# LAPORAN PENANGANAN INSIDEN KEAMANAN

## *(Incident Handling Report)*

**Kasus: *XSS Attack* pada Aplikasi Juice Shop**


Disusun Oleh:

Surya Aditia Febriyan | 88032023007


**POLITEKNIK BHAKTI SEMESTA**

**REKAYASA KEAMANAN SIBER**

**2025**

# PERNYATAAN KERAHASIAAN

Dokumen ini berisi informasi terkait insiden keamanan sistem dan hanya digunakan untuk kepentingan analisis internal, pembelajaran, dan peningkatan keamanan. Dilarang mendistribusikan dokumen ini tanpa izin pihak terkait.

# DAFTAR ISI

# RINGKASAN EKSEKUTIF

Keamanan aplikasi dan infrastruktur digital merupakan aspek krusial yang harus dijaga secara berkelanjutan oleh setiap organisasi. Seiring meningkatnya ketergantungan terhadap sistem berbasis aplikasi, risiko terjadinya insiden keamanan seperti injeksi skrip berbahaya, pencurian data sesi, dan manipulasi antarmuka pengguna juga semakin tinggi. Berdasarkan praktik terbaik penanganan insiden keamanan yang mengacu pada standar NIST SP 800-61 *(Computer Security Incident Handling Guide)*, setiap organisasi diwajibkan memiliki kemampuan untuk mendeteksi, merespons, dan memulihkan sistem dari insiden keamanan secara cepat dan terstruktur.

Laporan ini disusun untuk mendokumentasikan temuan insiden keamanan berupa kerentanan *DOM-Based Cross-Site Scripting (XSS)* pada fitur pencarian aplikasi *OWASP Juice Shop*, yang teridentifikasi dalam siklus pengembangan DevSecOps. Insiden ini ditandai dengan kemampuan sistem mengeksekusi payload skrip berbahaya yang dimasukkan melalui parameter pencarian, yang berpotensi mengancam keamanan sisi klien *(client-side)* dan integritas data pengguna apabila tidak segera ditangani.

Tujuan utama dari penyusunan laporan insiden ini adalah untuk memberikan gambaran menyeluruh mengenai kronologi kejadian, dampak insiden, akar penyebab, serta langkah-langkah mitigasi teknis yang telah dilakukan. Melalui proses identifikasi dan pengujian keamanan yang ketat, insiden berhasil dideteksi pada tahap awal sehingga tidak berkembang menjadi pelanggaran keamanan yang lebih serius, seperti pengambilalihan akun *(account hijacking)* atau pencurian cookie pengguna.

Sebagai bagian dari proses penanganan insiden, dilakukan serangkaian tindakan yang mencakup analisis akar penyebab pada kode sumber, isolasi kerentanan, serta langkah *Eradication* (Pembersihan) melalui implementasi fungsi sanitasi *utils.escapeHtml()*. Tindakan ini bertujuan untuk memastikan bahwa setiap input pengguna diproses sebagai data murni dan bukan sebagai instruksi kode. Tahap pemulihan (*Recovery*) dilakukan dengan membangun kembali container Docker yang telah diperkuat (*hardened*) dan memverifikasi bahwa celah keamanan tersebut telah tertutup sepenuhnya.

Hasil dari penanganan insiden ini disajikan dalam bentuk laporan yang terstruktur, mencakup analisis risiko serta rekomendasi perbaikan yang bersifat teknis dan prosedural. Diharapkan laporan ini dapat menjadi dasar bagi organisasi dalam

meningkatkan kesiapan incident handling, memperkuat kontrol keamanan aplikasi melalui pendekatan *Shift Left Security*, serta mengintegrasikan aspek keamanan secara lebih efektif ke dalam alur kerja DevSecOps guna mencegah terulangnya insiden serupa di masa mendatang.

# LATAR BELAKANG INSIDEN

Aplikasi berbasis web seperti OWASP Juice Shop merupakan komponen krusial dalam layanan digital modern, namun sering kali memiliki permukaan serangan *(attack surface)* yang luas akibat kompleksitas arsitektur dan ketergantungan pada pustaka pihak ketiga. Di tengah proses pengembangan yang cepat, risiko terjadinya insiden keamanan menjadi sangat tinggi jika aspek keamanan tidak diintegrasikan sejak awal. Oleh karena itu, kesiapan sistem dalam mengidentifikasi dan menangani kerentanan menjadi faktor penentu untuk menjaga integritas data dan kepercayaan pengguna.

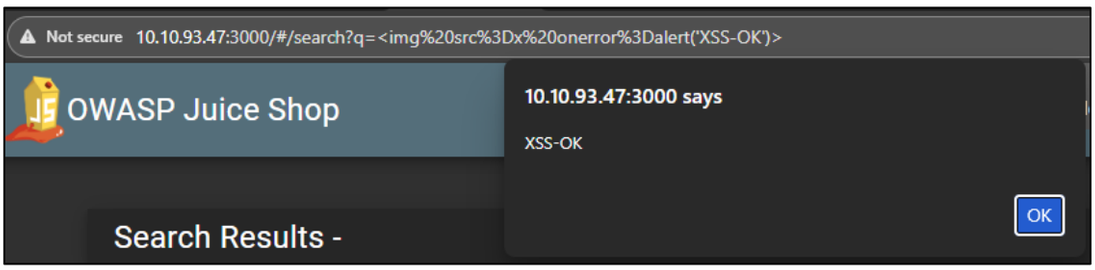
Salah satu ancaman serius yang ditemukan dalam simulasi ini adalah serangan *Cross-Site Scripting (DOM-Based XSS)* pada fitur pencarian aplikasi. Kerentanan ini terjadi karena aplikasi belum melakukan sanitasi input dan output encoding yang memadai terhadap data yang dimasukkan oleh pengguna. Jika tidak ditangani, celah keamanan ini memungkinkan penyerang mengeksekusi skrip JavaScript berbahaya di peramban pengguna, yang berpotensi menyebabkan pencurian sesi atau manipulasi informasi sensitif.

Dalam pendekatan DevSecOps, penanganan insiden dilakukan secara terstruktur melalui tahapan identifikasi, isolasi *(containment)*, perbaikan *(eradication)*, hingga pemulihan *(recovery)*. Simulasi penanganan insiden XSS ini menunjukkan pentingnya penerapan kontrol keamanan teknis, seperti penggunaan fungsi escape HTML untuk memastikan setiap input pengguna hanya diproses sebagai teks murni dan bukan instruksi kode. Laporan ini disusun untuk mendokumentasikan langkah-langkah respons insiden tersebut guna meminimalkan dampak serangan dan memastikan aplikasi kembali beroperasi dalam kondisi aman.

# RINGKASAN INSIDEN

Insiden yang terjadi adalah Administrator sistem melakukan pengujian keamanan aplikasi OWASP Juice Shop dengan mensimulasikan serangan Cross-Site Scripting (XSS) pada fitur pencarian (search). Serangan dilakukan dengan 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..

Sistem mendeteksi insiden melalui munculnya pop up setelah payload dikirim:




Insiden ini dikategorikan sebagai:

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

Tingkat Keparahan : High

Status : Terkendali dan Ditangani

# WAKTU DAN KRONOLOGI KEJADIAN

Insiden keamanan teridentifikasi selama periode operasional sistem melalui mekanisme monitoring dan pengujian penetrasi pada fitur aplikasi. Waktu kejadian dicatat berdasarkan hasil observasi eksekusi skrip pada sisi klien dan log aplikasi, dengan fokus pada urutan peristiwa utama yang relevan terhadap penanganan insiden XSS.

**Waktu Kejadian:**

Tanggal : 31 Desember 2025

Rentang Waktu : 13:30 – 15:00

Status : Insiden berhasil dideteksi dan ditangani pada hari yang sama

**Kronologi Kejadian:**

1. Awal Aktivitas Mencurigakan

   Percobaan injeksi skrip berbahaya dilakukan pada fitur pencarian aplikasi OWASP Juice Shop. Penyerang (atau administrator dalam simulasi ini) memasukkan payload berupa *<img src=x onerror=alert('XSS')>* ke dalam kotak pencarian guna menguji celah keamanan pada sisi frontend.

2. Deteksi Anomali oleh Monitoring

   Insiden terdeteksi secara visual saat peramban mengeksekusi skrip tersebut dan menampilkan pop-up JavaScript "XSS". Hal ini mengonfirmasi adanya kerentanan DOM-Based XSS karena aplikasi gagal melakukan sanitasi input dan output encoding yang memadai.

3. Konfirmasi Insiden

   Berdasarkan hasil eksekusi payload dan analisis alur data, aktivitas tersebut dikonfirmasi sebagai insiden keamanan tingkat tinggi. Kerentanan ini ditemukan pada penggunaan fungsi *req.__()* yang tidak membersihkan karakter berbahaya sebelum ditampilkan kembali ke pengguna.

4. Inisiasi Respons

   Insiden Setelah insiden dikonfirmasi, prosedur incident handling segera diaktifkan. Langkah awal dilakukan dengan menghentikan layanan container aplikasi *(containment)* untuk mencegah eksploitasi lebih lanjut, diikuti dengan penerapan *fungsi utils.escapeHtml()* pada kode sumber untuk menutup celah keamanan tersebut.

# DAMPAK INSIDEN

Insiden keamanan yang teridentifikasi memberikan dampak pada beberapa aspek sistem dan operasional. Meskipun tidak menyebabkan kerusakan infrastruktur secara fisik atau kehilangan data pada basis data secara langsung, insiden ini memiliki implikasi serius yang perlu menjadi perhatian dari sisi keamanan data pengguna, operasional tim pengembang, dan reputasi bisnis.

**Dampak terhadap Keamanan Sistem**

Eksploitasi DOM-Based XSS menunjukkan adanya celah pada sisi klien di mana skrip berbahaya dapat dijalankan di dalam peramban pengguna. Jika aktivitas ini dilakukan oleh pihak yang tidak bertanggung jawab, terdapat risiko tinggi terjadinya pencurian session cookie *(session hijacking)*, pengambilalihan akun, hingga serangan phishing yang menyasar pengguna lain. Kondisi ini secara langsung mengancam kerahasiaan *(confidentiality)* dan integritas *(integrity)* data sensitif yang dikelola oleh aplikasi.

**Dampak terhadap Ketersediaan Layanan**

Secara teknis, serangan XSS tidak menyebabkan lonjakan beban server seperti serangan *Brute Force* atau DDoS. Namun, dalam konteks penanganan insiden, ketersediaan layanan sempat terganggu sementara pada saat proses isolasi *(containment)* dilakukan. Penghentian kontainer aplikasi untuk melakukan perbaikan kode sumber mengakibatkan layanan tidak dapat diakses oleh pengguna selama beberapa waktu guna memastikan celah keamanan tertutup sepenuhnya sebelum aplikasi dipublikasikan kembali.

**Dampak terhadap Operasional**

Insiden ini memerlukan intervensi tim DevSecOps untuk melakukan analisis mendalam terhadap kode sumber *(code review)*, identifikasi titik kerentanan pada fungsi *req.__()*, serta penerapan mitigasi berupa output encoding menggunakan *utils.escapeHtml()*. Penanganan insiden ini menambah beban kerja operasional dan menunjukkan perlunya pengintegrasian alat pemindaian keamanan otomatis yang lebih ketat dalam pipeline CI/CD untuk mendeteksi kerentanan serupa di masa mendatang.

**Dampak terhadap Risiko Bisnis**

Dari sudut pandang bisnis, keberadaan celah XSS yang dapat dieksploitasi dengan mudah dapat merusak kepercayaan pengguna terhadap keamanan platform. Jika terjadi

kebocoran data sesi pengguna, perusahaan berisiko menghadapi tuntutan hukum terkait perlindungan data pribadi dan pelanggaran kepatuhan terhadap standar keamanan industri. Insiden ini menegaskan pentingnya penerapan Secure SDLC secara konsisten untuk menghindari kerugian finansial dan reputasi di masa depan.

# ANALISIS AKAR PENYEBAB *(ROOT CAUSE)*

Berdasarkan analisis log aplikasi, peninjauan kode sumber (code review), serta hasil simulasi serangan pada fitur pencarian aplikasi OWASP Juice Shop, insiden DOM-Based XSS terjadi akibat kelemahan pada cara aplikasi memproses dan menampilkan data yang berasal dari input pengguna. Akar penyebab utama insiden diidentifikasi sebagai ketiadaan mekanisme output encoding yang memadai sebelum data ditampilkan kembali ke peramban pengguna.

Secara lebih rinci, faktor-faktor yang berkontribusi terhadap terjadinya insiden meliputi:

1. Ketiadaan Sanitasi Input dan Output Encoding

   Aplikasi menerima input dari parameter pencarian dan langsung merendernya ke dalam *Document Object Model* (DOM) tanpa melakukan pembersihan *(filtering)* terhadap karakter-karakter khusus HTML. Kondisi ini memungkinkan skrip berbahaya yang disisipkan penyerang dieksekusi oleh peramban sebagai instruksi kode, bukan sebagai teks biasa.

2. Penggunaan Fungsi Pemrosesan Data yang Tidak Aman

   Aplikasi menggunakan fungsi atau metode yang secara implisit menganggap input pengguna adalah aman. Dalam kasus ini, penggunaan fungsi seperti ***req.__()*** untuk menampilkan kembali kata kunci pencarian tanpa pembungkus fungsi escape menyebabkan payload seperti ***<img src=x onerror=alert('XSS')>*** dapat memicu kegagalan pemuatan gambar yang berujung pada eksekusi skrip JavaScript.

3. Belum Optimalnya Pemindaian Keamanan Otomatis pada Tahap Pengembangan

   Meskipun pipeline CI/CD telah dibangun, kerentanan pada level logika tampilan seperti *DOM-Based XSS* seringkali luput dari pemindaian statis sederhana. Tidak adanya pengujian keamanan dinamis (DAST) yang spesifik menyasar manipulasi DOM menjadi faktor pendukung mengapa celah ini tetap ada hingga tahap operasional.

4. Penerapan Prinsip *Secure Coding* yang Belum Menyeluruh

   Fokus pengembangan fitur pencarian masih menitikberatkan pada fungsionalitas dan pengalaman pengguna *(user experience)*, sementara kontrol keamanan dasar pada titik keluar data *(data exit point)* belum diimplementasikan secara menyeluruh sejak awal tahap pengkodean.

Analisis ini menunjukkan bahwa insiden tersebut disebabkan oleh kelalaian dalam menangani data yang tidak tepercaya *(untrusted data)* pada sisi klien. Kondisi ini menegaskan pentingnya penerapan prinsip *Context-Aware Output Encoding* dan integrasi keamanan yang lebih ketat dalam setiap tahapan pengembangan aplikasi guna mencegah eksploitasi celah keamanan web yang umum terjadi.

# TINDAKAN PENANGANAN INSIDEN

Penanganan insiden dilakukan secara terstruktur dengan mengacu pada tahapan *Incident Response Lifecycle* sebagaimana direkomendasikan dalam NIST SP 800-61. Tujuan utama dari tindakan ini adalah menghentikan eksekusi skrip berbahaya, memperbaiki kerentanan pada kode sumber, serta memastikan aplikasi OWASP Juice Shop kembali beroperasi secara aman.

**1. Identifikasi Insiden (Identification)**

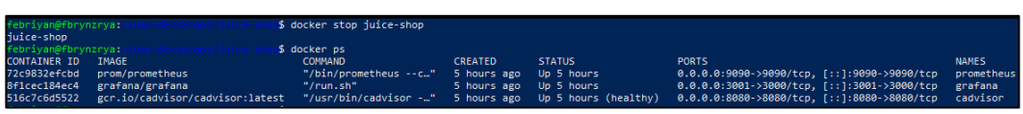
Langkah awal dilakukan dengan mengonfirmasi adanya celah keamanan aktif pada fitur pencarian aplikasi. Proses ini mencakup:

- Pengujian Payload dengan melakukan injeksi skrip <img src=x onerror=alert('XSS')> pada kolom pencarian untuk memvalidasi apakah aplikasi merendernya sebagai kode atau teks.
- Analisis Sisi Klien dengan observasi munculnya kotak dialog (alert) pada peramban yang menandakan skrip JavaScript berhasil dieksekusi melalui manipulasi DOM.
- Berikut Pop Up yang muncul:


**2. Isolasi Insiden (Containment)**

Setelah insiden terkonfirmasi, langkah isolasi dilakukan untuk membatasi risiko eksploitasi oleh pengguna lain selama proses perbaikan berlangsung. Tindakan yang diambil meliputi:

- Penghentian Layanan Sementara dengan menghentikan kontainer Docker yang menjalankan aplikasi *(docker stop [container_id])* guna mencegah akses publik ke versi aplikasi yang masih memiliki celah keamanan.
- Isolasi Lingkungan Pengembangan: Memindahkan proses analisis ke lingkungan lokal *(staging)* untuk melakukan *debugging* tanpa memengaruhi ketersediaan layanan utama jika dalam skenario produksi. Langkah ini memastikan tidak ada sesi pengguna yang dikompromikan saat tim teknis menyiapkan perbaikan kode.
- Proses penghentian aplikasi sementara:

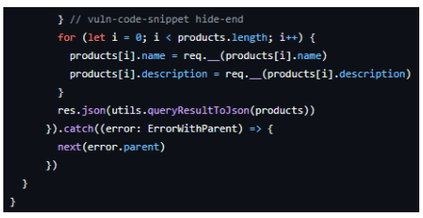


3. **Pembersihan Insiden (Eradication)**

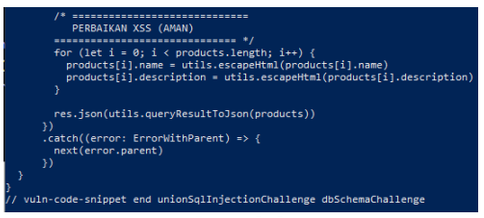
Pada tahap ini, dilakukan tindakan korektif untuk menghilangkan akar penyebab insiden, yaitu kerentanan pada kode sumber. Tahap *eradication* difokuskan pada penguatan validasi data. Tindakan yang dilakukan antara lain:

- Pencarian file yang teridentifikasi salah konfigurasi.


- Penerapan Output Encoding: Melakukan modifikasi pada file routes/search.ts (atau file terkait) dengan mengimplementasikan fungsi utils.escapeHtml() pada hasil pencarian sebelum ditampilkan kembali ke antarmuka pengguna.




Sebelum

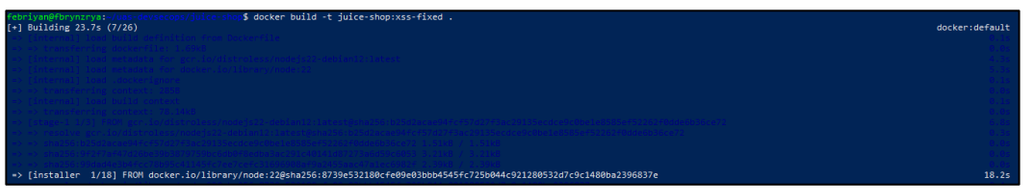



Sesudah

4. **Pemulihan Sistem (Recovery)**

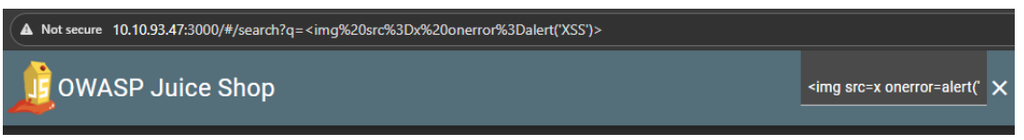
Setelah tindakan pembersihan selesai, sistem dikembalikan ke kondisi operasional normal. Proses pemulihan meliputi:

- Deployment versi aman dengan membangun ulang image Docker dan menjalankan kembali kontainer aplikasi dengan kode yang telah diperbaiki.





- Pengujian Validasi *(Sanity Check)* dengan mencoba memasukkan kembali payload XSS yang sama untuk memastikan bahwa aplikasi kini menampilkannya sebagai teks murni dan tidak lagi mengeksekusi skrip tersebut.



# REKOMENDASI DAN TINDAK LANJUT

Berdasarkan hasil analisis insiden *DOM-Based XSS*, penanganan yang telah dilakukan melalui perbaikan kode sumber, serta evaluasi terhadap kontrol keamanan pada aplikasi OWASP Juice Shop, berikut adalah rekomendasi dan rencana tindak lanjut yang bertujuan untuk meningkatkan ketahanan sistem dan mencegah terulangnya insiden serupa di masa mendatang.

**1. Rekomendasi Teknis**

a. Standardisasi Context-Aware Output Encoding

Disarankan untuk menerapkan output encoding secara menyeluruh pada setiap titik di mana data pengguna ditampilkan kembali ke peramban. Penggunaan fungsi *utils.escapeHtml()* harus menjadi standar wajib untuk memastikan karakter berbahaya seperti < dan > selalu diubah menjadi entitas teks biasa.

b. Implementasi Content Security Policy (CSP)

Sistem perlu dikonfigurasi dengan kebijakan CSP yang ketat melalui HTTP Header. CSP akan membatasi sumber skrip yang boleh dijalankan dan mencegah eksekusi skrip inline yang tidak sah, sehingga memberikan lapisan pertahanan tambahan jika terjadi kegagalan sanitasi.

c. Validasi Input pada Sisi Server

Meskipun serangan ini terjadi di sisi klien (DOM), penerapan validasi input yang ketat pada sisi server tetap diperlukan untuk memastikan bahwa data yang masuk ke sistem telah melewati filter keamanan dasar sebelum diproses lebih lanjut.

**2. Rekomendasi Operasional**

a. Otomasi Security Alerting

Sistem monitoring (seperti Prometheus/Grafana atau sistem logging) perlu dilengkapi dengan alert otomatis yang mendeteksi pola karakter aneh pada parameter URL atau request body, sehingga upaya injeksi skrip dapat diketahui secara *real-time*.

b. Program Edukasi Secure Coding (OWASP Top 10)

Disarankan untuk mengadakan pelatihan berkala bagi tim pengembang mengenai prinsip Secure SDLC, dengan fokus khusus pada kerentanan *Injection* dan *Cross-Site Scripting* guna menumbuhkan budaya peduli keamanan sejak tahap penulisan kode.

c. Audit Keamanan Kode (Manual Code Review)

   Melakukan tinjauan kode secara manual pada fitur-fitur kritis yang berinteraksi langsung dengan input pengguna, guna memastikan tidak ada fungsi berisiko seperti `innerHTML` yang digunakan tanpa kontrol keamanan.

**3. Tindak Lanjut**

Sebagai tindak lanjut dari insiden ini, akan dilakukan:

a. Implementasi perbaikan kode menggunakan utils.escapeHtml() pada seluruh fitur aplikasi yang masih memiliki potensi kerentanan serupa.
b. Evaluasi dan pemutakhiran pustaka (dependency) aplikasi secara berkala berdasarkan hasil temuan npm audit dan Trivy yang tercantum dalam laporan.
c. Pengujian penetrasi (penetration testing) ulang pada fitur pencarian dan fitur input lainnya untuk memverifikasi efektivitas mitigasi yang telah diterapkan.

Dengan dilaksanakannya rekomendasi dan tindak lanjut tersebut, diharapkan risiko serangan *web injection* dapat diminimalisir dan kesiapan tim DevSecOps dalam menjaga keamanan aplikasi dapat ditingkatkan secara berkelanjutan.

# KESIMPULAN

Insiden keamanan berupa kerentanan *DOM-Based Cross-Site Scripting (XSS)* pada fitur pencarian aplikasi OWASP Juice Shop berhasil terdeteksi dan ditangani secara efektif melalui pengujian penetrasi manual serta analisis alur data pada sisi klien. Meskipun insiden ini tidak menyebabkan kebocoran data sensitif atau kerusakan basis data secara langsung, temuan ini menunjukkan adanya kelemahan kritis pada kontrol keamanan dasar, khususnya dalam hal sanitasi input dan *output encoding*, yang perlu segera diperbaiki secara permanen.

Proses penanganan insiden yang dilakukan, mulai dari tahap identifikasi melalui pengujian payload, isolasi dengan penghentian *container* aplikasi, perbaikan kode sumber menggunakan fungsi *utils.escapeHtml()*, hingga pemulihan layanan, telah berjalan sesuai dengan tahapan *Incident Response Lifecycle*. Tindakan yang terstruktur ini terbukti mampu menghilangkan akar masalah dan mencegah eskalasi risiko, seperti pencurian sesi pengguna *(session hijacking)* atau manipulasi antarmuka aplikasi oleh pihak yang tidak bertanggung jawab.

Temuan dari insiden ini menegaskan bahwa ancaman keamanan sering kali tidak berasal dari eksploitasi infrastruktur yang kompleks, melainkan dari kurangnya penerapan prinsip *secure coding* pada komponen aplikasi yang berinteraksi langsung dengan input pengguna. Oleh karena itu, integrasi keamanan ke dalam proses pengembangan melalui praktik DevSecOps, seperti penggunaan alat pemindaian otomatis (SAST/DAST) dan tinjauan kode secara berkala, harus dilakukan secara konsisten untuk memastikan celah serupa tidak muncul kembali di masa mendatang.

Dengan menerapkan rekomendasi teknis seperti penggunaan *Content Security Policy* (CSP) dan tindak lanjut penguatan *pipeline CI/CD* yang telah diusulkan, organisasi diharapkan dapat meningkatkan ketahanan sistem terhadap serangan berbasis web. Upaya perbaikan berkelanjutan ini sangat penting untuk menekan risiko keamanan, memastikan keberlangsungan layanan, serta menjaga integritas data dan kepercayaan pengguna secara jangka panjang.

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