunjukkan bahwa permintaan ditolak karena autentikasi tidak valid atau pengguna belum melakukan login. Meskipun terjadi error autentikasi, aplikasi tetap berjalan normal dan tidak mengalami downtime. Hal ini menunjukkan bahwa mekanisme keamanan aplikasi berfungsi dengan baik dalam membatasi akses tidak sah.

2. **Skenario 2: Error Aplikasi**

Pengujian dilakukan dengan mengakses endpoint yang tidak valid pada aplikasi OWASP Juice Shop, seperti memasukkan URL acak yang tidak tersedia. Tindakan ini bertujuan untuk memicu error pada aplikasi dan mengamati bagaimana sistem merespons kondisi tersebut. Selama pengujian berlangsung, sistem monitoring tetap aktif untuk mencatat setiap perubahan yang terjadi pada aplikasi.



Gambar 8.5 – Test page url error

Gambar 8.6 – Logs “status” pada Elastic

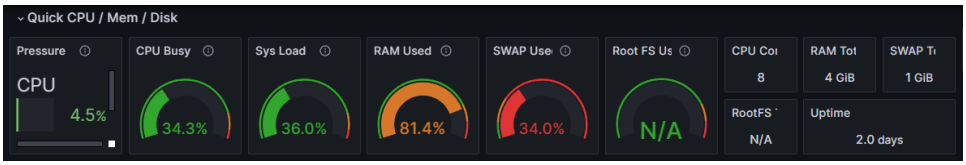
Berdasarkan hasil analisis log pada Elastic Discover, tidak ditemukan event dengan status HTTP 500 selama pengujian berlangsung. Hal ini menunjukkan bahwa

aplikasi mampu menangani kesalahan input atau akses yang tidak valid dengan baik tanpa menyebabkan kegagalan sistem. Error yang muncul dikategorikan sebagai error logika aplikasi dan tidak berdampak pada ketersediaan layanan, sehingga aplikasi tetap berjalan normal dan dapat diakses.

3. **Skenario 3: Lonjakan Resource (CPU & Memory)**

Pengujian dilakukan dengan cara menjalankan aktivitas berulang pada aplikasi OWASP Juice Shop, seperti melakukan refresh halaman secara terus-menerus, mencoba login berkali-kali, serta mengirim permintaan secara bertubi-tubi. Aktivitas ini bertujuan untuk mensimulasikan kondisi beban tinggi pada aplikasi.

Selama pengujian berlangsung, dashboard Grafana yang terhubung dengan Node Exporter diamati untuk melihat perubahan penggunaan resource sistem. Panel yang diperhatikan meliputi CPU Usage, Memory Usage, dan Load Average, baik pada kondisi normal maupun saat beban meningkat.



Gambar 8.7 – Dashboard Grafana

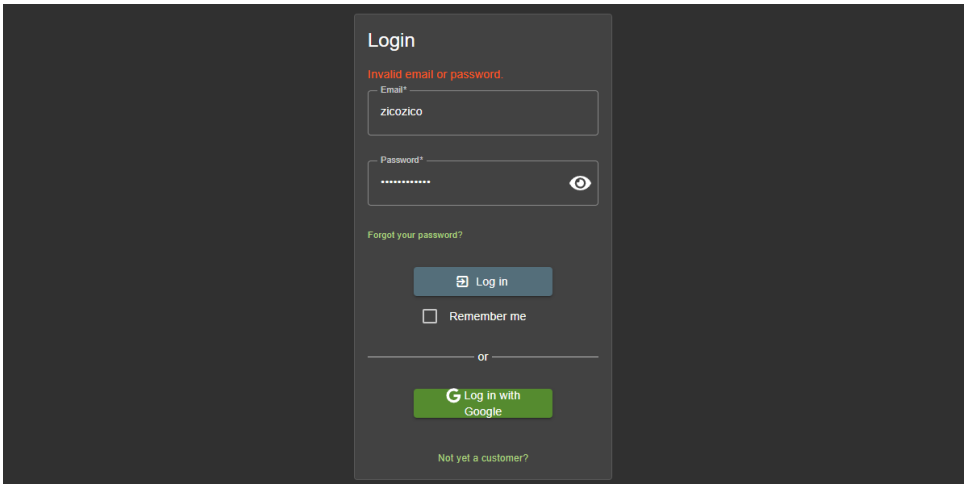
Berdasarkan hasil monitoring, terlihat adanya kenaikan penggunaan CPU dan memori ketika aktivitas aplikasi meningkat. Grafik CPU Usage menunjukkan lonjakan pemakaian prosesor, sementara Memory Usage juga mengalami peningkatan dibandingkan kondisi normal. Selain itu, nilai Load Average meningkat yang menandakan sistem sedang menangani lebih banyak proses secara bersamaan. Hal ini menunjukkan bahwa peningkatan aktivitas aplikasi berdampak langsung pada konsumsi resource sistem.

9. INCIDENT HANDLING

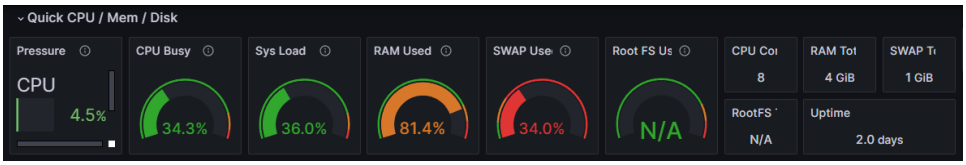
Pada tahap ini dilakukan penanganan insiden keamanan dengan skenario brute force login attack pada aplikasi OWASP Juice Shop. Skenario ini dipilih karena serangan brute force merupakan salah satu ancaman yang paling umum menargetkan mekanisme autentikasi aplikasi web. Penanganan insiden dilakukan mengikuti tahapan standar incident handling yang meliputi identification, containment, eradication, recovery, dan lessons learned.

9.1. Identification

Insiden brute force login diidentifikasi melalui analisis perilaku sistem dan hasil monitoring. Aktivitas login gagal dilakukan secara berulang dalam waktu singkat pada halaman autentikasi OWASP Juice Shop, yang menyebabkan peningkatan jumlah request dan lonjakan penggunaan resource pada container aplikasi. Meskipun tidak ditemukan log autentikasi secara eksplisit pada Kibana maupun docker logs, anomali tersebut dapat diamati melalui dashboard monitoring dan metrik sistem yang menunjukkan pola aktivitas tidak normal dibandingkan kondisi operasional normal.



Gambar 9.1 – Halaman login OWASP Juice Shop sebagai target autentikasi



Gambar 9.2 – Dashboard monitoring yang menunjukkan lonjakan aktivitas sistem

9.2. Containment

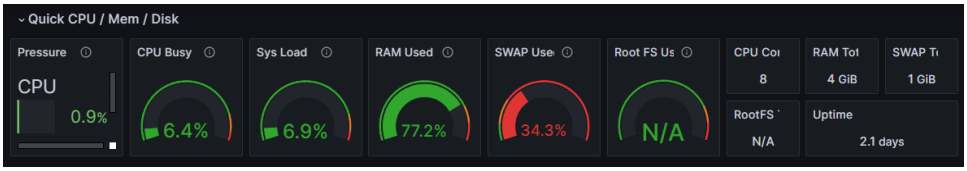
Setelah insiden teridentifikasi, dilakukan langkah containment untuk membatasi dampak serangan. Containment dilakukan dengan membatasi akses terhadap mekanisme login serta meningkatkan pengawasan terhadap aktivitas autentikasi. Langkah ini bertujuan untuk menghentikan percobaan login berulang dan mencegah terjadinya pembebanan sistem yang dapat berdampak pada ketersediaan layanan.

9.3. Eradication

Pada tahap eradication, dilakukan evaluasi terhadap mekanisme autentikasi dan konfigurasi aplikasi untuk memastikan tidak terdapat akun yang berhasil dikompromikan selama simulasi serangan berlangsung. Selain itu, dilakukan peninjauan terhadap konfigurasi keamanan aplikasi sebagai upaya mengurangi potensi terjadinya brute force login di masa mendatang.

9.4. Recovery

Tahap recovery dilakukan dengan memastikan aplikasi kembali beroperasi secara normal setelah aktivitas login berulang dihentikan. Monitoring lanjutan dilakukan untuk memastikan tidak terdapat lonjakan aktivitas mencurigakan dan seluruh komponen sistem berada dalam kondisi stabil.



Gambar 9.3 – Dashboard monitoring setelah sistem kembali normal

9.5. Lessons Learned

Dari insiden brute force login ini dapat disimpulkan bahwa monitoring berbasis metrics memiliki peran yang sangat penting dalam mendeteksi anomali keamanan, terutama pada aplikasi yang tidak menyediakan audit log autentikasi secara detail. Selain itu, incident handling yang terstruktur membantu memastikan bahwa insiden dapat diidentifikasi, dibatasi, dan ditangani secara sistematis meskipun terdapat keterbatasan pada mekanisme logging aplikasi.

9.6. Limitasi Sistem

Dalam proses penanganan insiden ini terdapat keterbatasan pada mekanisme logging aplikasi. Tidak ditemukannya log autentikasi seperti failed login attempt pada Kibana maupun docker logs disebabkan oleh keterbatasan mekanisme logging bawaan OWASP Juice Shop yang tidak mengirimkan audit log autentikasi ke sistem logging terpusat. Kondisi ini merupakan karakteristik dari OWASP Juice Shop sebagai aplikasi pembelajaran dan bukan aplikasi produksi. Oleh karena itu, identifikasi insiden dilakukan berdasarkan indikator anomali pada metrics, perilaku sistem, serta hasil monitoring resource yang menunjukkan pola aktivitas tidak normal.

10. EVALUASI & LESSON LEARNED

Berdasarkan pelaksanaan proyek DevSecOps pada aplikasi OWASP Juice Shop, dapat dievaluasi bahwa penerapan keamanan sejak tahap awal pengembangan memberikan dampak positif dalam mengurangi risiko keamanan aplikasi. Integrasi prinsip Secure SDLC dan threat modeling membantu dalam mengidentifikasi aset penting serta potensi ancaman sejak dini, sehingga proses pengamanan dapat dilakukan secara lebih terarah dan sistematis.

Automasi pengujian keamanan melalui CI/CD pipeline terbukti efektif dalam mendeteksi kerentanan secara konsisten pada setiap perubahan kode. Penggunaan SAST, SCA, container scanning, dan DAST memungkinkan identifikasi kerentanan dilakukan lebih cepat tanpa bergantung pada proses manual. Meskipun tidak semua kerentanan diperbaiki, hasil pengujian tersebut memberikan gambaran yang jelas mengenai postur keamanan aplikasi dan area yang memerlukan perhatian lebih lanjut.

Selain itu, implementasi monitoring dan observabilitas menunjukkan peran penting dalam mendukung keamanan operasional aplikasi. Monitoring berbasis metrics memungkinkan deteksi dini terhadap anomali sistem, seperti lonjakan aktivitas autentikasi dan penggunaan resource yang tidak normal, sehingga proses identifikasi dan penanganan insiden dapat dilakukan dengan lebih cepat. Hal ini menjadi sangat relevan pada aplikasi yang memiliki keterbatasan mekanisme logging.

Proyek ini juga memberikan pembelajaran bahwa keamanan bukan hanya tanggung jawab tim keamanan semata, melainkan merupakan tanggung jawab bersama antara developer, operator, dan security engineer. Pendekatan DevSecOps menuntut kolaborasi lintas peran dalam satu alur kerja yang terintegrasi, sehingga keamanan dapat menjadi bagian inheren dari proses pengembangan dan operasional aplikasi, bukan sekadar tahapan tambahan di akhir siklus pengembangan.

11. KESIMPULAN

Berdasarkan seluruh tahapan yang telah dilaksanakan dalam proyek ini, dapat disimpulkan bahwa penerapan pendekatan DevSecOps pada aplikasi OWASP Juice Shop berhasil memberikan gambaran menyeluruh mengenai pentingnya integrasi keamanan dalam seluruh siklus pengembangan dan operasional aplikasi. Proyek ini mencakup tahapan perencanaan, threat modeling, implementasi CI/CD pipeline, pengujian keamanan otomatis, pengamanan container, monitoring, hingga penanganan insiden keamanan secara sistematis.

Integrasi Secure SDLC dan threat modeling membantu dalam mengidentifikasi aset penting serta potensi ancaman sejak tahap awal, sehingga risiko keamanan dapat dipahami dan dikelola dengan lebih baik. Implementasi CI/CD pipeline yang terintegrasi dengan berbagai tools security testing membuktikan bahwa automasi mampu meningkatkan konsistensi dan efektivitas deteksi kerentanan pada setiap perubahan kode aplikasi.

Selain itu, penerapan Docker hardening dan container scanning menunjukkan bahwa pengamanan lingkungan runtime merupakan bagian penting dari keamanan aplikasi secara keseluruhan. Meskipun masih ditemukan kerentanan pada dependensi aplikasi, proses hardening berhasil memperkecil attack surface dan meningkatkan postur keamanan container. Monitoring dan observabilitas juga berperan penting dalam mendukung keamanan operasional, terutama dalam mendeteksi anomali dan mempercepat proses identifikasi insiden keamanan.

Melalui simulasi incident handling dengan skenario brute force login attack, proyek ini menunjukkan bahwa insiden keamanan dapat ditangani secara terstruktur meskipun terdapat keterbatasan pada mekanisme logging aplikasi. Pendekatan berbasis monitoring dan perilaku sistem terbukti efektif dalam mendukung proses identifikasi, containment, dan recovery insiden. Secara keseluruhan, proyek ini menegaskan bahwa keamanan bukanlah tanggung jawab satu peran tertentu, melainkan hasil dari kolaborasi antara developer, operator, dan security engineer dalam satu alur DevSecOps yang terintegrasi.