Laman web jurnal: https://ejournal.unama.ac.id/index.php/processor

Jurnal Processor

P-ISSN: 1907-6738 | E-ISSN: 2528-0082

Analisis Keamanan Aplikasi Web Menggunakan Penetration Testing Berdasarkan Standar OWASP Pada DVWA

Muhamad Reza Fahlevi[1*], Sharipuddin[2], Kurniabudi[3], Dwi Chaya Medika[4]

[1,2,3,4] Fakultas Ilmu Komputer, Program Studi Teknik Informatika, Universitas Dinamika Bangsa, Jl. Jend. Sudirman, The Hok, Kec. Jambi Sel., Kota Jambi, Jambi 36138, Indonesia.
*Penulis Korespondensi, Email: muhamadrezaf00@gmail.com

Abstrak—Aplikasi berbasis web kini menjadi infrastruktur vital layanan publik, namun sering dikembangkan dengan mengabaikan prinsip *Secure Software Development Life Cycle* (SSDLC). Penelitian ini bertujuan menganalisis mekanisme eksploitasi pada celah keamanan kritis dan mengevaluasi efektivitas pertahanan kode (*secure coding*) dibandingkan pertahanan infrastruktur. Menggunakan objek uji *Damn Vulnerable Web App* (DVWA) pada lingkungan server lokal terisolasi, penelitian ini menerapkan standar *Penetration Testing Execution Standard* (PTES). Fokus pengujian adalah membedah pola serangan *SQL Injection* (SQLi) dan *Stored Cross-Site Scripting* (XSS). Hasil analisis pola serangan menunjukkan bahwa absennya validasi input memungkinkan injeksi *payload* tautologi yang mengekspos 100% data pengguna (pelanggaran *Confidentiality*) dan injeksi skrip persisten yang membahayakan sisi klien (*Integrity*). Kontribusi ilmiah penelitian ini terletak pada pemetaan karakteristik *payload* serangan terhadap respons basis data MySQL, yang membuktikan bahwa ketergantungan pada *Web Application Firewall* (WAF) saja tidak memadai tanpa penerapan *Prepared Statements* dan sanitasi *output* pada level kode.

Kata Kunci: Keamanan Web; Penetration Testing; OWASP; SQL Injection; Stored XSS.

Abstract—Web-based applications have become vital infrastructure for public services, yet they are often developed while neglecting Secure Software Development Life Cycle (SSDLC) principles. This study aims to analyze exploitation mechanisms of critical security vulnerabilities and evaluate the effectiveness of secure coding defenses compared to infrastructure defenses. Using the Damn Vulnerable Web App (DVWA) in an isolated local server environment, this study applies the Penetration Testing Execution Standard (PTES). The testing focuses on dissecting SQL Injection (SQLi) and Stored Cross-Site Scripting (XSS) attack patterns. Results of the attack pattern analysis indicate that the absence of input validation allows for tautology payload injection, exposing 100% of user data (Confidentiality breach), and persistent script injection that compromises the client-side (Integrity). The scientific contribution of this study lies in mapping attack payload characteristics against MySQL database responses, proving that reliance on Web Application Firewalls (WAF) alone is insufficient without implementing Prepared Statements and output sanitization at the code level.

Keywords: Web Security; Penetration Testing; OWASP; SQL Injection; Stored XSS.

1. PENDAHULUAN

Transformasi digital di Indonesia telah mengubah cara organisasi beroperasi secara fundamental. Meskipun efisiensi meningkat, pergeseran ini membuka celah keamanan baru. Data menunjukkan bahwa aplikasi web kini menjadi target utama serangan siber karena sering kali dikembangkan dengan prioritas kecepatan rilis dibanding keamanan kode. Internet kini bukan lagi sekadar media pertukaran informasi statis, melainkan telah berevolusi menjadi ekosistem digital yang kompleks di mana aplikasi berbasis web memegang peranan vital. Organisasi dari berbagai sektor, mulai dari perbankan, *e-commerce*, pendidikan, hingga pemerintahan, kini sangat bergantung pada aplikasi web untuk memberikan layanan yang efisien, cepat, dan terjangkau kepada pengguna. Namun, ketergantungan yang tinggi terhadap infrastruktur digital ini membawa konsekuensi serius berupa meningkatnya permukaan serangan (*attack surface*) yang dapat dieksploitasi oleh pihak-pihak yang tidak bertanggung jawab. Keamanan siber (*cyber security*) kini bukan lagi opsi tambahan, melainkan kebutuhan fundamental yang harus dipenuhi untuk menjamin keberlangsungan operasional suatu sistem.

Berdasarkan laporan lanskap ancaman siber global tahun 2024, serangan terhadap aplikasi web terus menempati posisi teratas dalam statistik kejahatan siber. Data sensitif seperti informasi pribadi (*Personally Identifiable Information* - PII), kredensial login (*username* dan *password*), serta data finansial menjadi komoditas utama yang diincar oleh peretas di pasar gelap (*dark web*). Fenomena kebocoran data (*data breach*) yang marak terjadi di Indonesia belakangan ini menjadi bukti nyata bahwa sistem keamanan yang diterapkan pada banyak

Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

aplikasi web masih memiliki celah kerentanan (*vulnerability*) yang signifikan. Masalah mendasar yang sering ditemukan adalah kurangnya implementasi prinsip keamanan sejak fase awal pengembangan perangkat lunak atau yang dikenal dengan istilah *Secure Software Development Life Cycle* (SSDLC). Banyak pengembang (*developer*) yang lebih memprioritaskan fungsionalitas fitur dan kecepatan rilis (*time-to-market*) dibandingkan aspek keamanan, sehingga validasi input sering kali terabaikan atau tidak diterapkan dengan standar yang benar [1],[2].

Dua jenis serangan yang paling dominan dan konsisten muncul dalam daftar risiko keamanan tertinggi adalah *SQL Injection (SQLi)* dan *Cross-Site Scripting (XSS)*. Dalam konteks keamanan basis data, serangan *SQL Injection (SQLi)* bekerja dengan memanfaatkan kegagalan aplikasi dalam memisahkan data input pengguna dan perintah instruksi. Hal ini memungkinkan intruder memanipulasi logika backend untuk mengakses informasi yang seharusnya tertutup. Celah ini terjadi ketika input dari pengguna dikirim langsung ke basis data tanpa melalui proses penyaringan (*filtering*) atau parameterisasi yang tepat. Dampak dari serangan ini sangat fatal, mulai dari pencurian seluruh data pengguna, modifikasi data, hingga penghapusan basis data secara permanen yang dapat melumpuhkan sistem [3],[4]. Berbeda dengan *SQL Injection* yang menyerang sisi server (*server-side*), *Cross-Site Scripting (XSS)* menargetkan pengguna lain dari sisi klien (client-side). Celah ini timbul ketika aplikasi menampilkan kembali input mentah dari pengguna tanpa proses sanitasi, sehingga peramban menerjemahkannya sebagai kode eksekusi, bukan teks biasa. (biasanya JavaScript) ke dalam halaman web yang dilihat oleh pengguna lain. Berbeda dengan SQLi yang menyerang sisi server (*server-side*), XSS menyerang sisi klien (*client-side*). Serangan ini dapat digunakan untuk mencuri session cookies, melakukan pengalihan halaman (*redirect)* ke situs phishing, atau bahkan mengambil alih kendali peramban pengguna [5].

Untuk memitigasi risiko tersebut, diperlukan pendekatan evaluasi keamanan yang komprehensif dan proaktif. Salah satu metode yang diakui standar industri adalah *Penetration Testing* (Uji Penetrasi). Penetration Testing adalah simulasi serangan siber yang dilakukan secara legal dan terkendali untuk mengidentifikasi kelemahan sistem sebelum dieksploitasi oleh peretas jahat (black hat hacker). Dalam pelaksanaannya, pengujian ini sering mengacu pada standar internasional seperti Open Web Application Security Project (OWASP) Top 10, yang merupakan dokumen panduan mengenai sepuluh risiko keamanan aplikasi web paling kritis. OWASP memberikan kerangka kerja yang sistematis bagi penguji keamanan untuk mendeteksi, mengeksploitasi, dan memberikan rekomendasi perbaikan terhadap celah keamanan yang ditemukan [6]. Penerapan standar OWASP dalam pengujian penetrasi memastikan bahwa evaluasi yang dilakukan relevan dengan tren ancaman siber terkini.

Penelitian ini bertujuan untuk melakukan analisis mendalam mengenai mekanisme serangan dan pertahanan pada aplikasi web dengan menggunakan Damn Vulnerable Web App (DVWA) sebagai objek studi. DVWA adalah aplikasi web berbasis PHP/MySQL yang sengaja dirancang dengan berbagai celah keamanan untuk tujuan edukasi dan pelatihan keamanan siber. Penulis memilih Damn Vulnerable Web App (DVWA) sebagai objek uji karena kerangka kerjanya menyediakan isolasi lingkungan (sandbox) yang ideal. Hal ini memungkinkan analisis mendalam terhadap logika serangan SQLi dan XSS tanpa risiko hukum atau kerusakan infrastruktur produksi yang sebenarnya. (sandbox environment), sehingga memungkinkan peneliti untuk membedah logika serangan SQL Injection dan Stored XSS secara terperinci tanpa melanggar hukum [7]. Berbeda dengan penelitian sebelumnya yang mungkin hanya berfokus pada deteksi kerentanan menggunakan tools otomatis, penelitian ini menekankan pada analisis manual (manual exploitation) untuk memahami akar penyebab kerentanan pada level kode program (source code).

Fokus utama penelitian ini adalah menganalisis kerentanan pada tingkat keamanan rendah (Low Security), di mana aplikasi tidak menerapkan filter keamanan sama sekali. Kondisi ini merepresentasikan aplikasi web yang dikembangkan tanpa kesadaran keamanan (insecure coding practice). Penelitian ini akan mendemonstrasikan bagaimana SQL Injection dapat dieksploitasi untuk mem-bypass autentikasi dan mengekstrak data, serta bagaimana Stored XSS dapat ditanamkan secara permanen dalam basis data. Selain itu, kontribusi utama dari penelitian ini adalah penyusunan rekomendasi mitigasi yang konkret berupa penulisan ulang kode program (code refactoring) menggunakan metode Prepared Statements dan Input Sanitization. Diharapkan hasil penelitian ini dapat memberikan wawasan teknis yang berharga bagi para pengembang aplikasi web, akademisi, dan praktisi keamanan siber mengenai pentingnya validasi input dan implementasi kode yang aman untuk mencegah insiden kebocoran data di masa depan.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

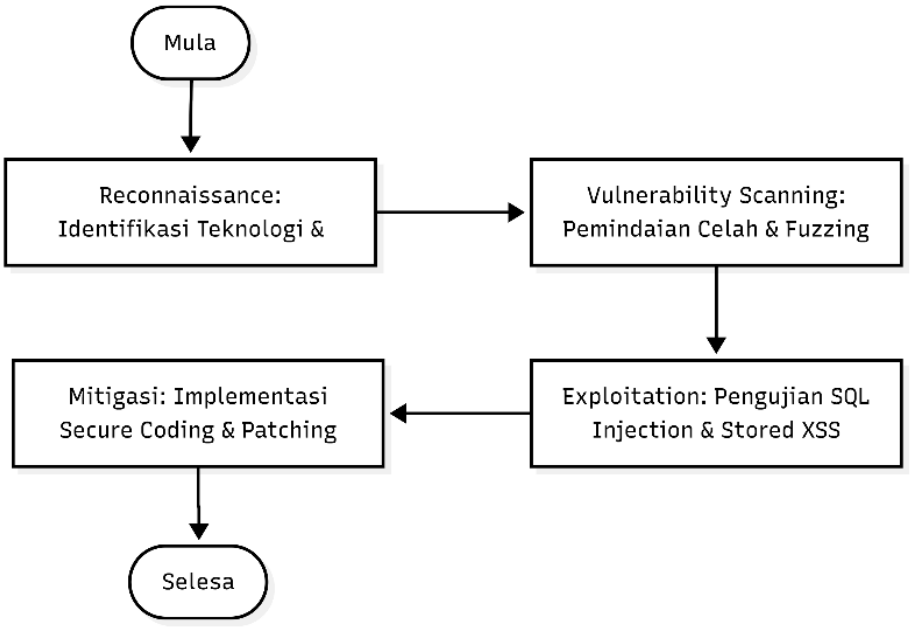
2. METODOLOGI PENELITIAN

2.1 Desain Penelitian

Penelitian ini menggunakan metode eksperimental dengan pendekatan simulasi serangan siber (*cyber attack simulation*). Pendekatan yang dipilih adalah *Black Box Testing*, sebuah metodologi pengujian keamanan di mana penguji menempatkan diri pada posisi penyerang eksternal yang tidak memiliki akses atau pengetahuan awal mengenai arsitektur internal sistem, kode sumber (*source code*), maupun struktur basis data target. Pendekatan ini dipilih karena paling mendekati skenario serangan nyata (*real-world attack scenario*) yang dihadapi oleh aplikasi web publik. Dalam *Black Box Testing*, penguji harus melakukan pengumpulan informasi (*information gathering*) secara mandiri untuk memetakan permukaan serangan sebelum melakukan eksploitasi.

2.2 Alur Penelitian (Framework)

Kerangka kerja penelitian ini mengadaptasi standar *Penetration Testing Execution Standard* (PTES) [8],[9],[10] yang disederhanakan menjadi empat tahapan utama yang siklik dan terstruktur. Tahapan ini dimulai dari pengumpulan informasi hingga penerapan langkah mitigasi, sebagaimana diilustrasikan pada Gambar 1.



Gambar 1. Diagram Alir Tahapan Penelitian *Penetration Testing*

Penjelasan rinci dari setiap tahapan dalam Gambar 1 adalah sebagai berikut:

1. Reconnaissance (Pengumpulan Informasi): Tahap ini merupakan fondasi dari seluruh proses pengujian penetrasi. Tujuannya adalah untuk mengumpulkan informasi sebanyak mungkin mengenai target operasi. Pada penelitian ini, karena target berada pada lingkungan lokal (*localhost*), tahapan ini difokuskan pada identifikasi teknologi yang berjalan (*fingerprinting*).
2. Vulnerability Scanning (Pemindaian Celah Keamanan): Setelah informasi awal terkumpul, tahap selanjutnya adalah memindai potensi celah keamanan. Penelitian ini menggunakan teknik pemindaian manual (*manual probing*) dengan cara menyisipkan karakter-karakter khusus (*fuzzing*) pada setiap formulir input.
3. Exploitation (Eksploitasi): Tahap ini adalah inti dari pengujian penetrasi, di mana peneliti mencoba membuktikan keberadaan celah keamanan dengan melakukan serangan aktif (SQL Injection dan Stored XSS).
4. Reporting & Mitigation (Analisis dan Perbaikan): Tahap akhir melibatkan analisis mendalam terhadap penyebab kerentanan pada level kode dan merancang solusi perbaikan (*patching*) menggunakan praktik pemrograman aman (*secure coding*).

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

2.3 Lingkungan Penelitian

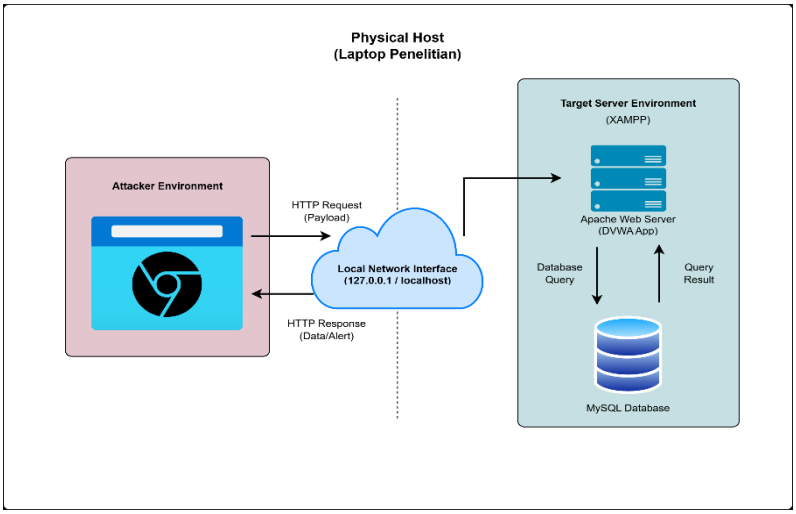
Untuk memastikan keamanan dan mencegah dampak hukum, seluruh pengujian dilakukan pada lingkungan laboratorium tertutup yang terisolasi dari jaringan internet publik. Perangkat keras dan perangkat lunak yang digunakan dalam penelitian ini dirinci pada Tabel 1.

Tabel 1. Spesifikasi Lingkungan Pengujian

Komponen Penelitian	Spesifikasi/Keterangan
Perangkat Keras (Hardware)	Laptop dengan Processor Intel Core i3-1215U, RAM 16 GB, dan Penyimpanan SSD.
Sistem Operasi	Microsoft Windows 11 Home 64-bit.
Web Server Environment	XAMPP Versi 8.2 (Mencakup Apache Web Server dan Database MySQL/MariaDB).
Aplikasi Target	Damn Vulnerable Web App (DVWA) yang di-hosting pada *localhost* dengan konfigurasi *Low Security*
Tools Pendukung	Web Browser Google Chrome dan Text Editor (Notepad/VS Code).

Penggunaan spesifikasi di atas dipilih untuk memastikan bahwa simulasi berjalan lancar tanpa kendala performa, mengingat proses *fuzzing* dan pemindaian kerentanan terkadang membutuhkan sumber daya memori yang cukup besar.

Alur infrastruktur pengujian yang menunjukkan interaksi logis antara mesin penyerang dan server target divisualisasikan pada Gambar 2.



Gambar 2. Topologi Jaringan Logis Lab Virtual

Infrastruktur pengujian dalam penelitian ini divisualisasikan melalui topologi jaringan logis pada **Gambar 2**. Diagram tersebut menggambarkan interaksi antara mesin penyerang (*Attacker*) yang menggunakan peramban web untuk mengirimkan *payload* menuju server target yang menjalankan layanan Apache dan basis data MySQL di lingkungan XAMPP. Seluruh alur komunikasi data dilakukan melalui antarmuka *loopback* internal (127.0.0.1) untuk memastikan proses pengujian berlangsung dalam laboratorium virtual yang terisolasi dan aman dari jaringan luar.

3. HASIL DAN PEMBAHASAN

Pengujian penetrasi dilakukan pada modul aplikasi DVWA yang telah dikonfigurasi pada tingkat keamanan "Low". Tingkat keamanan ini mensimulasikan kondisi aplikasi web yang sangat buruk, di mana pengembang sama sekali tidak menerapkan mekanisme validasi input (*input validation*) maupun penyaringan output (*output encoding*). Pada kondisi ini, setiap data yang dimasukkan oleh pengguna akan diproses secara mentah oleh server dan basis data. Hal ini memberikan peluang besar bagi penyerang untuk melakukan manipulasi instruksi program. Bagian

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

ini akan membahas secara mendalam analisis teknis dari dua vektor serangan yang diuji: *SQL Injection* dan *Stored Cross-Site Scripting* (XSS).

3.1 Analisis Mendalam Serangan SQL Injection

3.1.1 Mekanisme Kerentanan

SQL Injection (SQLi) adalah serangan injeksi kode yang terjadi pada lapisan basis data (*database layer*) dari sebuah aplikasi web. Kerentanan ini muncul ketika aplikasi menggabungkan data input pengguna dengan string kueri SQL secara dinamis tanpa pemisahan yang jelas antara data dan perintah. Pada modul DVWA "SQL Injection", terdapat formulir pencarian "User ID" yang bertujuan untuk menampilkan informasi pengguna (Nama Depan dan Nama Belakang) berdasarkan ID numerik yang dimasukkan.
Pada level keamanan rendah, kode PHP di sisi server diasumsikan menangani input dengan cara yang tidak aman seperti berikut: $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
Dalam kode di atas, variabel $id diambil langsung dari input pengguna (metode GET atau POST) dan disambungkan (*concatenated*) langsung ke dalam string perintah SQL. Basis data tidak dapat membedakan mana bagian yang merupakan perintah asli dari pengembang dan mana bagian yang merupakan data input dari pengguna. Inilah akar permasalahan yang memungkinkan terjadinya injeksi.

Tabel 2. Analisis Pola Serangan (*Attack Pattern*) *SQL Injection*

Komponen Payload	Sintaks	Fungsi dalam Serangan	Respons Database (Logika)
Prefix	'	Menutup *string literal* ID pengguna asli.	Mengakhiri input data yang sah.
Operator	OR	Operator Logika Boolean.	Mengubah logika seleksi data.
Tautology	1=1	Kondisi yang selalu bernilai *TRUE*.	Memaksa database mengabaikan filter ID.
Suffix	#	Karakter komentar MySQL.	Menghapus sisa kueri asli (') agar tidak terjadi *syntax error*.

Analisis Data Serangan: Berdasarkan pengujian, pola serangan yang terbentuk pada HTTP Request adalah sebagai berikut: GET /dvwa/vulnerabilities/sqli/?id=%27+OR+1%3D1+%23&Submit=Submit HTTP/1.1
Server menerjemahkan encoded URL tersebut menjadi kueri SQL mentah: SELECT first_name, last_name FROM users WHERE user_id = '' OR 1=1 #';
Data nyata menunjukkan bahwa basis data tidak memvalidasi tipe data integer pada kolom user_id, sehingga injeksi string dapat dieksekusi. Hal ini mengonfirmasi bahwa kerentanan bukan hanya pada logic error, tetapi juga type handling.

3.1.2 Tahapan Eksploitasi

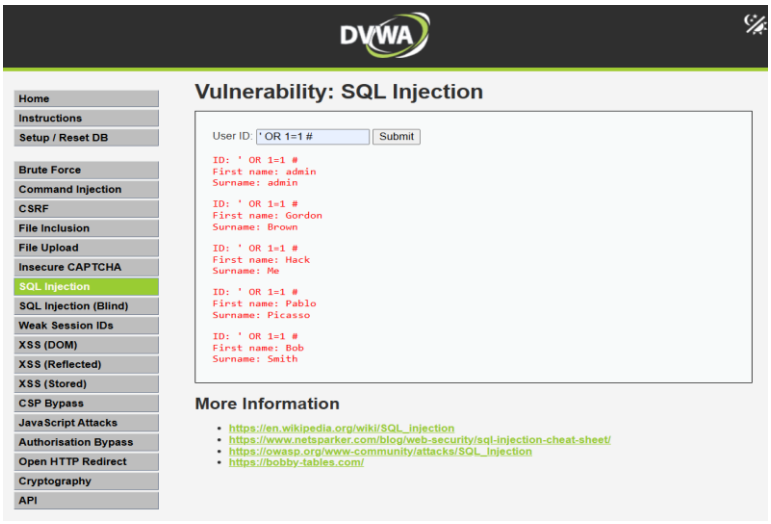
Proses eksploitasi dimulai dengan tahap pengujian respons aplikasi. Ketika peneliti memasukkan input normal berupa angka 1, aplikasi mengembalikan data yang valid: ID: 1, First name: admin, Surname: admin. Ini mengindikasikan bahwa aplikasi berkomunikasi dengan basis data secara normal.
Selanjutnya, peneliti mencoba menyisipkan karakter pemecah kueri, yaitu tanda petik tunggal ('). Jika aplikasi menampilkan pesan kesalahan basis data (SQL Syntax Error), itu adalah indikasi kuat adanya kerentanan SQLi. Namun, pada percobaan ini, peneliti langsung menggunakan teknik Tautology atau Boolean-Based Injection dengan payload: ' OR 1=1 #. Analisis teknis dari payload tersebut telah dijabarkan pada Tabel 2 di atas.

3.1.3 Hasil dan Dampak Eksploitasi

Ketika *payload* tersebut diproses, kueri SQL yang dieksekusi oleh server berubah drastis menjadi: SELECT first_name, last_name FROM users WHERE user_id = '' OR 1=1 #';`

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

Secara logika basis data, perintah ini dibaca sebagai: *"Tampilkan nama depan dan nama belakang dari tabel users di mana user_id kosong ATAU di mana 1 sama dengan 1"*. Karena kondisi 1=1 selalu benar untuk setiap baris data dalam tabel, basis data akhirnya mengembalikan **seluruh baris data** yang ada di tabel users.



Gambar 3. Hasil Eksploitasi SQL Injection Menampilkan Seluruh Data Pengguna

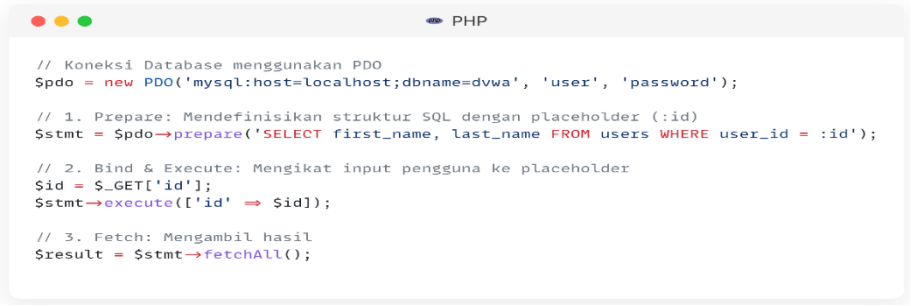
Seperti yang terlihat pada Gambar 3, aplikasi menampilkan daftar lengkap pengguna yang terdiri dari: admin, Gordon, Hack Me, Pablo, dan Bob. Dampak dari kerentanan ini sangat kritis terhadap aspek Kerahasiaan (*Confidentiality*) dalam segitiga keamanan CIA (*Confidentiality, Integrity, Availability*). Penyerang berhasil melakukan *Data Dumping* atau *Data Exfiltration* tanpa otorisasi. Jika tabel ini berisi informasi yang lebih sensitif seperti *password hash*, nomor kartu kredit, atau alamat rumah, kerugian yang ditimbulkan akan sangat masif. Selain itu, jika penyerang menggunakan teknik *UNION BASED SQL Injection*, mereka bahkan bisa membaca data dari tabel lain di luar tabel users, atau dalam kasus terburuk, mengambil alih kendali server basis data sepenuhnya menggunakan perintah FILE atau xp_cmdshell [11],[12].

3.1.4 Mitigasi dan Rekomendasi Perbaikan

Solusi tradisional seperti melakukan validasi input menggunakan *blacklist* (melarang kata kunci OR, SELECT, ') sering kali tidak efektif karena penyerang memiliki banyak cara untuk melakukan *bypass*. Solusi terbaik dan paling direkomendasikan oleh standar industri saat ini adalah penggunaan Prepared Statements atau Parameterized Queries.

Prepared Statements bekerja dengan cara memisahkan struktur kueri SQL dari datanya. Basis data akan mengompilasi struktur kueri terlebih dahulu, dan input pengguna kemudian dimasukkan hanya sebagai "data literal" atau parameter, bukan sebagai bagian dari perintah yang dapat dieksekusi. Dengan metode ini, meskipun penyerang memasukkan karakter berbahaya seperti ' OR 1=1 #, basis data akan menganggapnya sebagai string teks biasa dan tidak akan mengubah logika kueri [13].

Implementasi teknis perbaikan kode pada sisi server menggunakan PHP Data Objects (PDO) dapat dilihat pada Gambar 4.



Gambar 4. Implementasi *Prepared Statements* Menggunakan PHP PDO untuk Mencegah SQL Injection.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

Pada Gambar 4, penggunaan *placeholder* :id memastikan bahwa input yang diterima dari variabel $_GET['id'] diikat (*bound*) secara aman sebagai data. Selain itu, penerapan mekanisme enkripsi data sensitif juga direkomendasikan sebagai lapisan pertahanan terakhir jika *firewall* atau validasi input gagal.

3.2 Analisis Mendalam Serangan Stored Cross-Site Scripting (XSS)

3.2.1 Mekanisme Kerentanan

Stored Cross-Site Scripting (juga dikenal sebagai *Persistent XSS*) adalah jenis serangan XSS yang paling berbahaya. Berbeda dengan *Reflected XSS* yang bersifat sementara dan membutuhkan interaksi korban dengan tautan jahat, *Stored XSS* terjadi ketika *payload* berbahaya disimpan secara permanen di server target (biasanya dalam basis data, sistem komentar, forum, atau buku tamu). Setiap kali pengguna (korban) membuka halaman web yang memuat data tersebut, skrip berbahaya akan otomatis dimuat dan dieksekusi oleh peramban (*browser*) korban. Pada modul DVWA "Guestbook", aplikasi menerima input "Name" dan "Message" dari pengguna dan menyimpannya ke dalam basis data MySQL. Kemudian, aplikasi menampilkan kembali pesan-pesan tersebut ke halaman web agar bisa dibaca oleh pengunjung lain. Kerentanan muncul karena aplikasi tidak melakukan sanitasi terhadap input sebelum menyimpannya, dan juga tidak melakukan *encoding* saat menampilkannya kembali ke browser. Aplikasi secara buta mempercayai bahwa input pengguna hanyalah teks biasa, padahal input tersebut bisa berisi tag HTML atau skrip JavaScript.

Tabel 3. Karakteristik Payload Stored XSS

Vektor Serangan	Payload Uji	Lokasi Penyimpanan	Dampak Eksekusi
Input Script	<script>alert('XSS')</script>	Tabel MySQL: guestbook, Kolom: comment	Browser mengeksekusi JavaScript saat *rendering*.
Bypass Filter	Tidak ada filter	Server (Backend)	Data disimpan mentah (*raw*).
Output Echo	echo $row['comment'];	Browser (Frontend)	DOM HTML dimanipulasi dengan tag asing.

Analisis Data Serangan: Pada serangan XSS tersimpan, data serangan menjadi persisten. Ketika pengguna mengakses halaman, server mengirimkan respons HTTP yang mengandung kode berbahaya yang telah menyatu dengan kode asli (HTML Injection). Berikut adalah cuplikan kode sumber halaman (Page Source) yang terinfeksi:

Pola ini menunjukkan kegagalan server dalam melakukan encoding karakter spesial (< dan >) menjadi entitas HTML (< dan >).

3.2.2 Tahapan Eksploitasi

Peneliti menargetkan kolom "Message" untuk menyisipkan *payload*. Tujuannya adalah membuktikan bahwa browser dapat diperintah untuk mengeksekusi kode JavaScript arbitrer. *Payload* standar yang digunakan untuk *Proof of Concept* (PoC) adalah: <script>alert('Website ini Tidak Aman!');</script>
Analisis *payload*:

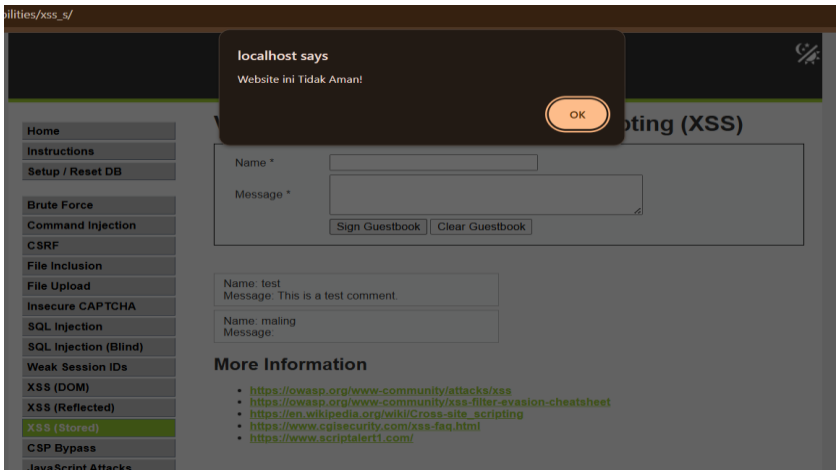
1. Tag <script>: Ini adalah tag HTML yang memberitahu peramban bahwa teks di antaranya adalah kode program (biasanya JavaScript) yang harus dieksekusi, bukan teks yang harus ditampilkan di layar.
2. Fungsi alert(): Fungsi bawaan JavaScript untuk memunculkan jendela *pop-up* peringatan. Ini adalah indikator visual paling sederhana untuk membuktikan keberhasilan XSS.
3. Tutup Tag </script>: Menandakan akhir dari blok kode program.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

Setelah peneliti menekan tombol "Sign Guestbook", data tersebut dikirim ke server dan disimpan ke tabel guestbook di basis data.

3.2.3 Hasil dan Dampak Eksploitasi

Segera setelah data tersimpan, aplikasi melakukan *refresh* halaman untuk menampilkan daftar pesan terbaru. Saat peramban memuat (render) halaman tersebut, ia menemukan tag <script> yang tadi disisipkan. Karena tidak ada mekanisme pengamanan, peramban mengeksekusi tag tersebut sebagaimana ia mengeksekusi kode JavaScript asli dari pengembang web.



Gambar 5. Tampilan Pop-up Alert Akibat Serangan Stored XSS

Gambar 5 menunjukkan munculnya *pop-up* dengan pesan "Website ini Tidak Aman!". Ini membuktikan bahwa serangan berhasil. Dampak dari *Stored XSS* sangat luas dan berbahaya. Dalam skenario nyata, penyerang tidak akan menggunakan alert(), melainkan skrip jahat yang tidak terlihat (*silent script*), seperti *Session Hijacking* (pencurian cookie sesi), *Phishing* (pengalihan halaman palsu), atau distribusi *Malware*. Bahaya utama *Stored XSS* adalah sifatnya yang permanen; satu kali injeksi bisa menyerang ribuan pengguna yang mengunjungi halaman tersebut tanpa mereka sadari [14].

3.2.4 Mitigasi dan Rekomendasi Perbaikan

Kunci utama mencegah XSS adalah prinsip *"Never Trust User Input"*. Pertahanan harus dilakukan berlapis, namun pertahanan paling krusial adalah pada saat data akan ditampilkan kembali ke pengguna, yang dikenal dengan istilah *Output Encoding*. Metode ini memastikan bahwa peramban (*browser*) menerjemahkan data sebagai teks biasa, bukan sebagai elemen HTML atau skrip yang dapat dieksekusi. Fungsi mitigasi bawaan yang paling efektif dan umum digunakan di PHP adalah htmlspecialchars(). Fungsi ini bertugas mengonversi karakter-karakter khusus yang memiliki makna sintaksis dalam HTML (seperti <, >, ', ") menjadi entitas HTML yang aman (seperti <, >, ', "). Contoh penerapan penulisan kode yang aman (*secure coding*) untuk menangani input pada buku tamu dapat dilihat pada Gambar 6 [15],.

PHP

```php
// Mengambil input (sebaiknya juga divalidasi)
$raw_message = $_POST['message'];

// Melakukan sanitasi sebelum ditampilkan (Output Encoding)
// ENT_QUOTES memastikan tanda petik satu dan dua ikut di-encode
// UTF-8 memastikan encoding karakter yang tepat
$safe_message = htmlspecialchars($raw_message, ENT_QUOTES, 'UTF-8');

// Menampilkan pesan yang sudah aman
echo "Pesan: " . $safe_message;
```

Gambar 6. Implementasi Sanitasi Output Menggunakan Fungsi htmlspecialchars untuk Mencegah XSS.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

Sebagaimana ditunjukkan pada Gambar 6, input mentah dari pengguna ($raw_message) diproses terlebih dahulu menggunakan htmlspecialchars dengan parameter ENT_QUOTES dan UTF-8 sebelum ditampilkan kembali dengan perintah echo. Dengan penerapan ini, jika penyerang memasukkan *payload* <script>alert(1)</script>, peramban akan menerima kode sumber yang telah dikonversi menjadi <script>alert(1)</script> dan menampilkannya hanya sebagai teks statis, sehingga serangan berhasil digagalkan.

4. KESIMPULAN

Penelitian ini telah berhasil melakukan analisis komprehensif terhadap kerentanan aplikasi web menggunakan metode *Penetration Testing* dengan standar OWASP pada target *Damn Vulnerable Web App* (DVWA). Berdasarkan serangkaian pengujian yang dilakukan pada tingkat keamanan rendah (*Low Security*), dapat disimpulkan bahwa aplikasi web yang dibangun tanpa mekanisme validasi input dan sanitasi output yang memadai memiliki risiko keamanan yang sangat kritis.

Hasil pengujian membuktikan bahwa celah keamanan *SQL Injection* memungkinkan penyerang tanpa otorisasi untuk memanipulasi kueri basis data. Hal ini berakibat fatal pada aspek kerahasiaan (*confidentiality*) data, di mana seluruh data pengguna dalam tabel basis data dapat diakses, dicuri, dan berpotensi disalahgunakan. Temuan ini menegaskan bahwa SQL Injection tetap menjadi salah satu ancaman terbesar bagi integritas data organisasi. Di sisi lain, celah keamanan *Stored Cross-Site Scripting* (XSS) terbukti memungkinkan penyerang untuk menanamkan kode berbahaya secara permanen di dalam aplikasi. Kerentanan ini mengancam keamanan pengguna aplikasi dari sisi klien, dengan potensi dampak mulai dari pencurian sesi (*session hijacking*) hingga penyebaran *malware*. Sifat persisten dari serangan ini menjadikannya sangat berbahaya karena dapat memakan banyak korban dalam waktu singkat.

Implikasi dari penelitian ini menunjukkan bahwa keamanan aplikasi web tidak dapat hanya bergantung pada infrastruktur jaringan seperti firewall semata [13]. Keamanan harus dibangun dari fondasi paling dasar, yaitu kode program (secure coding). Solusi teknis yang direkomendasikan dan terbukti efektif dalam penelitian ini adalah penerapan Prepared Statements untuk menanggulangi SQL Injection dan Output Encoding menggunakan fungsi htmlspecialchars untuk mencegah XSS. Kedua metode ini harus menjadi standar wajib dalam prosedur pengembangan perangkat lunak.

Keterbatasan penelitian ini terletak pada lingkup pengujian yang hanya dilakukan pada lingkungan lokal dengan tingkat keamanan rendah. Penelitian selanjutnya disarankan untuk memperluas cakupan pengujian pada tingkat keamanan yang lebih tinggi (Medium dan High) serta menggunakan kombinasi alat pengujian otomatis (automated tools) dan manual untuk mendapatkan hasil yang lebih holistik. Selain itu, analisis terhadap dampak penggunaan Web Application Firewall (WAF) dalam memitigasi serangan-serangan ini juga menjadi topik yang menarik untuk dieksplorasi di masa depan.

REFERENCES

[1] A. I. Rafeli, H. B. Seta, and I. W. Widi, "Pengujian Celah Keamanan Menggunakan Metode OWASP Web Security Testing Guide (WSTG)," *Inform. J. Ilmu Komput.*, vol. 18, no. 2, p. 97, 2022, doi: 10.52958/iftk.v18i2.4632.

[2] A. H. Alallah, M. Nasrullah, and M. I. Alhari, "Penetration Testing Pada Sebuah Website Perusahaan Education Development Dengan Framework Owasp Top-10," *J. Sist. Inf. dan Bisnis Cerdas*, vol. 18, no. 2, pp. 154–163, 2025, doi: 10.33005/sibc.v18i2.418.

[3] L. A. Nugraha, I. A. Kautsar, and A. S. Fitrani, "SQL Injection: Analisis Efektivitas Uji Penetrasi dalam Aplikasi Web," *Smatika J.*, vol. 14, no. 01, pp. 111–123, 2024, doi: 10.32664/smatika.v14i01.1224.

[4] N. C. Sari *et al.*, "Deteksi Kerentanan SQL Injection pada Website Menggunakan Vulnerability Assessment," *J. Data Insights*, vol. 2, no. 1, pp. 9–17, 2024, doi: 10.26714/jodi.v2i1.393.

[5] I. O. Riandhanu, "Analisis Metode Open Web Application Security Project (OWASP) Menggunakan Penetration Testing pada Keamanan Website Absensi," *J. Inf. dan Teknol.*, vol. 4, no. 3, pp. 160–165, 2022, doi: 10.37034/jidt.v4i3.236.

[6] M. Syarifudin, L. Widyawati, and O. Asroni, "Web Security Vulnerability Analysis and Mitigation Based on OWASP TOP 10," *J. Artif. Intell. Eng. Appl.*, vol. 4, no. 3, pp. 2808–4519, 2025, doi: 10.59934/jaiea.v4i3.1029.

[7] A. P. Armadhani, D. Nofriansyah, and K. Ibnutama, "Analisis Keamanan Untuk Mengetahui Vulnerability Pada DVWA Lab testing Menggunakan Penetration Testing Standart OWASP," *J. SAINTIKOM (Jurnal Sains Manaj. Inform. dan Komputer)*, vol. 21, no. 2, p. 80, 2022, doi: 10.53513/jis.v21i2.6119.

[8] L. F. Burhani and D. Priyawati, "Analisis Pengujian Keamanan Website Pengelolaan Internet Desa Kragan Menggunakan Metode Penetration Testing Execution Standard (PTES)," *JIPI (Jurnal Ilm. Penelit. dan Pembelajaran Inform.*, vol. 9, no. 1, pp. 307–319, 2024, doi: 10.29100/jipi.v9i1.4455.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026

[9] M. Tahir and M. Risky, “Analisis Keamanan Website Dinas Pemerintahan Yogyakarta Dengan Metode PTES (Penetration Testing Execution Standard),” *J. Tek. Inform. UNIKA ST.Thomas*, vol. 9, pp. 2657–1501, 2024, [Online]. Available: https://ejournal.ust.ac.id/index.php/JTIUST/article/view/3334

[10] M. Y. Firnanda, Henni Endah Wahanani, and Achmad Junaidi, “Website Security Testing Using PTES Method and OWASP Top 10 Approach,” *bit-Tech*, vol. 8, no. 1, pp. 404–415, 2025, doi: 10.32877/bt.v8i1.2564.

[11] N. D. Darno and E. L. Tjiong, “Analisis Celah Keamanan Pada Website PDDIKTI Menggunakan Metode Penetration Testing Dan Framework ISSAF,” *KalbiScientia, J. Sains dan Teknol.*, vol. 12, no. 02, pp. 1–13, 2025, doi: 10.53008/hnjgd446.

[12] A. S. Mizar Ismu Arief, Dede Syahrul Anwar, “Analisis Kerentanan Website Melalui Pendekatan Penetration Testing Berdasarkan Standart OWASP Top 10,” *J. ELEKTRO Inform. SWADHARMA*, vol. 05, no. 02, 2025, doi: 10.56486/jeis.vol5no2.798.

[13] W. Wahdana and K. H. Hanif, “Implementasi Keamanan Informasi Menggunakan Metode Web Application Firewall terhadap Serangan SQL Injection,” *J. Inform. Polinema*, vol. 11, no. 4, pp. 399–406, 2025, doi: 10.33795/jip.v11i4.7376.

[14] S. Hidayatulloh and D. Saptadiaji, “Penetration Testing pada Website Universitas ARS Menggunakan Open Web Application Security Project (OWASP),” *J. Algoritm.*, vol. 18, no. 1, pp. 77–86, 2021, doi: 10.33364/algoritma/v.18-1.827.

[15] F. S. Kristara and M. A. Adiguna, “Pengujian Celah Keamanan Input Validation Pada Aplikasi Website Menggunakan Framework OWASP,” *J. Penelit. Ilmu Komput.*, vol. 1, no. 4, pp. 50–55, 2023, doi: 10.62404/jupik.v1i4.66.

https://doi.org/10.33998/processor.2026.21.1.2686
Submitted: 06 Januari 2026; Reviewed: 08 Februari 2026; Accepted; 21 April 2026; Published: 30 April 2026