Jurnal Teknologi dan Sains Modern
E-ISSN: 3062-7966
DOI: https://doi.org/10.69930/jtsm.v3i4.857
Research Article

ScienceTech. Group

Vol. 3 (4), 2026
Hal. 274-285

Implementasi *Penetration Testing* untuk Mendeteksi Kerentanan XSS berbasis CLI (*Command Line Interface*)

Farhat Abdurrahman *,Dian Asmarajati, Iman Ahmad Ihsannuddin

Program studi Teknik Informatika, Fakultas Teknik dan Ilmu Komputer, Universitas Sains Al-Qur'an, Indonesia
*Email (Penulis Korespondensi): frhtabdrrhmn@mhs.unsiq.ac.id

Abstrak. *Ketergantungan layanan publik terhadap platform berbasis web turut meningkatkan risiko keamanan aplikasi, salah satunya kerentanan Cross-Site Scripting (XSS) yang menurut data Badan Siber dan Sandi Negara (BSSN) tercatat dalam jumlah anomali trafik siber yang tinggi di Indonesia sepanjang tahun 2023, sementara alat pemindai kerentanan berbasis antarmuka grafis (GUI) seperti OWASP ZAP masih membutuhkan konsumsi sumber daya yang besar dan kurang fleksibel untuk diintegrasikan ke dalam alur kerja otomasi seperti Continuous Integration/Continuous Deployment (CI/CD). Penelitian ini bertujuan merancang dan mengimplementasikan aplikasi pemindai kerentanan XSS berbasis Command-Line Interface (CLI) bernama xssniff menggunakan bahasa Python dengan metode black-box testing terhadap Damn Vulnerable Web Application (DVWA) pada tingkat keamanan Low, Medium, dan High. Aplikasi dibangun dengan arsitektur pipeline tujuh tahap yang meliputi Target Parser, Preflight, Parameter Mining dan Crawler, Reflection dan Context Analyzer, Payload Engine dan Encoder, WAF Fingerprinting, serta Verifier dan Reporter, dan dievaluasi menggunakan pendekatan analisis fungsional deskriptif. Hasil penelitian menunjukkan bahwa xssniff berhasil mendeteksi dan memverifikasi ketiga tipe XSS, yaitu Reflected, Stored, dan DOM-based, pada seluruh tingkat keamanan DVWA, termasuk keberhasilan melewati filter pada tingkat High melalui manipulasi atribut event handler, validasi Stored XSS melalui metode dua langkah POST kemudian GET, serta konfirmasi eksekusi DOM-based XSS tanpa hasil positif palsu menggunakan integrasi headless browser Playwright. Aplikasi juga menghasilkan keluaran JSON Envelope yang mendukung kebutuhan otomasi pengujian keamanan. Ketersediaan alat deteksi XSS yang ringan dan dapat diotomasi ini turut mendukung capaian Tujuan Pembangunan Berkelanjutan (SDGs), khususnya SDG 9 Target 9.1 mengenai penyediaan infrastruktur yang andal, SDG 16 Target 16.10 mengenai perlindungan akses publik terhadap informasi, dan SDG 8 mengenai keamanan data dalam ekonomi digital.*

Kata Kunci: *Cross-Site Scripting; Penetration Testing; Command-Line Interface; Black-Box Testing, DVWA*

Abstract. *Web application security has become increasingly critical as public services in sectors such as government, education, and banking continue to migrate to web-based platforms, a shift accompanied by the high number of cyber-traffic anomalies reported in Indonesia throughout 2023, while widely used scanners such as OWASP ZAP still depend on graphical interfaces that are resource-intensive and less flexible to integrate into automated workflows such as CI/CD pipelines. This study aims to design and implement an XSS vulnerability scanner named xssniff, built using the Python programming language with a black-box testing method against Damn Vulnerable Web Application (DVWA) at the Low, Medium, and High security levels. The application follows a seven-stage pipeline architecture consisting of Target Parser, Preflight, Parameter Mining and Crawler based on the Breadth-First Search algorithm, Reflection and Context Analyzer, a context-*

Received: 8 Juli 2026 / Accepted: 2 Agustus 2026 / Available online:4 Agustus 2026

aware Payload Engine and Encoder, WAF Fingerprinting, and a Verifier and Reporter, evaluated through a descriptive functional analysis approach. The results show that xssniff successfully detected and verified all three XSS types across every DVWA security level, including bypassing the High-level filter through HTML event-handler manipulation, validating Stored XSS through a two-step POST-then-GET method, and confirming DOM-based XSS execution without false positives using headless browser integration through Playwright. The tool also produces structured JSON Envelope output that distinguishes between Verified and Reflected-only findings to support security-testing automation needs. Beyond its technical contribution, the availability of a lightweight, CI/CD-ready XSS detection tool supports the Sustainable Development Goals (SDGs), particularly SDG 9 (Target 9.1) on reliable and resilient infrastructure, SDG 16 (Target 16.10) on the protection of public access to information, and SDG 8 on secure data handling within the digital economy. This research demonstrates that a Command-Line Interface (CLI)-based approach can serve as a lightweight and portable alternative to graphical-interface-based tools.

Keywords: *Cross-Site Scripting, Penetration Testing, Command-Line Interface, Black-Box Testing, DVWA*

1. Pendahuluan

Ketergantungan layanan publik terhadap platform berbasis web semakin meningkat seiring dengan transformasi digital di berbagai sektor, mulai dari pemerintahan, pendidikan, hingga perbankan. Kondisi ini secara bersamaan memperluas permukaan serangan (*attack surface*) yang dapat dieksploitasi, salah satunya melalui kerentanan *Cross-Site Scripting* (XSS). XSS terjadi ketika penyerang menyisipkan skrip berbahaya ke dalam halaman web yang kemudian dieksekusi oleh peramban pengguna akibat absennya validasi atau sanitasi masukan yang memadai, sehingga penyerang dapat mencuri *session cookie*, melakukan *phishing*, maupun memanipulasi tampilan halaman secara tersembunyi (Wibowo & Sulaksono, 2021). Serangan ini terbagi menjadi tiga tipe utama, yaitu *Reflected*, *Stored*, dan *DOM-based* XSS, yang masing-masing memiliki mekanisme eksploitasi berbeda sehingga memerlukan pendekatan deteksi yang komprehensif (Khazal & Hussain, 2021).

Urgensi permasalahan ini semakin nyata ketika ditinjau dari data keamanan siber nasional. Data Badan Siber dan Sandi Negara mencatat bahwa anomali trafik siber di Indonesia mencapai lebih dari 400 juta kejadian sepanjang tahun 2023, dengan sektor administrasi pemerintahan sebagai salah satu yang paling banyak terdampak (Badan Siber Dan Sandi Negara - Monitoring Keamanan Siber, 2024.), sebuah kondisi yang sejalan dengan kewajiban keamanan sistem elektronik yang diamanatkan melalui kebijakan *e-government* seperti Perpres No. 95 Tahun 2018 tentang Sistem Pemerintahan Berbasis Elektronik (Septiani et al., 2022). Secara global, laporan OWASP Top 10 edisi 2025 mengonsolidasikan XSS ke dalam kategori A05:2025-Injection yang tercatat ditemukan pada seluruh aplikasi yang masuk dalam data pengujian, dengan lebih dari 1.404.249 kemunculan dan lebih dari 30.000 CVE yang terdokumentasi khusus untuk XSS (A05 Injection - OWASP Top 10:2025). Pada tingkat implementasi nyata, penelitian (Rahman & Fatkhur Razak, 2024) menemukan bahwa situs PDDikti selaku platform data pendidikan tinggi nasional terbukti rentan terhadap XSS akibat celah pada lapisan autentikasi, sementara (Afrizal Ramadhan & Samsiar Ilmananda, 2024) mendokumentasikan delapan titik kerentanan XSS aktif pada sistem informasi akademik kampus melalui pengujian OWASP ZAP. Kedua temuan ini menegaskan bahwa XSS bukan sekadar ancaman teoretis, melainkan kerentanan yang secara nyata masih ditemukan pada layanan publik dan institusi pendidikan di Indonesia, sehingga kebutuhan terhadap alat deteksi yang dapat digunakan secara rutin dan berulang menjadi semakin mendesak.

Urgensi ini semakin diperkuat oleh keterbatasan alat *penetration testing* yang tersedia saat ini. Alat berbasis antarmuka grafis (GUI) yang populer seperti OWASP ZAP telah terbukti efektif mendeteksi berbagai kerentanan XSS pada aplikasi web nyata (Pramuja Inngam Fanani et al., 2025), namun ketergantungannya pada antarmuka grafis membuat alat semacam ini membutuhkan konsumsi sumber daya sistem yang relatif tinggi serta memerlukan interaksi manual yang tidak sejalan dengan tuntutan siklus pengembangan perangkat lunak modern yang serba cepat (Qadir et al., 2025). Pada praktik rekayasa perangkat lunak kontemporer, pengujian keamanan idealnya dijalankan secara otomatis pada setiap iterasi kode melalui *pipeline Continuous Integration/Continuous Deployment* (CI/CD), sebuah kebutuhan yang sulit dipenuhi oleh alat berbasis GUI karena tidak dirancang untuk dieksekusi tanpa interaksi manusia (*headless*) maupun untuk menghasilkan keluaran yang dapat diproses secara otomatis oleh sistem lain. Upaya menutup kesenjangan ini melalui pengembangan platform otomasi deteksi kerentanan yang lebih kompleks pun masih menghadapi keterbatasan portabilitas dan ketergantungan pada komponen layanan eksternal (Moreira et al., 2025). Kondisi ini menciptakan kebutuhan yang belum terpenuhi secara luas, yaitu sebuah alat deteksi XSS yang ringan, portabel, dan dapat dioperasikan langsung dari *Command-Line Interface* (CLI) tanpa ketergantungan pada antarmuka grafis maupun infrastruktur pendukung yang rumit.

Berdasarkan kesenjangan tersebut, penelitian ini bertujuan merancang dan mengimplementasikan aplikasi deteksi kerentanan XSS berbasis CLI bernama *xssniff* menggunakan bahasa pemrograman Python dengan metode *penetration testing* aktif melalui pendekatan *black-box testing*. Aplikasi dirancang untuk mampu mendeteksi dan memverifikasi tiga tipe XSS secara otomatis pada aplikasi web target, sekaligus menghasilkan keluaran yang dapat dimanfaatkan dalam kebutuhan otomasi pengujian keamanan.

Kebaruan penelitian ini terletak pada kombinasi arsitektur dan metodologi yang belum ditemukan secara bersamaan pada alat CLI XSS sejenis yang telah beredar secara luas, seperti *XSStrike* dan *Dalfox*. *XSStrike* merupakan pemindai XSS berbasis Python yang menyediakan kemampuan *fuzzing*, *crawling*, dan deteksi WAF, namun konfirmasi *DOM-based* XSS pada alat ini masih mengandalkan pembukaan *proof-of-concept* secara manual pada jendela peramban, bukan mekanisme verifikasi *headless* yang terintegrasi langsung ke dalam logika klasifikasi temuan (S0md3v/XSStrike: Most Advanced XSS Scanner). *Dalfox*, di sisi lain, dibangun menggunakan bahasa Go dan kini Rust, serta telah menyediakan opsi verifikasi *headless browser* dan deteksi ketiga tipe XSS (Hahwul/Dalfox: 🌙🦊 Dalfox Is a Powerful Open-Source XSS Scanner and Utility Focused on Automation). Dengan demikian, keunggulan *xssniff* pada penelitian ini bukan terletak pada klaim menjadi alat pertama yang mendukung verifikasi *headless*, melainkan pada implementasi *native* berbasis Python dengan arsitektur *pipeline* tujuh tahap yang bersifat modular dan bertanggung jawab tunggal pada tiap tahapnya, sehingga setiap tahap dapat diuji dan dievaluasi secara independen, serta pada penyediaan keluaran *JSON Envelope* yang memuat telemetri performa (waktu eksekusi, penggunaan CPU, dan memori puncak) per skenario pengujian, sebuah kombinasi yang belum secara eksplisit didokumentasikan pada kedua alat pembanding tersebut. Kebaruan ini menjadikan *xssniff* relevan khususnya bagi kalangan akademis dan praktisi yang beroperasi pada ekosistem Python serta membutuhkan artefak pengujian yang transparan dan dapat direproduksi untuk keperluan evaluasi fungsional deskriptif.

Kontribusi penelitian ini juga selaras dengan pencapaian Tujuan Pembangunan Berkelanjutan (*Sustainable Development Goals*/SDGs). Alat deteksi XSS berbasis CLI yang ringan dan dapat diintegrasikan ke dalam *pipeline* otomasi mendukung SDG 9 (Industri, Inovasi, dan Infrastruktur), khususnya Target 9.1 yang menekankan pengembangan infrastruktur yang berkualitas, andal, dan tangguh untuk mendukung pembangunan ekonomi dan kesejahteraan manusia, yang dalam konteks transformasi digital dapat dimaknai sebagai keandalan infrastruktur layanan berbasis web yang bebas dari eksploitasi kerentanan keamanan. Kemampuan alat dalam mengidentifikasi dan mencegah eksploitasi XSS turut mendukung SDG 16 (Perdamaian, Keadilan, dan Kelembagaan yang Tangguh), khususnya Target 16.10 mengenai jaminan akses publik terhadap informasi, karena eksploitasi XSS yang berhasil dapat digunakan untuk memanipulasi maupun mencuri data pada sistem informasi publik sehingga mengancam integritas akses informasi tersebut. Selain itu, tersedianya alat keamanan yang ringan dan terjangkau bagi pengembang skala kecil turut mendukung SDG 8 (Pekerjaan Layak dan Pertumbuhan Ekonomi) melalui penguatan perlindungan data dalam aktivitas ekonomi digital yang terus bertumbuh.

2. Metode

Penelitian ini menggunakan metode *black-box testing*, yaitu pendekatan pengujian yang mengevaluasi perilaku sistem semata-mata berdasarkan masukan dan keluarannya tanpa memerlukan pengetahuan mengenai struktur internal atau kode sumber sistem yang diuji (Mardian et al., 2021). Pelaksanaannya mengacu pada siklus operasional *penetration testing* yang terdiri dari empat tahapan, yaitu pengumpulan informasi awal (*reconnaissance*), pemindaian dan analisis kerentanan (*scanning and analysis*), eksekusi pengujian atau eksploitasi (*exploitation*), dan penyusunan laporan hasil temuan (*reporting*), yang kemudian diterjemahkan ke dalam arsitektur *pipeline* tujuh tahap pada aplikasi *xssniff* (Arfian et al., n.d.). Objek pengujian pada penelitian ini adalah *Damn Vulnerable Web Application* (DVWA), sebuah aplikasi web yang sengaja dirancang memiliki celah keamanan untuk keperluan pembelajaran dan pengujian keamanan siber secara legal. DVWA dijalankan sebagai kontainer Docker pada jaringan lokal (*localhost*) untuk mengeliminasi variabel latensi jaringan eksternal, dan diuji pada tiga tingkat keamanan, yaitu *Low*, *Medium*, dan *High*, guna mengevaluasi ketangguhan alat dalam menghadapi variasi mekanisme sanitasi masukan.

Aplikasi *xssniff* dibangun menggunakan Python versi 3.11 ke atas dengan memanfaatkan pustaka *requests* untuk komunikasi HTTP, *BeautifulSoup* dan *lxml* untuk *parsing* struktur DOM, *Playwright* untuk verifikasi *headless browser*, *concurrent.futures* untuk pengelolaan *concurrency*, serta *rich* dan *argparse* untuk penyusunan antarmuka CLI. Arsitektur sistem disusun dalam alur kerja *pipeline* tujuh tahap. Tahap pertama, *Target Parser*, menormalisasi input URL dan parameter pengguna menjadi objek target terstruktur. Tahap kedua, *Preflight*, memvalidasi status hidup target beserta kebijakan keamanan respons sebelum pengujian dimulai. Tahap ketiga, *Parameter Mining* dan *Crawler*, menjalankan penjelajahan otomatis berbasis algoritma *Breadth-First Search* untuk memetakan tautan dan formulir yang tersedia, dilengkapi mekanisme proteksi sesi agar tidak mengakses tautan destruktif. Tahap keempat, *Reflection* dan *Context Analyzer*, menyisipkan penanda unik guna mendeteksi area pantulan serta mengklasifikasikan konteks injeksi pada struktur DOM. Tahap kelima, *Payload Engine* dan *Encoder*, memilih variasi *payload* secara *context-aware* dan menyandikannya apabila filter keamanan terdeteksi. Tahap keenam, WAF *Fingerprinting*,

mengidentifikasi indikasi keberadaan *Web Application Firewall* melalui analisis sinyal pada respons HTTP, seperti kode status tertentu dan pola teks spesifik vendor WAF, mengingat penelitian terkini menunjukkan bahwa WAF berbasis kecocokan pola (*signature matching*) memiliki keterbatasan dalam menghadapi *payload* yang dimodifikasi secara sintaksis (Akhavani et al., 2025), sebagaimana juga ditunjukkan oleh *payload* XSS hasil sintesis otomatis yang mampu melewati ModSecurity WAF pada sebagian besar percobaan (Babaey & Ravindran, 2025). Tahap ketujuh, *Verifier* dan *Reporter*, membuktikan apakah *payload* benar-benar membentuk struktur DOM yang dapat dieksekusi, termasuk melalui konfirmasi *headless browser*, sebelum merangkum hasil akhir.

Rancangan skenario pengujian disusun berdasarkan kombinasi tiga variabel, yaitu tipe XSS (*Reflected*, *Stored*, dan *DOM-based*), tingkat keamanan DVWA (*Low*, *Medium*, *High*), serta titik masukan yang menjadi target pada masing-masing modul, yaitu parameter *name* pada modul *xss_r* untuk *Reflected* XSS, *field txtName* dan *mtxMessage* pada modul *xss_s* untuk *Stored* XSS, serta parameter *default* pada modul *xss_d* untuk *DOM-based* XSS. Setiap kombinasi skenario dijalankan sebagai satu sesi pemindaian independen agar hasil deteksi pada satu tingkat keamanan tidak tercampur dengan tingkat keamanan lainnya. Basis *payload* yang digunakan bersumber dari OWASP Testing Guide v4.2 dan PortSwigger XSS Cheat Sheet, yang setelah dikurasi menghasilkan himpunan data sebanyak 7.240 baris *payload* mentah yang dikategorikan ke dalam tiga kelompok vektor, yaitu *payload* berbasis tag <script>, *payload* berbasis atribut *event handler* seperti *onerror* dan *onload*, serta *payload* berbasis manipulasi DOM seperti skema *javascript:* URI. Pengelompokan ini menjadi dasar bagi mekanisme *filter-bypass* adaptif pada *Payload Engine*, yang secara otomatis beralih ke kelompok vektor lain apabila satu kelompok vektor telah dinetralkan oleh mekanisme penyaringan input pada tingkat keamanan tertentu.

Untuk memastikan bahwa proses pengujian tidak merusak sistem target, penelitian ini menerapkan beberapa mekanisme pengamanan. Pertama, seluruh pengujian dijalankan secara eksklusif terhadap instans DVWA yang berjalan pada kontainer Docker lokal dan terisolasi dari jaringan produksi, sehingga tidak ada risiko terhadap sistem pihak ketiga. Kedua, aplikasi mewajibkan penyertaan argumen otorisasi (*--i-am-authorized*) pada setiap eksekusi sebagai bentuk konfirmasi eksplisit bahwa pengujian dilakukan pada target yang sah dan diizinkan, sejalan dengan prinsip kehati-hatian dalam praktik *penetration testing* yang bertanggung jawab (Tedyyana & Ghazali, 2021). Ketiga, modul *Parameter Mining* dan *Crawler* menerapkan tiga lapis proteksi sesi, yaitu validasi keabsahan sesi sebelum penjelajahan dimulai, penerapan daftar tolak (*blacklist*) terhadap tautan yang bersifat destruktif seperti tautan *logout* atau reset data, serta deteksi indikasi berakhirnya sesi di tengah proses penjelajahan, sehingga proses *crawling* tidak merusak kondisi autentikasi yang sedang digunakan. Keempat, jumlah *worker* pada mekanisme *concurrent.futures.ThreadPoolExecutor* dibatasi secara terkontrol beserta jeda antarpermintaan, mengingat pengiriman permintaan secara simultan dalam jumlah besar berpotensi membebani server target secara berlebihan apabila tidak diatur dengan baik (Tedyyana & Ghazali, 2021). Kelima, khusus pada pengujian *Stored* XSS yang bersifat akumulatif, basis data *guestbook* DVWA disegarkan (*reset*) sebelum pengujian pada setiap tingkat keamanan dimulai, untuk mencegah *payload* yang tersimpan dari tingkat keamanan sebelumnya mencemari hasil pengujian pada tingkat keamanan berikutnya.

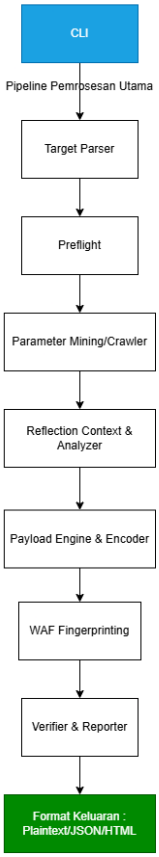
Evaluasi terhadap kemampuan deteksi alat dilakukan menggunakan pendekatan analisis fungsional deskriptif, yaitu penilaian berdasarkan kesesuaian antara output yang dihasilkan alat dengan kondisi nyata pada target, tanpa melibatkan pengukuran metrik statistik seperti *true positive rate* atau *false positive rate* (Ramadhany & Eviyanti, 2021). Setiap skenario pengujian dicatat dengan status biner, yaitu *Detected* apabila kerentanan yang memang ada berhasil diidentifikasi, atau *Not Detected* apabila alat gagal mendeteksinya, sehingga hasil pengujian dapat disajikan secara objektif dan mudah direplikasi.

3. Hasil dan Pembahasan

3.1. Implementasi Antarmuka CLI dan Realisasi Arsitektur Pipeline

Aplikasi *xssniff* berhasil direalisasikan sebagai alat berbasis *CLI* yang dapat dikonfigurasi secara ekstensif melalui sejumlah argumen, di antaranya penentuan *parameter* target, penyertaan *cookie* sesi, pemilihan tipe pengujian (*Reflected*, *Stored*, atau *dom*), pengaktifan verifikasi *headless*, serta pengaturan format keluaran. Aplikasi juga menerapkan sistem *exit code* standar yang memungkinkan integrasi langsung ke dalam *pipeline* CI/CD: *exit code* 0 menandakan tidak ditemukannya kerentanan, *exit code* 1 menandakan adanya temuan yang terverifikasi sehingga dapat digunakan oleh sistem CI/CD untuk menggagalkan proses *build* secara otomatis guna mencegah publikasi kode yang rentan, dan *exit code* 2 menandakan kesalahan internal maupun pengabaian argumen otorisasi pengujian.

Realisasi teknis dari arsitektur *pipeline* tujuh tahap yang telah diuraikan pada bagian Metode ditunjukkan secara visual pada Gambar 1. Diagram tersebut menggambarkan alur data yang bersifat sekuensial dan modular, mulai dari tahap *Target Parser* yang menormalisasi input pengguna, dilanjutkan dengan *Preflight*, *Parameter Mining* dan *Crawler*, *Reflection* dan *Context Analyzer*, *Payload Engine* dan *Encoder*, WAF *Fingerprinting*, hingga tahap akhir *Verifier* dan *Reporter* yang menentukan status akhir temuan.



Gambar 1. Diagram arsitektur pipeline pemindaian tujuh tahap pada *xssniff*

Sebagai bentuk dukungan terhadap kebutuhan integrasi *machine-to-machine*, aplikasi juga menghasilkan keluaran terstruktur dalam format *JSON Envelope* sebagaimana diilustrasikan pada Gambar 2, yang merangkum metadata performa eksekusi (waktu eksekusi, penggunaan CPU, dan memori puncak), konteks kerentanan spesifik, serta rincian bukti verifikasi pada setiap skenario pengujian.

```
{
  "meta": {
    "xssniff_version": "0.1.0",
    "targets": [
      "http://localhost/DVWA-master/vulnerabilities/xss_d/?default=English"
    ],
    "started_at": "2026-06-28T07:11:47.310486+00:00",
    "duration_ms": 1610,
    "total_findings": 1,
    "total_scenarios": 1,
    "total_detected": 1,
    "total_poc": 1,
    "execution_time_s": 2.806146144869434,
    "cpu_avg_percent": 22.04182098765432,
    "peak_memory_mb": 53.84375,
    "total_suspected": 0,
    "metrics": {
      "execution_time_s": 2.806146144869434,
      "cpu_avg_percent": 22.04182098765432,
      "peak_memory_mb": 53.84375,
      "total_scenarios": 1,
      "total_detected": 1
    },
    "runs": [
      {
        "execution_time_s": 5.320821523666382,
        "cpu_avg_percent": 20.062962962962963,
        "peak_memory_mb": 53.3203125,
        "total_scenarios": 1,
        "total_detected": 1
      },
      {
        "execution_time_s": 1.547080785369873,
        "cpu_avg_percent": 21.82%,
        "peak_memory_mb": 53.85546875,
        "total_scenarios": 1,
        "total_detected": 1
      },
      {
        "execution_time_s": 2.806146144869434,
        "cpu_avg_percent": 22.04182098765432,
        "peak_memory_mb": 53.84375,
        "total_scenarios": 1,
        "total_detected": 1
      }
    ],
    "runs_summary": {
      "runs": 3,
      "execution_time_s_stddev": 2.177772917591011,
      "cpu_avg_percent_stddev": 2.095697586464716?,
      "peak_memory_mb_stddev": 0.5176776142588044
    },
    "security_level": "low"
  },
  "findings": [
    {
      "type": "V",
      "type_description": "DOM XSS verified — browser dialog terpicu oleh headless browser",
      "context": "DOM",
      "parameter": "default#query",
      "method": "GET",
      "payload": "'></option></select><img src=x onerror=alert(1)>",
      "poc_url": "http://localhost/DVWA-master/vulnerabilities/xss_d/?default='></option></select><img src=x onerror=alert(1)>",
      "evidence": "Source: document.location, location.href\nSink   : document.write(\nPoC    : http://localhost/DVWA-master/vulnerabilities/xss_d/?default='></option></select><img src=x onerror=alert(1)>",
      "verified": true,
      "verification": "Browser dialog executed: alert, init_script",
      "status": "Detected",
      "timestamp": "2026-06-28T07:11:48.716640+00:00",
      "duration_ms": 879,
      "clickable_poc": "http://localhost/DVWA-master/vulnerabilities/xss_d/?default='></option></select><img src=x onerror=alert(1)>",
      "clickable_note": "",
      "verified_by": "headless"
    }
  ],
  "scenario_statuses": [
    {
      "xss_type": "DOM",
      "entry_point": "default(GET)",
      "status": "Detected",
      "poc_count": 1
    }
  ],
  "warnings": []
}
```


Gambar 2. Contoh format keluaran terstruktur (JSON Envelope) yang merangkum hasil akhir pemindaian

3.2. Analisis Fungsional *Reflected* XSS

Pengujian *Reflected* XSS dilaksanakan pada parameter *name* milik modul *xss_r* DVWA melalui metode GET. Tabel 1 merangkum hasil rekapitulasi pengujian pada ketiga tingkat keamanan.

Tabel 1. Rekapitulasi Hasil Pengujian *Reflected* XSS

Security Level	Parameter	Jumlah PoC [V] Verified	Status
Low	name	37	Detected
Medium	name	37	Detected
High	name	33	Detected

Mekanisme pembuktian temuan pada *xssniff* tidak berhenti pada kemunculan tekstual *payload* di dalam respons HTTP, melainkan ditelusuri lebih lanjut melalui pemeriksaan struktur DOM hasil *parsing* terhadap respons tersebut. Pada tingkat keamanan *Low* dan *Medium*, *payload* berbasis tag <script>, misalnya <script>alert(1)</script>, serta *payload* berbasis atribut *event handler* seperti <svg onload=alert(1)> yang disisipkan pada parameter

name dikembalikan oleh server tanpa mengalami transformasi karakter apa pun, sehingga struktur markup yang dihasilkan tetap utuh dan dapat dieksekusi. Pada tingkat keamanan *High*, ditemukan bahwa seluruh *payload* berbasis tag <script> secara konsisten mengalami transformasi menjadi entitas HTML (<script>) sehingga tergolong *Reflected*-only. Meskipun demikian, jumlah temuan *Verified* pada Tabel 1 tetap signifikan pada tingkat *High*, dengan 33 dari keseluruhan *payload* yang diuji tetap berhasil terverifikasi berkat mekanisme *filter-bypass* adaptif yang secara otomatis beralih ke *payload* berbasis atribut *event handler*. Contoh konkret yang berhasil menembus tingkat *High* adalah <img src=x onerror=alert(1)>, yang disisipkan pada konteks *inAttr* (di dalam nilai atribut HTML yang telah ada), karena mekanisme penyaringan pada tingkat tersebut secara spesifik menyasar transformasi karakter pembentuk tag pembuka dan penutup, namun belum menjangkau seluruh kombinasi atribut *event handler* yang dapat disisipkan melalui teknik pemutusan konteks atribut (*attribute breakout*). Temuan pada Tabel 1 ini menegaskan bahwa pendekatan *context-aware payload engine* lebih unggul dibandingkan pendekatan pengujian dengan satu jenis *payload* statis untuk seluruh skenario (Qadir et al., 2025).

3.3. Analisis Fingsionalitas *Stored* XSS

Pengujian *Stored* XSS dilaksanakan pada formulir *guestbook* modul *xss_s* DVWA melalui metode POST, dengan *payload* disisipkan pada *field txtName* dan *mtxMessage*. *xssniff* menerapkan metodologi pengujian dua langkah, yaitu pengiriman *payload* melalui permintaan POST yang kemudian diikuti dengan pengambilan ulang halaman melalui permintaan GET untuk memastikan *payload* benar-benar tersimpan pada basis data target, bukan sekadar muncul pada respons konfirmasi sementara. Hasil rekapitulasi pengujian pada kedua *field* dan ketiga tingkat keamanan dirangkum pada Tabel 2.

Tabel 2. Rekapitulasi Hasil Pengujian *Stored* XSS

Security Level	Field	Jumlah PoC [V] Verified	Status
Low	txtName	24	Detected
Low	mtxMessage	24	Detected
Medium	txtName	22	Detected
Medium	mtxMessage	22	Detected
High	txtName	20	Detected
High	mtxMessage	20	Detected

Data pada Tabel 2 memperlihatkan status *Detected* yang konsisten pada kedua *field* di seluruh tingkat keamanan, meskipun jumlah temuan *Verified* menurun seiring meningkatnya tingkat keamanan. Pada tingkat *Medium*, *payload* berbasis tag <script>, seperti <script>alert(1)</script>, mengalami penyaringan sehingga muncul kembali dalam bentuk teks biasa tanpa markup aktif dan tergolong *Reflected*-only. Pada tingkat *High*, cakupan penyaringan diperluas hingga turut menyasar skema *javascript:* URI, sehingga *payload* seperti <iframe src=javascript:alert(1)> beserta variasi kapitalisasinya turut tereliminasi. Meskipun demikian, *payload* berbasis atribut *event handler*, yaitu <img src=x onerror=alert(1)> dan <body onload=alert(1)>, secara konsisten tetap tembus dan tersimpan tanpa transformasi karakter apa pun pada ketiga tingkat keamanan, sebagaimana tercermin dari selisih jumlah temuan

antartingkat pada Tabel 2. Temuan ini membuktikan bahwa mekanisme penyaringan input pada DVWA *Stored* XSS bersifat selektif terhadap vektor tertentu, yaitu lebih memfokuskan penyaringan pada tag <script> dan skema *javascript:* URI, namun belum menjangkau seluruh permukaan vektor XSS berbasis atribut *event handler*.

3.4. Analisis Fungsionalitas *DOM-based* XSS

Pengujian *DOM-based* XSS dilaksanakan pada parameter *default* modul *xss_d* DVWA dengan verifikasi *headless browser* diaktifkan melalui *Playwright* untuk mengonfirmasi eksekusi nyata *payload* pada peramban *Chromium*. Hasil rekapitulasi pengujian disajikan pada Tabel 3.

Tabel 3. Rekapitulasi Hasil Pengujian *DOM-based* XSS

Security Level	Lokasi Injeksi yang Berhasil	Versifikasi Headless	Jumlah PoC [V] Verified	Status
Low	Query string (?default=)	Ya	1	Detected
Medium	Query string (?default=)	Ya	1	Detected
High	Fragment URL (#default=)	Ya	1	Detected

Lokasi injeksi pada Tabel 3 merujuk pada bagian URL tempat *payload* disisipkan, yaitu parameter *query string* (?default=...) untuk tingkat *Low* dan *Medium*, serta *fragment* URL (#default=...) untuk tingkat *High*. Sebagaimana ditunjukkan pada Tabel 3, *xssniff* berhasil mencapai status *Detected* pada ketiga tingkat keamanan tanpa satu pun temuan positif palsu. Pendekatan verifikasi dinamis yang mengombinasikan analisis *source-sink* statis dengan konfirmasi eksekusi nyata pada mesin JavaScript peramban dapat menurunkan tingkat *false positive* dibandingkan metode berbasis analisis *string* murni (Hannousse et al., 2024). Temuan yang secara arsitektural signifikan pada Tabel 3 adalah perbedaan lokasi injeksi yang berhasil antara tingkat *Low* dan *Medium* yang tereksekusi melalui parameter *query string*, dengan tingkat *High* yang hanya tereksekusi melalui *fragment* URL. Perbedaan ini terjadi karena bagian *fragment* pada suatu URL secara teknis tidak pernah dikirimkan ke server, sehingga validasi pada sisi server tidak dapat menjangkau jalur tersebut. *Payload* yang berhasil diverifikasi pada tingkat *High* adalah '></option></select><img src=x onerror=alert(1)>, yang memutus konteks atribut *value* pada elemen <option> sebelum menyisipkan elemen <img> dengan *event handler onerror*; keberhasilan eksekusinya dikonfirmasi langsung oleh modul *dom_scanner.py* melalui penyadapan (*hooking*) terhadap peristiwa dialog *alert()* pada peramban *Chromium*, bukan sekadar asumsi berdasarkan analisis teks statis.

Kesimpulan

Penelitian ini berhasil menjawab kedua tujuan yang telah dirumuskan. Pertama, aplikasi pemindai kerentanan *Cross-Site Scripting* berbasis *CLI* bernama *xssniff* berhasil dirancang dan diimplementasikan menggunakan bahasa pemrograman Python dengan

arsitektur *pipeline* tujuh tahap yang mencakup *Target Parser*, *Preflight*, *Parameter Mining* dan *Crawler*, *Reflection* dan *Context Analyzer*, *Payload Engine* dan *Encoder*, WAF *Fingerprinting*, serta *Verifier* dan *Reporter*. Kedua, proses identifikasi dan klasifikasi kerentanan XSS berhasil dilakukan secara otomatis oleh aplikasi melalui analisis respons HTTP dan verifikasi dinamis, dengan hasil pengujian pada *Damn Vulnerable Web Application* menunjukkan bahwa aplikasi mampu mendeteksi dan memverifikasi ketiga tipe XSS, yaitu *Reflected*, *Stored*, dan *DOM-based*, secara konsisten pada tingkat keamanan *Low*, *Medium*, dan *High*, termasuk keberhasilan melewati mekanisme filter pada tingkat *High* melalui manipulasi atribut *event handler*, validasi *Stored* XSS melalui metode dua langkah POST kemudian GET, serta konfirmasi eksekusi *DOM-based* XSS tanpa hasil positif palsu menggunakan integrasi *headless browser Playwright*. Dengan demikian, dapat disimpulkan bahwa pendekatan berbasis CLI yang dikembangkan pada penelitian ini terbukti mampu menjadi alternatif yang ringan, portabel, dan dapat diintegrasikan ke dalam alur otomasi pengujian keamanan dibandingkan alat berbasis antarmuka grafis yang selama ini banyak digunakan, sekaligus memberikan kontribusi nyata terhadap penguatan keamanan infrastruktur digital yang selaras dengan capaian Tujuan Pembangunan Berkelanjutan.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Fakultas Teknik dan Ilmu Komputer, Universitas Sains Al-Qur'an, atas dukungan fasilitas dan lingkungan akademik yang diberikan selama proses penelitian ini berlangsung. Ucapan terima kasih juga disampaikan kepada Dosen Pembimbing atas bimbingan, arahan, dan masukan yang sangat berarti selama penyusunan penelitian, serta kepada seluruh pihak yang telah membantu kelancaran pelaksanaan penelitian ini.

Daftar Pustaka

A05 Injection - OWASP Top 10:2025. (n.d.). Retrieved May 11, 2026, from https://owasp.org/Top10/2025/A05_2025-Injection/

Afrizal Ramadhan, M. F., & Samsiar Ilmananda, A. (2024). ANALISIS ANCAMAN KEAMANAN PADA SISTEM INFORMASI AKADEMIK KAMPUS MENGGUNAKAN METODE OWASP ZAP. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *8*(4), 7985–7991. https://doi.org/10.36040/jati.v8i4.10599

Akhavani, S. A., Jabiyev, B., Kallus, B., Topcuoglu, C., Bratus, S., & Kirda, E. (2025). *WAFFLED: Exploiting Parsing Discrepancies to Bypass Web Application Firewalls*. https://doi.org/10.1109/ACSAC67867.2025.00062

Arfian, H., Sunupurwa Asri, J., Tjahjono, B., Septian, F., & Hadi Arfian, M. (n.d.). *Pengujian Keamanan Website dengan Metode Penetration Testing (Studi Kasus: Universitas Esa Unggul)*.

Babaey, V., & Ravindran, A. (2025). GenXSS: An AI-Driven Framework for Automated Detection of XSS Attacks in WAFs. *Conference Proceedings - IEEE SOUTHEASTCON*, 1519–1524. https://doi.org/10.1109/SOUTHEASTCON56624.2025.10971558

Badan Siber dan Sandi Negara - Monitoring Keamanan Siber. (n.d.). Retrieved July 29, 2026, from https://www.bssn.go.id/monitoring-keamanan-siber/

hahwul/dalfox: 🌙🦊 Dalfox is a powerful open-source XSS scanner and utility focused on automation. (n.d.). Retrieved July 29, 2026, from https://github.com/hahwul/dalfox

Khazal, I. F., & Hussain, M. A. (2021). Server Side Method to Detect and Prevent Stored XSS Attack. *Iraqi Journal for Electrical and Electronic Engineering*, *17*(2), 58–65. https://doi.org/10.37917/ijeee.17.2.8

Mardian, A., Budiman, T., Haroen, R., & Yasin, V. (2021). PERANCANGAN APLIKASI PEMANTAUAN KINERJA KARYAWAN BERBASIS ANDROID DI PT. SALESTRADE CORP. INDONESIA. *Jurnal Manajamen Informatika Jayakarta*, *1*(3), 169. https://doi.org/10.52362/jmijayakarta.v1i3.481

Moreira, D., Seara, J. P., Pavia, J. P., & Serrão, C. (2025). Intelligent Platform for Automating Vulnerability Detection in Web Applications. *Electronics (Switzerland)*, *14*(1). https://doi.org/10.3390/electronics14010079

Pramuja Inngam Fanani, G., Muhammad Amirul Mu'min, & Tristanti, N. (2025). Analisis dan Pengujian Kerentanan Website Menggunakan OWASP ZAP. *Jurnal Riset Sistem Dan Teknologi Informasi*, *3*(1), 36–50. https://doi.org/10.30787/restia.v3i1.1886

Qadir, S., Waheed, E., Khanum, A., & Jehan, S. (2025). Comparative evaluation of approaches & tools for effective security testing of Web applications. *PeerJ Computer Science*, *11*, 1–42. https://doi.org/10.7717/peerj-cs.2821

Rahman, R., & Fatkhur Razak, D. (n.d.). *JURNAL RISET SISTEM INFORMASI*. https://doi.org/10.69714/e4rhmk70

Ramadhany, S. N., & Eviyanti, A. (2021). Designing Web Based Offering and Sales Information System (Case Study : PT. Daya Berkah Sentosa Nusantara). *Procedia of Engineering and Life Science*, *1*(2). https://doi.org/10.21070/pels.v1i2.1030

s0md3v/XSStrike: Most advanced XSS scanner. (n.d.). Retrieved July 29, 2026, from https://github.com/s0md3v/XSStrike

Septiani, A., Syamsir, S., Aulia, A. R., Resti, A., Fazira, V., Sukma Wijaya, D. A., & Aldeo, Z. (2022). Peranan E-Government Dalam Pelayanan Publik. *JURNAL SYNTAX IMPERATIF : Jurnal Ilmu Sosial Dan Pendidikan*, *3*(5), 302. https://doi.org/10.36418/syntax-imperatif.v3i5.183

Tedyyana, A., & Ghazali, O. (2021). Teler Real-time HTTP Intrusion Detection at Website with Nginx Web Server. *JOIV : International Journal on Informatics Visualization*, *5*(3), 327–332. https://doi.org/10.30630/JOIV.5.3.510

Wibowo, R. M., & Sulaksono, A. (2021). Web Vulnerability Through Cross Site Scripting (XSS) Detection with OWASP Security Shepherd. *Indonesian Journal of Information Systems*, 149–159. https://doi.org/10.24002/ijis.v3i2.4192