LAPORAN PROYEK AKHIR

Pengamanan Aplikasi OWASP Juice Shop Menggunakan Pendekatan DevSecOps

Dosen Pemimbing:

Khamarudin Syarif

Mata Kuliah Development Security Operation (DevSecOps)

Disusun Oleh:

Nama : Surya Aditia Febriyan

NIM : 88032023007

PROGRAM STUDI REKAYASA KEAMANAN SIBER

POLITEKNIK BHAKTI SEMESTA

2025

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan laporan Ujian Akhir Semester (UAS) mata kuliah *Development Security Operation (DevSecOps)* dengan model *Project Based Learning (PBL)* ini dengan baik. Laporan ini disusun sebagai bentuk pertanggungjawaban akademik atas pelaksanaan proyek pengamanan aplikasi *OWASP Juice Shop* menggunakan pendekatan DevSecOps

Laporan ini membahas penerapan DevSecOps secara menyeluruh, mulai dari tahap perencanaan dan perancangan arsitektur sistem, *penerapan Secure Software Development Life Cycle (Secure SDLC)* dan *Threat Modeling*, pembangunan *CI/CD pipeline*, pelaksanaan security testing otomatis, pengamanan container Docker, hingga implementasi monitoring dan penanganan insiden keamanan *(incident handling)*. Seluruh tahapan tersebut dilakukan secara nyata dan terintegrasi sesuai dengan ketentuan yang telah ditetapkan dalam instruksi UAS.

Penyusunan laporan ini bertujuan untuk memberikan pemahaman praktis mengenai bagaimana prinsip keamanan dapat diintegrasikan ke dalam proses pengembangan dan operasional aplikasi secara berkelanjutan. Selain itu, proyek ini diharapkan dapat meningkatkan kesadaran akan pentingnya keamanan aplikasi serta kemampuan mahasiswa dalam mengidentifikasi, menganalisis, dan menangani potensi ancaman keamanan pada sistem berbasis web.

Penulis menyadari bahwa laporan ini masih memiliki keterbatasan dan kekurangan, baik dari segi penyajian maupun kedalaman analisis. Oleh karena itu, penulis mengharapkan kritik dan saran yang membangun demi perbaikan di masa mendatang. Akhir kata, penulis berharap laporan ini dapat memberikan manfaat bagi pembaca serta menjadi referensi dalam penerapan DevSecOps pada pengembangan aplikasi.

DAFTAR ISI

BAB I
PENDAHULUAN

1.1. Latar Belakang

Aplikasi berbasis web saat ini menjadi komponen utama dalam penyediaan layanan digital, baik di lingkungan pendidikan, bisnis, maupun layanan publik. Kompleksitas aplikasi yang semakin tinggi, ditambah dengan ketergantungan pada *third-party libraries*, *containerization*, dan proses *continuous deployment*, menyebabkan permukaan serangan *(attack surface)* terhadap aplikasi juga semakin luas. Kondisi ini menjadikan keamanan aplikasi sebagai aspek krusial yang tidak dapat dipisahkan dari proses pengembangan dan operasional sistem.

Dalam praktik pengembangan perangkat lunak konvensional, aspek keamanan sering kali ditempatkan sebagai tahap akhir atau dilakukan secara terpisah setelah aplikasi selesai dibangun. Pendekatan tersebut berpotensi menyebabkan keterlambatan dalam pendeteksian kerentanan, meningkatnya biaya perbaikan, serta risiko terjadinya insiden keamanan pada sistem yang telah berjalan di lingkungan produksi. Oleh karena itu, diperlukan sebuah pendekatan yang mampu mengintegrasikan keamanan secara menyeluruh dan berkelanjutan sejak tahap awal pengembangan hingga operasional aplikasi.

DevSecOps hadir sebagai pendekatan yang menggabungkan praktik *Development*, *Security*, dan *Operations* dalam satu alur kerja terpadu. Pendekatan ini menekankan otomatisasi, kolaborasi, serta integrasi pengujian keamanan ke dalam *CI/CD pipeline*, sehingga potensi kerentanan dapat dideteksi dan dianalisis lebih dini. Dengan demikian, keamanan tidak lagi menjadi hambatan dalam proses pengembangan, melainkan menjadi bagian inheren dari siklus hidup aplikasi.

Pada proyek Ujian Akhir Semester ini, aplikasi OWASP Juice Shop digunakan sebagai objek studi kasus karena secara sengaja dirancang memiliki berbagai kerentanan keamanan yang merepresentasikan risiko nyata pada aplikasi web modern. Melalui penerapan DevSecOps pada aplikasi tersebut, mahasiswa diharapkan mampu memahami

konsep, tools, dan proses pengamanan aplikasi secara praktis, terukur, dan sesuai dengan standar keamanan yang berlaku.

1.2. Tujuan

Tujuan dari pelaksanaan proyek dan penyusunan laporan ini adalah sebagai berikut:

1. Menerapkan pendekatan DevSecOps dalam pengamanan aplikasi web secara end-to-end menggunakan studi kasus OWASP Juice Shop.
2. Mengimplementasikan *Secure Software Development Life Cycle (Secure SDLC)* dan *Threat Modeling* untuk mengidentifikasi aset, ancaman, serta tingkat risiko keamanan aplikasi.
3. Membangun dan mengonfigurasi *CI/CD pipeline* yang mengintegrasikan *security testing* otomatis, meliputi SAST, SCA, DAST, dan container security.
4. Melakukan analisis terhadap hasil pengujian keamanan untuk mengidentifikasi temuan kritis dan berisiko tinggi.
5. Mengimplementasikan sistem monitoring dan observabilitas untuk mendeteksi anomali serta potensi insiden keamanan.
6. Melakukan simulasi dan penanganan insiden keamanan (incident handling) berdasarkan skenario yang ditentukan.

1.3. Ruang Lingkup

Agar pembahasan dalam proyek ini tetap terarah dan sesuai dengan tujuan yang telah ditetapkan, maka ruang lingkup pekerjaan dibatasi sebagai berikut:

1. Objek yang diamankan adalah aplikasi *OWASP Juice Shop* yang dijalankan pada lingkungan lokal menggunakan Docker container.
2. Implementasi DevSecOps mencakup tahapan *Secure SDLC*, *Threat Modeling*, *CI/CD pipeline*, *security testing*, *container security*, *monitoring*, dan *incident handling*.
3. Tools yang digunakan meliputi GitHub sebagai source code repository, GitHub Actions atau Jenkins sebagai *CI/CD platform*, serta beberapa tools keamanan seperti npm audit, Trivy, dan OWASP ZAP.

4. Monitoring dan observabilitas difokuskan pada pengumpulan log dan metrik untuk mendeteksi skenario tertentu, seperti login gagal berulang, error aplikasi, dan lonjakan penggunaan sumber daya.
5. Perbaikan kerentanan tidak dilakukan secara menyeluruh, namun difokuskan pada analisis temuan serta penerapan kontrol keamanan yang relevan sesuai kebutuhan proyek UAS.

BAB II
ARSITEKTUR SISTEM

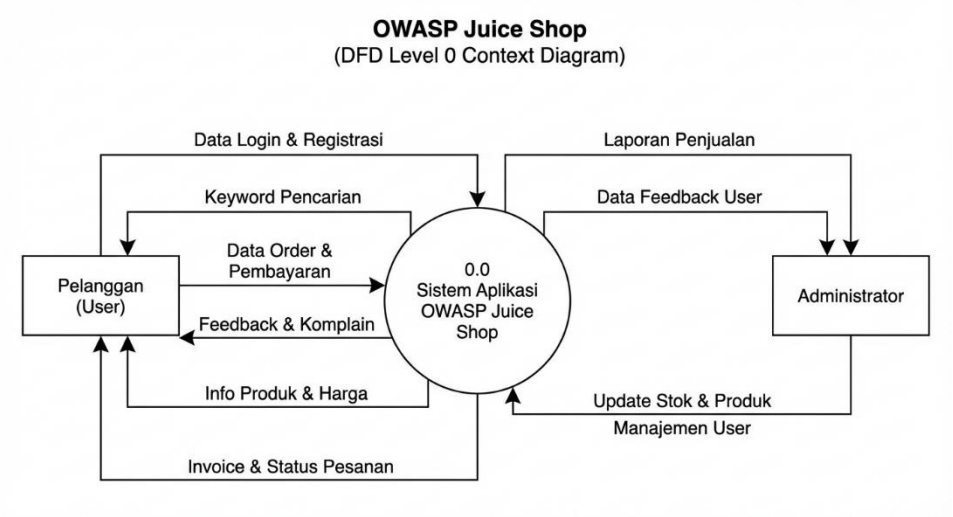
2.1. Gambaran Umum Sistem

OWASP Juice Shop merupakan aplikasi web berbasis *JavaScript* dan *Node.js* yang dirancang untuk mensimulasikan berbagai kerentanan keamanan pada aplikasi web modern. Aplikasi ini menggunakan arsitektur *client–server* di mana pengguna berinteraksi melalui antarmuka web *(frontend)*, sementara proses bisnis, autentikasi, dan pengelolaan data dilakukan pada sisi *backend* yang terhubung dengan basis data.

Dalam proyek ini, OWASP Juice Shop dijalankan pada lingkungan lokal menggunakan teknologi *containerization* Docker. Pendekatan ini dipilih untuk memastikan konsistensi lingkungan eksekusi, mempermudah proses *deployment*, serta memungkinkan penerapan pengamanan container dan integrasi dengan *CI/CD pipeline*. Seluruh aktivitas pengembangan, pengujian, dan pengamanan aplikasi dilakukan dengan mengacu pada prinsip DevSecOps, di mana keamanan diintegrasikan sejak tahap awal hingga operasional sistem.

2.2. Data Flow Diagram (DFD) Aplikasi

2.2.1. Diagram Konteks Sistem (Level 0)



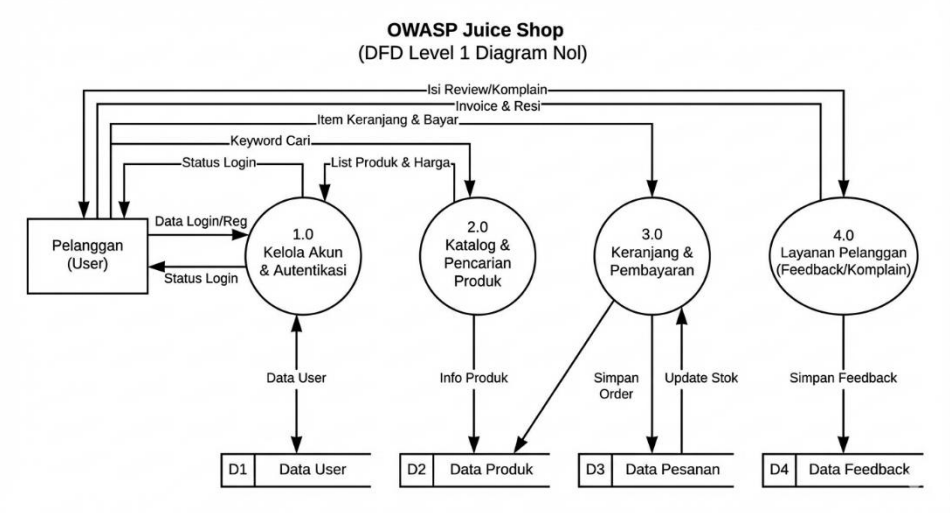
Bagian-bagian dari diagram tersebut adalah:

1. External Entity : Pelanggan (User) & Administrator.
2. Proses : Sistem Aplikasi OWASP Juice Shop (Proses 0.0).

Sedangkan proses yang terjadi sebagai berikut:

1. Pelanggan melakukan registrasi akun baru atau login ke dalam sistem menggunakan email dan password.
2. Sistem memvalidasi kredensial tersebut; jika valid, sistem memberikan akses masuk (sesi login) kepada Pelanggan.
3. Pelanggan memasukkan kata kunci pencarian *(keyword)* untuk mencari produk jus atau *merchandise*.\
4. Sistem merespon dengan menampilkan daftar produk, deskripsi detail, harga, dan ketersediaan stok kepada Pelanggan.
5. Pelanggan memilih produk, memasukkannya ke keranjang, dan melakukan proses *checkout* (mengirim data alamat dan pembayaran).
6. Sistem memproses pesanan, menerbitkan *invoice* (tagihan), dan memberikan nomor resi/status pengiriman kepada Pelanggan.
7. Pelanggan mengirimkan ulasan *(feedback)* atau mengajukan komplain terkait layanan.
8. Administrator masuk ke sistem untuk melakukan manajemen stok barang dan mengelola data pengguna.
9. Sistem memberikan laporan penjualan dan meneruskan data feedback atau komplain dari Pelanggan kepada Administrator untuk ditinjau.

2.2.2. Diagram Nol (Level 1)



Dalam diagram nol tersebut terdapat beberapa proses yang terjadi. Proses tersebut adalah:

Proses 1.0: Kelola Akun & Autentikasi

1. Pelanggan memasukkan data diri (email, password, pertanyaan keamanan) untuk registrasi atau login.
2. Sistem memeriksa kecocokan data tersebut dengan database pada Data Store Users (D1).
3. Jika data cocok, sistem membuat token otentikasi dan memberikannya kepada Pelanggan sebagai kunci akses untuk berbelanja.

Proses 2.0: Penelusuran & Katalog Produk

1. Pelanggan menjelajahi halaman utama atau menggunakan kotak pencarian *(search bar)* untuk menemukan produk.
2. Sistem menerima kata kunci tersebut, lalu mengambil data detail (Gambar, Deskripsi, Harga, Ulasan Produk) dari Data Store Products (D2).
3. Sistem menampilkan hasil pencarian atau katalog produk tersebut kembali ke layar Pelanggan.

Proses 3.0: Transaksi & Checkout

1. Pelanggan memilih barang untuk dimasukkan ke keranjang, memilih alamat pengiriman, dan metode pembayaran.

2. Sistem menghitung total biaya, menyimpan detail pesanan baru ke dalam Data Store Orders (D3).
3. Secara otomatis, sistem juga akan mengurangi jumlah stok barang yang terjual pada Data Store Products (D2).
4. Sistem mengirimkan Konfirmasi Pesanan dan ID Pelacakan (Tracking ID) kepada Pelanggan.

Proses 4.0: Layanan Pelanggan (Feedback & Komplain)

1. Pelanggan mengakses menu *"Contact Us"* untuk memberi rating/komentar, atau menu *"Complain"* untuk mengunggah bukti keluhan (file upload).
2. Sistem menerima input teks dan file tersebut, lalu menyimpannya secara permanen ke dalam Data Store Feedbacks (D4).
3. Sistem memberikan pesan konfirmasi *(pop-up)* kepada Pelanggan bahwa masukan mereka telah diterima.

BAB III

SECURE SDLC & THREAT MODELING

3.1. Identifikasi Aset Aplikasi

Identifikasi aset merupakan langkah awal yang krusial dalam proses pengamanan sistem, karena aset yang bernilai tinggi menjadi target utama bagi penyerang. Pada aplikasi OWASP Juice Shop, aset-aset penting berikut diidentifikasi berdasarkan dampaknya terhadap kerahasiaan *(confidentiality)*, integritas *(integrity)*, dan ketersediaan *(availability)* sistem:

No	Aset Penting	Deskripsi	Dampak Jika Terganggu
1	Akun Pengguna	Data akun pengguna yang digunakan untuk autentikasi dan akses aplikasi	Kebocoran akun, pengambilalihan identitas
2	Mekanisme Autentikasi	Proses login, manajemen sesi, dan token autentikasi	Akses tidak sah ke sistem
3	Basis Data Aplikasi	Menyimpan data pengguna, produk, dan transaksi	Kebocoran atau manipulasi data
4	API Endpoint	Endpoint backend yang melayani permintaan aplikasi	Penyalahgunaan API, manipulasi data
5	Docker Image Aplikasi	Image container yang digunakan untuk menjalankan aplikasi	Penyebaran image berbahaya atau rentan
6	CI/CD Pipeline	Proses otomatis build, test, dan deployment	Penyisipan kode berbahaya, supply chain attack

3.2. Analisis STRIDE

Metode STRIDE digunakan untuk mengidentifikasi jenis ancaman yang mungkin terjadi terhadap aset-aset penting dalam sistem. STRIDE mengelompokkan ancaman ke dalam enam kategori utama, yaitu *Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service,* dan *Elevation of Privilege*.

Kategori STRIDE	Aset Terdampak	Deskripsi Ancaman
Spoofing	Akun Pengguna, Autentikasi	Penyerang memalsukan identitas pengguna untuk mendapatkan akses ke sistem
Tampering	Basis Data, API Endpoint	Data aplikasi dimodifikasi secara tidak sah melalui celah keamanan
Repudiation	Sistem Log	Pengguna atau penyerang menyangkal aktivitas karena logging tidak memadai
Information Disclosure	Basis Data, API Endpoint	Kebocoran data sensitif seperti kredensial atau data pengguna
Denial of Service	Aplikasi, API Endpoint	Aplikasi tidak dapat diakses akibat serangan berulang atau lonjakan trafik
Elevation of Privilege	Akun Pengguna	Penyerang memperoleh hak akses lebih tinggi dari yang seharusnya

3.3. Analisis DREAD

Metode DREAD digunakan untuk menilai tingkat risiko dari setiap ancaman yang telah diidentifikasi. Penilaian dilakukan berdasarkan lima parameter, yaitu *Damage Potential* (D), *Reproducibility* (R), *Exploitability* (E), *Affected Users* (A), dan *Discoverability* (D) dengan skala nilai 1–10. Nilai total diperoleh dari rata-rata seluruh parameter dan digunakan untuk menentukan tingkat risiko.

No	Ancaman	D	R	E	A	D	Total/5	Level
1	Brute Force Login	7	8	7	6	8	7,2	Tinggi

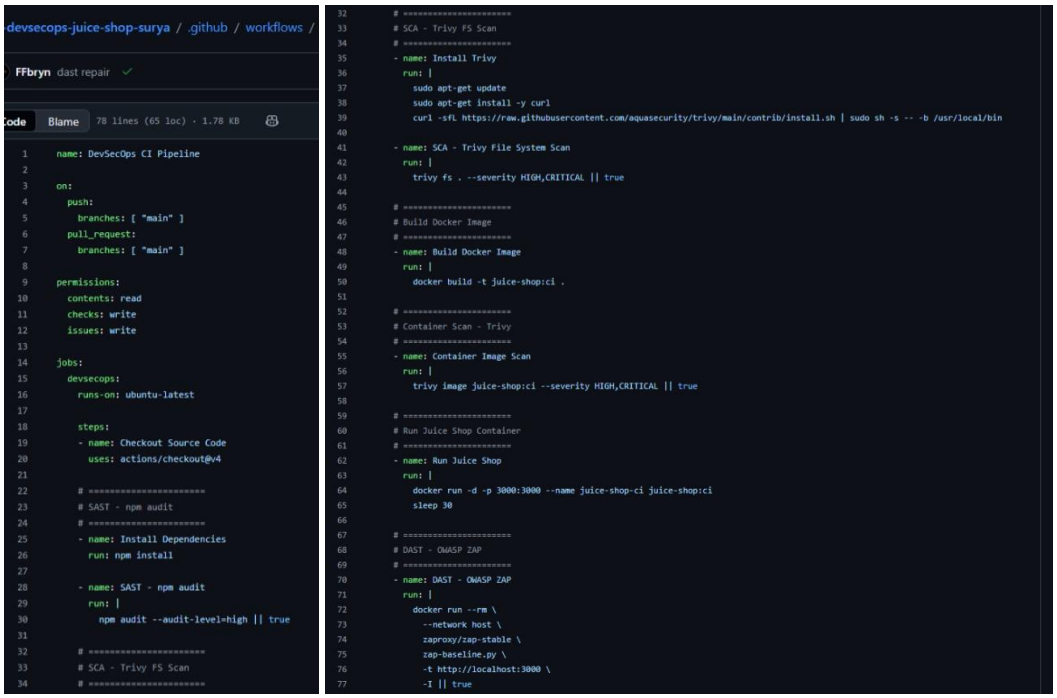
2	SQL Injection	9	8	8	8	7	8	Tinggi
3	Cross-Site Scripting (XSS)	6	7	7	6	7	6,6	Sedang
4	API Abuse	7	6	6	6	6	6,2	Sedang
5	Denial of Service (DoS)	8	7	6	7	6	6,8	Sedang

BAB IV

CI/CD PIPELINE & SECURITY TESTING

4.1. File Pipeline – GitHub Workflow

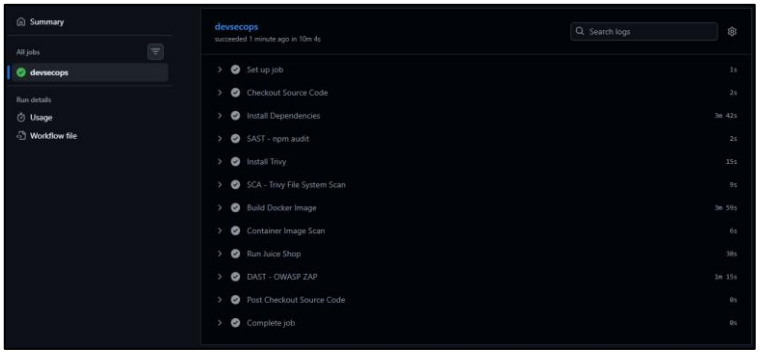
Pipeline CI/CD DevSecOps diimplementasikan menggunakan GitHub Actions yang terdiri dari tahapan SAST menggunakan npm audit, Software Composition Analysis(SCA) dan container scanning menggunakan Trivy, build Docker image, serta Dynamic Application Security Testing menggunakan OWASP ZAP. Seluruh tahapan dijalankan secara otomatis pada setiap push ke branch main. Pipeline tersebut berisi code berikut:



```
devsecops-juice-shop-surya / .github / workflows /
FFbryn dast repair
Code  Blame  78 lines (65 loc) · 1.78 KB
```

```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

    # ====================
    # SCA - Trivy FS Scan
    # ====================
    # ====================
    # 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
```

Dari source code tersebut pipeline berhasil dijalankankan menggunakan github actions:



4.2. SAST - npm audit

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

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

Proses dimulai dengan perintah `npm install` untuk menyiapkan lingkungan, kemudian dilanjutkan dengan perintah `npm audit --audit-level=high` yang berfungsi memindai pustaka *(library)* pihak ketiga guna mencari celah keamanan dengan tingkat keparahan minimal "High" (Tinggi). Penggunaan operator `|| true` di akhir perintah bertujuan agar pipeline tetap dianggap berhasil dan tidak berhenti di tengah jalan meskipun ditemukan kerentanan, sehingga tim pengembang tetap dapat meninjau laporan keamanan tanpa menghambat proses build atau deployment lainnya.

Salah satu hasil yang didapat sebagai berikut:

```
ws  7.0.0 - 7.5.9
Severity: high
ws affected by a DoS when handling a request with many HTTP headers - https://github.com/advisories/GHSA-3h5v-q93c-6h6q
fix available via `npm audit fix --force`
Will install socket.io@4.8.3, which is a breaking change
node_modules/engine.io-client/node_modules/ws
node_modules/engine.io/node_modules/ws

45 vulnerabilities (1 low, 18 moderate, 19 high, 7 critical)
```

Berdasarkan hasil pemindaian, terdeteksi celah keamanan tingkat tinggi pada paket *websocket* (ws) versi 7.0.0 - 7.5.9 yang membuat aplikasi rentan terhadap serangan *Denial of Service* (DoS) ketika menangani permintaan dengan jumlah header HTTP yang sangat banyak.

Secara keseluruhan, laporan audit ini menunjukkan kondisi keamanan proyek yang cukup kritis dengan total 45 celah keamanan yang terdeteksi, yang terdiri dari 1 level rendah, 18 sedang, 19 tinggi, dan 7 sangat kritis.

4.3. SCA - Trivy FS Scan

```
# ======================
# 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
```

Tahap pertama, *Install Trivy*, berfungsi untuk memperbarui repositori sistem dan menginstal binary Trivy langsung dari sumber resminya menggunakan *curl*. Tahap kedua, *SCA - Trivy File System Scan*, menjalankan perintah *trivy fs .* untuk memindai seluruh direktori proyek dengan filter tingkat keparahan *--severity HIGH,CRITICAL*, yang berarti hanya temuan berbahaya saja yang akan ditampilkan. Penggunaan operator || *true* memastikan pipeline tetap berjalan sukses meskipun ditemukan celah keamanan, sehingga tim dapat meninjau laporan tanpa menghentikan proses pengembangan lainnya.

Hasil dari SCA tersebut adalah:

```
Report Summary

┌──────────────────────────────────────────────────────────────────────────────┬──────┬─────────────────┬─────────┐
│                                    Target                                    │ Type │ Vulnerabilities │ Secrets │
├──────────────────────────────────────────────────────────────────────────────┼──────┼─────────────────┼─────────┤
│ build/lib/insecurity.js                                                      │ text │        -        │    1    │
├──────────────────────────────────────────────────────────────────────────────┼──────┼─────────────────┼─────────┤
│ frontend/src/app/app.guard.spec.ts                                           │ text │        -        │    0    │
├──────────────────────────────────────────────────────────────────────────────┼──────┼─────────────────┼─────────┤
│ frontend/src/app/last-login-ip/last-login-ip.component.spec.ts               │ text │        -        │    0    │
├──────────────────────────────────────────────────────────────────────────────┼──────┼─────────────────┼─────────┤
│ lib/insecurity.ts                                                            │ text │        -        │    1    │
└──────────────────────────────────────────────────────────────────────────────┴──────┴─────────────────┴─────────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)


build/lib/insecurity.js (secrets)
=================================
Total: 1 (HIGH: 1, CRITICAL: 0)

HIGH: AsymmetricPrivateKey (private-key)
════════════════════════════════════════
Asymmetric Private Key
──────────────────────────────────────────
 build/lib/insecurity.js:47 (offset: 2835 bytes)
──────────────────────────────────────────
   45   const z85 = __importStar(require("z85"));
   46   exports.publicKey = node_fs_1.default ? node_fs_1.default.readFileSync('encryptionkeys/jwt.pub', 'ut
   47 [ -----BEGIN RSA PRIVATE KEY-----
********************************************************************************************************************************
********************************************************************************************************************************
********************************************************************************************************************************
********************************************************************************************************************************
********************************************************************************************************************************
********************************************************************************************************************************
********************************************************************************-----END RSA PRIVATE
   48   const hash = (data) => node_crypto_1.default.createHash('md5').update(data).digest('hex');
──────────────────────────────────────────
```

Hasil log pemindaian menunjukkan ditemukannya ancaman keamanan tingkat tinggi berupa kebocoran data sensitif, yaitu kunci privat *(Asymmetric Private Key)* yang tertanam langsung di dalam kode. Temuan ini terdeteksi pada file *build/lib/insecurity.js* di baris ke-47 dan juga pada *file lib/insecurity.ts*, di mana sistem mengenali adanya header -----BEGIN RSA PRIVATE KEY-----. Meskipun pemindaian tidak menemukan celah keamanan pada pustaka pihak ketiga (Vulnerabilities: 0), keberadaan kunci enkripsi yang terekspos di dalam repositori merupakan risiko keamanan HIGH karena dapat disalahgunakan oleh pihak tidak berwenang untuk memecahkan enkripsi atau mengakses server secara ilegal.

4.4. Container Scan – Trivy

```
# =====================
# 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
```

Tahap pertama, Build Docker Image, menjalankan perintah docker *build -t juice-shop:ci* . untuk membuat sebuah container image dari source code aplikasi. Tahap selanjutnya, Container Image Scan, menggunakan alat keamanan Trivy untuk memindai image yang baru saja dibuat tersebut guna mendeteksi celah keamanan sistem operasi maupun pustaka di dalamnya. Sama seperti tahap sebelumnya, filter *--severity HIGH,CRITICAL* digunakan untuk hanya menampilkan temuan yang berbahaya, dan operator || true disematkan agar proses integrasi tetap berjalan meskipun ditemukan kerentanan.

Hasil yang diperoleh dari container scan adalah:

```
Node.js (node-pkg)
==================
Total: 30 (HIGH: 22, CRITICAL: 8)
```

```
/juice-shop/build/lib/insecurity.js (secrets)
==============================================
Total: 1 (HIGH: 1, CRITICAL: 0)
```

Berdasarkan hasil scan container menggunakan Trivy, ditemukan 30 kerentanan pada file Node.js yang berupa 22 HIGH dan 8 Critical. Sedangkan pada file insecurity.js ditemukan 1 kerentanan dengan status HIGH.

4.5. DAST - OWASP ZAP

```
# =====================
# 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
```

Perintah ini menggunakan *zaproxy/zap-stable* untuk mengeksekusi skrip *zap-baseline.py* yang bertujuan memindai kerentanan dasar pada aplikasi yang berjalan di *http://localhost:3000*. Penggunaan argumen *--network host* memungkinkan kontainer Docker mengakses jaringan lokal penyalin (host), sementara parameter *-I* serta perintah || *true* di akhir baris berfungsi agar alur kerja pipeline tetap berlanjut (tidak berhenti/gagal) meskipun ditemukan celah keamanan selama proses pemindaian.

Hasil dari pemindaian tersebut contohnya:

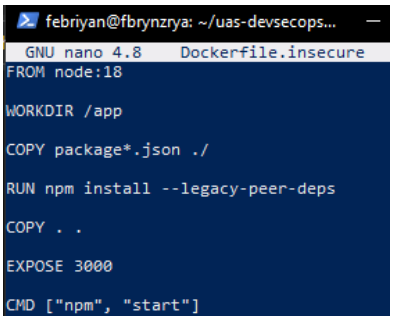
```
216  WARN-NEW: Insufficient Site Isolation Against Spectre Vulnerability [90004] x 10
217          http://localhost:3000 (200 OK)
218          http://localhost:3000/ (200 OK)
219          http://localhost:3000/ftp (200 OK)
220          http://localhost:3000/juice-shop/node_modules/express/lib/router/index.js:328:13 (200 OK)
221          http://localhost:3000/sitemap.xml (200 OK)
222  FAIL-NEW: 0     FAIL-INPROG: 0  WARN-NEW: 10    WARN-INPROG: 0  INFO: 0 IGNORE: 0       PASS: 57
```

Salah satu hasil pemindaian menunjukkan bahwa OWASP ZAP berhasil memeriksa aplikasi dan menemukan 10 peringatan baru (WARN-NEW) terkait celah keamanan *"Insufficient Site Isolation Against Spectre Vulnerability"* dengan kode identitas [90004]. Peringatan ini muncul pada berbagai endpoint seperti halaman utama, direktori /ftp, hingga file */sitemap.xml*, yang mengindikasikan bahwa aplikasi belum mengonfigurasi header *HTTP Cross-Origin* secara memadai untuk melindungi data dari potensi serangan tingkat prosesor.\

Secara keseluruhan, pemindaian mencatat 57 pengecekan yang berhasil lulus (PASS) dan 10 terdeteksi sebagai WARN-NEW, sehingga status akhirnya dianggap aman untuk melanjutkan proses integrasi meskipun peringatan tersebut perlu ditinjau lebih lanjut.

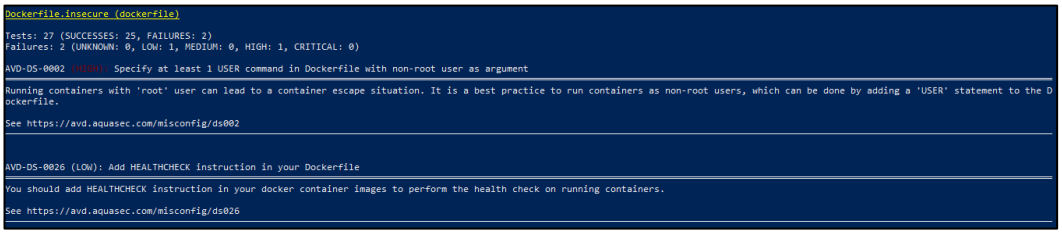
BAB V
DOCKER & IAC SECURITY

5.1. Dockerfile Awal (Insecure)



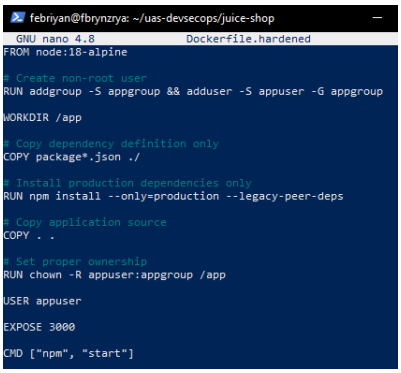
Dockerfile.insecure yang menggunakan base image *node:18* cenderung berukuran besar dan memiliki permukaan serangan yang lebih luas. Berkas ini menginstruksikan Docker untuk menyalin kode sumber dan menginstal dependensi Node.js tanpa adanya pembatasan hak akses pengguna. Hal ini mengakibatkan aplikasi dijalankan dengan hak akses root di dalam kontainer, yang merupakan praktik berbahaya karena jika terjadi eksploitasi, penyerang berpotensi mendapatkan kontrol penuh atas kontainer dan mencoba melakukan container escape ke sistem host.

5.2. Hasil Scan Trivy



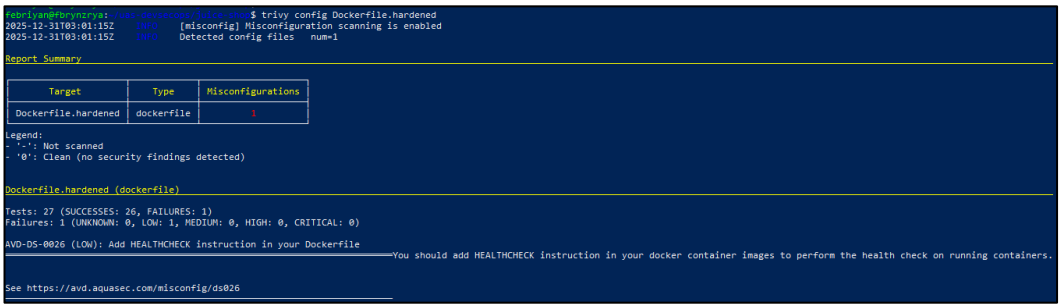
Hasil pemindaian keamanan Dockerfile.insecure menunjukkan adanya dua kesalahan konfigurasi serius dari total 27 pengujian. Temuan dengan tingkat keparahan HIGH (AVD-DS-0002) secara eksplisit memperingatkan bahwa tidak ada instruksi USER non-root yang ditentukan, sehingga kontainer berjalan dengan hak akses administratif. Selain itu, terdapat temuan tingkat LOW (AVD-DS-0026) karena tidak adanya instruksi HEALTHCHECK untuk memantau status kesehatan kontainer secara otomatis, yang mengonfirmasi bahwa konfigurasi ini memiliki celah keamanan yang signifikan.

5.3. Dockerfile Akhir (Hardened)



Dockerfile.hardened menunjukkan base image node:18-alpine yang jauh lebih ringkas dan memiliki minimal kerentanan bawaan. Perubahan kunci meliputi pembuatan grup dan pengguna baru *(appuser)* agar aplikasi tidak berjalan sebagai root, serta penggunaan perintah *chow*n untuk membatasi kepemilikan berkas hanya pada pengguna tersebut. Selain itu, proses instalasi dependensi dibatasi hanya untuk modul produksi *(--only=production)*, yang secara efektif mengurangi ukuran citra dan meminimalkan potensi pintu masuk serangan dari tools pengembangan yang tidak diperlukan di lingkungan runtime.

Setelah dilakukan scan ulang dengan Trivy maka menghasilkan:

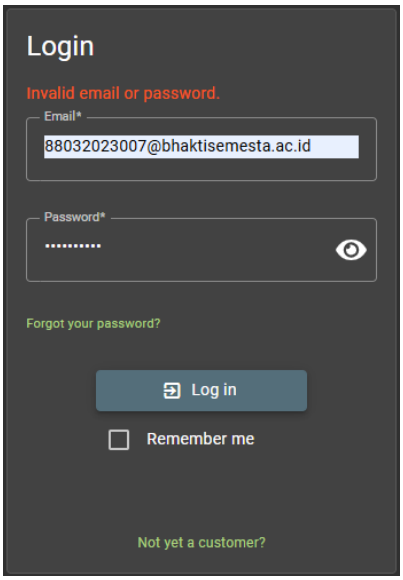


Hasil pemindaian Trivy terhadap Dockerfile.hardened menunjukkan peningkatan keamanan yang drastis di mana kerentanan tingkat HIGH sebelumnya telah berhasil dihilangkan. Dari total 27 tes, 26 di antaranya dinyatakan lulus (SUCCESS), dan hanya menyisakan satu temuan tingkat LOW terkait instruksi HEALTHCHECK yang belum ditambahkan. Status ini menandakan bahwa citra kontainer sekarang jauh lebih aman untuk digunakan di lingkungan produksi karena telah menerapkan prinsip hak akses terendah *(least privilege)* dan menggunakan fondasi sistem operasi yang lebih terlindungi.

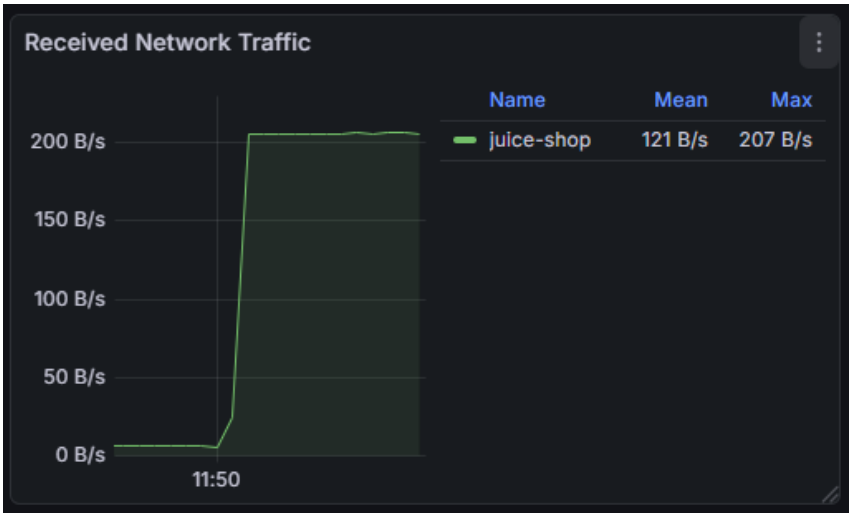
BAB VI

MONITORING & OBSERVABILITY

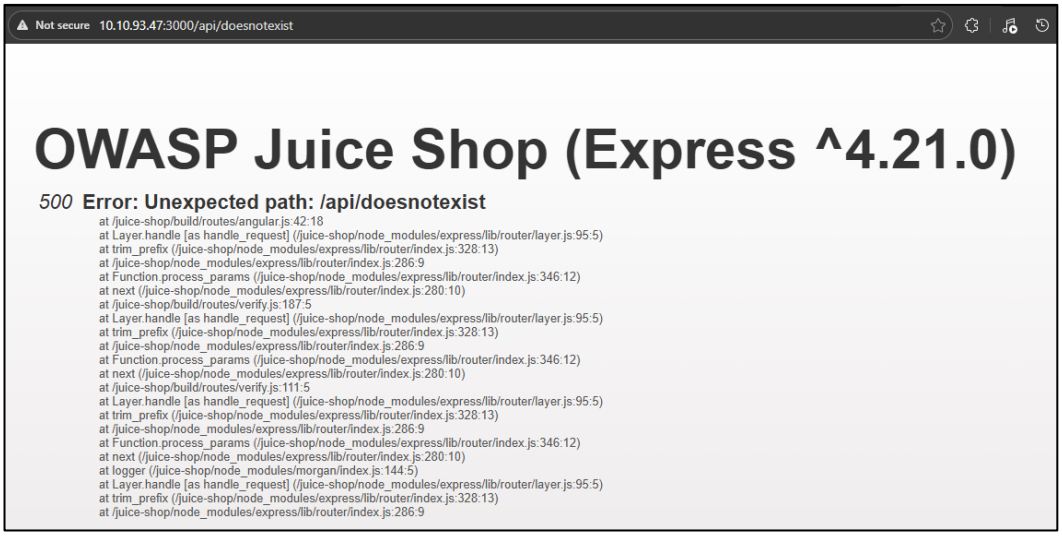
6.1. Login Gagal Berulang



Skenario login gagal dilakukan melalui alamat http://10.10.93.47:3000/#/login secara berulang kali. Hal ini tidak memunculkan log khusus karena aplikasi tidak didesain memiliki app.log. Namun akan terjadi network trafict yang meningkat karena banyaknya permintaan. Meningkatnya network traffic dapat dilihat melalui grafana berikut:



6.2. Error Aplikasi



Simulasi error dengan akses URL http://10.10.93.47:3000/api/doesnotexist menghasilkan respons 500 Internal Server Error.

```
febriyan@fbrynzrya:~/oas-devsecops/juice-shop$ docker logs juice-shop | grep -i error
Error: Unexpected path: /api/doesnotexist
    at /juice-shop/build/routes/angular.js:42:18
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at /juice-shop/build/routes/verify.js:187:5
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at /juice-shop/build/routes/verify.js:111:5
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)
    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at logger (/juice-shop/node_modules/morgan/index.js:144:5)
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
```

Sistem mencatat error yang sama secara real-time dengan pesan "Error: Unexpected path: /api/doesnotexist". Log ini mengonfirmasi bahwa aplikasi mengalami kegagalan saat mencoba memproses rute (path) yang tidak terdefinisi atau tidak dikenali dalam logika routing Node.js/Express. Stack trace yang muncul menunjukkan bahwa permintaan tersebut melewati berbagai lapisan middleware, mulai dari logger morgan hingga modul verify.js dan angular.js, namun karena aplikasi tidak memiliki penanganan error (error handling) yang tepat untuk rute yang salah, sistem langsung melempar pengecualian (exception) yang menyebabkan error 500 tersebut muncul di sisi server maupun klien.

6.3. Peningkatan Resource

Selain Network Traffic yang meingkat, CPU dan RAM juga meningkat karena percobaan login yang terus berulang. Penggunaan CPU mengalami kenaikan dari kondisi idle hingga mencapai puncak di angka 2.99%, sementara penggunaan memori melonjak dan stabil di angka 133 MiB, yang mencerminkan beban kerja mesin dalam memproses setiap permintaan otentikasi serta pengecekan kredensial di database. Secara simultan, grafik trafik jaringan mencatat lonjakan data masuk (Received) hingga 1.01 kB/s dan data keluar (Sent) hingga 838 B/s, yang mengonfirmasi adanya aliran paket data terus-menerus akibat upaya login yang intensif, sehingga memaksa sistem untuk beroperasi pada level penggunaan resource yang lebih tinggi dibandingkan kondisi normalnya.

BAB VII
INCIDENT HANDLING (XSS ATTACK)

7.1. Incident Overview (Ringkasan Insiden)

Jenis Insiden : Cross-Site Scripting (DOM-Based XSS)

Aplikasi : OWASP Juice Shop

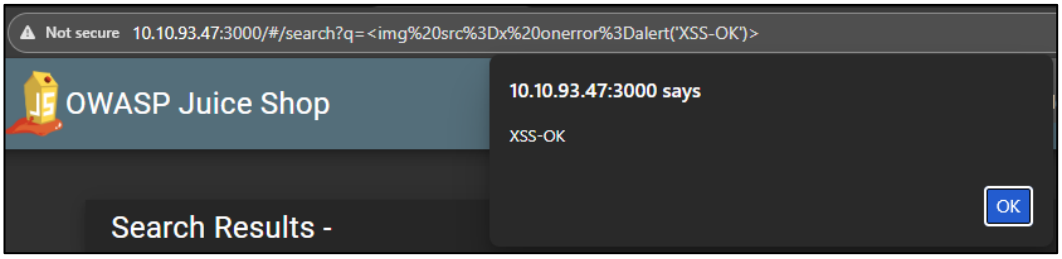
Lokasi Kerentanan : Fitur pencarian (Search)

Payload Contoh : <img src=x onerror=alert('XSS')>

Dampak Awal : Eksekusi JavaScript tidak sah pada browser pengguna.

7.2. Identification

Administrator sistem melakukan pengujian keamanan aplikasi OWASP Juice Shop dengan mensimulasikan serangan Cross-Site Scripting (XSS) pada fitur pencarian (search). Pengujian ini dilakukan sebagai bagian dari aktivitas security testing untuk mengidentifikasi potensi kelemahan pada mekanisme validasi input sisi klien. Admin memasukkan payload XSS berupa kode HTML dan JavaScript ke dalam parameter pencarian, yang kemudian diproses dan dirender langsung oleh Document Object Model (DOM) aplikasi.



Hasil pengujian menunjukkan bahwa payload berhasil dieksekusi pada browser, ditandai dengan munculnya pop-up JavaScript, yang mengindikasikan adanya kerentanan DOM-Based XSS pada fitur search. Temuan ini menegaskan bahwa aplikasi belum sepenuhnya melakukan sanitasi input terhadap data yang berasal dari pengguna.

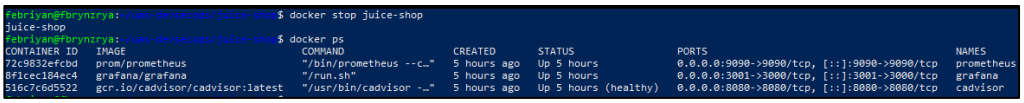
7.3. Containment (Isolasi)

Pada tahap containment, layanan aplikasi OWASP Juice Shop dihentikan sementara dengan menghentikan container Docker. Tindakan ini dilakukan untuk mencegah eksekusi payload Cross-Site Scripting (XSS) yang tersimpan serta menghindari dampak lebih lanjut terhadap pengguna lain. Penghentian layanan memberikan waktu bagi tim untuk melakukan analisis dan perbaikan tanpa risiko eksploitasi lanjutan.

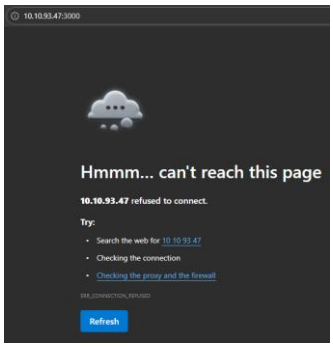
Langkah yang dilakukan:

1. Hentikan aplikasi melalui docker:

   ```
   docker stop juice-shop
   ```



2. Cek di browser apakah aplikasi berhenti:



7.4. Eradication (Perbaikan)

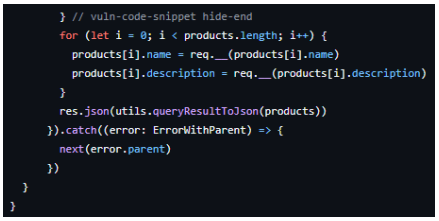
Perbaikan dilakukan pada fitur pencarian (search) dengan menambahkan mekanisme sanitasi input untuk mencegah eksekusi script berbahaya. Dengan pendekatan ini, payload XSS yang sebelumnya dapat dieksekusi kini hanya diperlakukan sebagai teks biasa, sehingga akar penyebab insiden berhasil dieliminasi dan risiko terulangnya serangan serupa dapat dicegah.

Proses Perbaikan tersebut berupa:

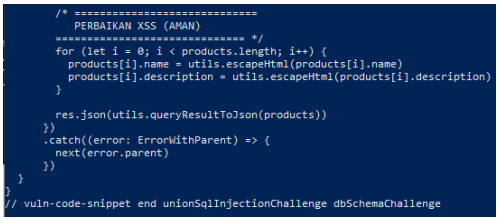
1. Cari lokasi konfigurasi search:

   ```
   grep -R "search" routes -n
   ```

2. Perbaiki bagian berikut:



Sebelum



Sesudah

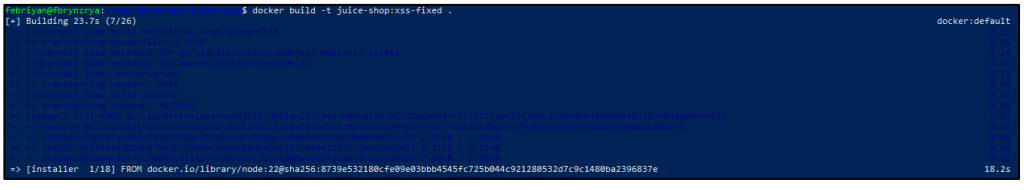
Di dalam perulangan sebelum diperbaiki (kiri), aplikasi menggunakan fungsi ***req.__()*** pada properti nama dan deskripsi produk. Fungsi ini biasanya digunakan untuk internasionalisasi atau penerjemahan teks, namun tidak melakukan pembersihan (sanitization) terhadap karakter-karakter HTML yang berbahaya. Jika data produk tersebut mengandung skrip berbahaya, skrip tersebut akan dikirimkan langsung ke sisi klien dan dapat dieksekusi oleh peramban pengguna, yang berisiko pada pencurian data sesi atau serangan lainnya.

Perbaikan keamanan dengan menerapkan fungsi ***utils.escapeHtml()***. Fungsi ini bekerja dengan melakukan output encoding, yaitu mengubah karakter khusus HTML seperti “ < ” dan “ > ” menjadi format entitas yang aman agar tidak diterjemahkan sebagai kode oleh peramban. Dengan cara ini, meskipun data produk mengandung input berbahaya, peramban hanya akan menampilkannya sebagai teks biasa dan bukan sebagai instruksi yang dapat dijalankan. Langkah ini secara efektif menutup celah XSS dan memastikan data yang ditampilkan kepada pengguna tetap aman.

7.5. Recovery (Pemulihan)

1. Build ulang aplikasi

```
docker build -t juice-shop:xss-fixed .
```



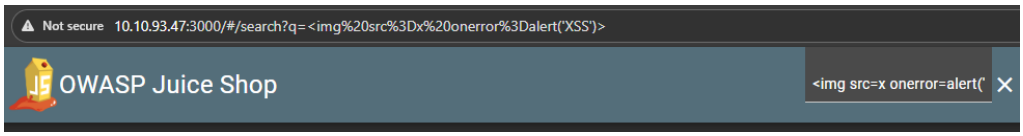
Perintah ini berfungsi untuk membangun (build) sebuah Docker Image baru yang menyertakan perubahan kode (perbaikan XSS) yang telah dilakukan.

2. Jalankan Aplikasi Kembali

```
docker run -d -p 3000:3000 juice-shop:xss-fixed
```

Perintah ini berfungsi untuk menjalankan container berdasarkan image yang baru saja Anda buat. Opsi -d *(detached mode)* memastikan container berjalan di latar belakang sehingga terminal tetap bisa digunakan. Cek Dengan `docker ps` dan aplikasi Juice Shop yang sudah aman dari celah XSS akan aktif dan dapat diakses melalui browser.

3. Cek kerentanan XSS di search aplikasi



Setelah aplikasi kembali berjalan, ketika payload berbahaya berupa *<img src=x onerror=alert('XSS')>* dimasukkan ke dalam parameter pencarian, aplikasi tidak lagi mengeksekusi skrip tersebut (tidak muncul pop up), melainkan menampilkannya sebagai teks biasa di antarmuka pengguna. Hal ini menandakan bahwa fungsi *utils.escapeHtml()* telah bekerja dengan benar untuk mengubah karakter khusus seperti < dan > menjadi entitas HTML, sehingga peramban menganggap input tersebut sebagai data murni dan bukan instruksi kode yang harus dijalankan. Dapat disimpulkan bahwa perbaikan celah keamanan *Cross-Site Scripting (XSS)* telah berhasil diterapkan.

EVALUASI & KESIMPULAN

Evaluasi

Evaluasi ini mencakup efektivitas langkah-langkah keamanan yang telah diimplementasikan dalam siklus hidup aplikasi:

1. Identifikasi Risiko yang Akurat

   Threat Modeling menggunakan metode STRIDE dan DREAD berhasil memetakan ancaman nyata. Identifikasi Brute Force dan SQL Injection sebagai risiko tingkat tinggi, yang menjadi landasan kuat untuk prioritas pengamanan.

2. Efektivitas Pemindaian Keamanan (SAST & SCA)
   - Penggunaan *npm audit* berhasil mengungkap kondisi kritis aplikasi dengan deteksi 45 celah keamanan, termasuk 7 kerentanan tingkat Critical.
   - Melalui *Trivy FS Scan*, Anda menemukan kesalahan fatal berupa kebocoran kunci privat *(Asymmetric Private Key)* yang tertanam dalam kode sumber di file *insecurity.js* dan *insecurity.ts*. Temuan ini sangat krusial karena kebocoran kunci enkripsi merupakan risiko keamanan tingkat tinggi.

3. Keberhasilan Hardening Infrastruktur

   Transformasi dari Dockerfile Insecure ke Dockerfile Hardened menunjukkan peningkatan keamanan yang signifikan. Temuan tingkat HIGH (AVD-DS-0002) berhasil diperbaiki dengan mengganti pengguna *root* menjadi *appuser (non-root)* dan beralih ke base image Alpine yang lebih ringkas dan aman.

4. Ketepatan Respon Insiden

   Simulasi penanganan insiden *DOM-Based XSS* dilakukan dengan metodologi yang benar yaitu *identification, containment, erdadiction,* dan *recovery*. Penggunaan fungsi *utils.escapeHtml()* untuk output encoding terbukti berhasil menetralkan payload berbahaya sehingga hanya ditampilkan sebagai teks biasa oleh browser.

Kesimpulan

Dari seluruh proses pengerjaan sistem ini, terdapat beberapa pelajaran penting *(lesson learned)* yang bisa diambil:

1. Keamanan harus dilakukan sedini mungkin *(Shift Left)* dengan integrasi pemindaian otomatis dalam CI/CD pipeline memungkinkan penemuan kesalahan fatal, seperti kunci privat yang terekspos, sebelum aplikasi mencapai tahap produksi.
2. Bahaya tersembunyi pada dependensi ditandai dengan tingginya jumlah kerentanan (45 temuan) pada pustaka pihak ketiga menunjukkan bahwa keamanan aplikasi bukan hanya tentang kode yang kita tulis, tetapi juga tentang ekosistem library yang kita gunakan. Audit rutin (npm audit) adalah keharusan.
3. Prinsip hak akses terendah *(Least Privilege)* dilakukan dengan menjalankan container dengan akses *root* adalah risiko besar yang sering diabaikan. Menggunakan pengguna *non-root* dan sistem operasi yang minimal (seperti Alpine) secara drastis mempersempit permukaan serangan (attack surface).
4. Monitoring berbasis metrik sebagai pertahanan lapis kedua dilakukan ketika log aplikasi tidak memadai untuk mendeteksi serangan, metrik sumber daya *(CPU/RAM/Traffic)* dapat menjadi "mata" tambahan untuk mengenali aktivitas mencurigakan secara *real-time*.
5. Validasi dan sanitasi adalah fondasi keamanan Web. Pada kasus XSS pada fitur pencarian mengajarkan bahwa menganggap semua input pengguna sebagai ancaman dan melakukan encoding pada sisi output adalah langkah sederhana namun sangat efektif untuk mencegah eksploitasi skrip berbahaya

REFERENSI

1. OWASP Juice Shop (Official Project Page)
   https://owasp.org/www-project-juice-shop/

2. OWASP Top 10:2021 - A03: Injection
   https://owasp.org/Top10/A03_2021-Injection/

3. OWASP Top 10:2021 - A06
   https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

4. Docker Security Best Practices
   https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

5. Microsoft Threat Modeling (STRIDE)
   https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats

6. OWASP DevSecOps Guideline
   https://owasp.org/www-project-devsecops-guideline/

LAMPIRAN

1. Repository Aplikasi
   https://github.com/FFbryn/uas-devsecops-juice-shop-surya

2. Laporan Insiden Handling
   https://github.com/FFbryn/uas-devsecops-juice-shop-surya/blob/main/UAS_Incident_Handling_SuryaAF_88032023007.pdf