MJI

Media Jurnal Informatika
Vol. 17 No 1, Juni 2025
http://jurnal.unsur.ac.id/mjinformatika

p-issn : 2088-2114
e-issn : 2477-2542

Implementasi Dan Pengujian Teknik Preventif Cross-Site Scripting (Xss) Pada Aplikasi Php Berdasarkan Xsstrike

Juri Pebrianto[1], Riky Susanto[2], Tri Prasetyo[3]
[1,2,3]*Program Studi Teknik Informatika, Universitas Pamulang*
juripebrianto@gmail.com[1], rikycloud@gmail.com[2], prasetyotri295@gmail.com[3]

Abstract

Cross-Site Scripting (XSS) is one of the most common security threats in web applications, especially in native PHP-based applications that have not implemented modern security standards. This research aims to evaluate the effectiveness of preventive techniques against XSS attacks using the XSStrike automated tool. The research method used was an experimental approach, where vulnerable PHP applications were tested against three types of XSS attacks: Reflected, Stored, and DOM-Based, both automatically and manually. The test results show that Reflected XSS attacks can be detected with high confidence by XSStrike, while Stored and DOM-Based XSS require manual testing techniques due to tool limitations. After the implementation of input validation, output encoding, and Content Security Policy (CSP), the script injection is no longer executed by the browser and is only displayed as plain text. The conclusion of this research is that implementing a combination of preventive techniques can significantly improve the resilience of native PHP applications against various forms of XSS attacks.

Keywords: Cross-Site Scripting, XSStrike, PHP Native, Web Security, CSP

Abstrak

Cross-Site Scripting (XSS) merupakan salah satu ancaman keamanan paling umum pada aplikasi web, khususnya pada aplikasi berbasis PHP Native yang belum menerapkan standar keamanan modern. Penelitian ini bertujuan untuk mengevaluasi efektivitas teknik preventif terhadap serangan XSS dengan menggunakan alat otomatis XSStrike. Metode penelitian yang digunakan adalah pendekatan eksperimen, di mana aplikasi PHP rentan diuji terhadap tiga jenis serangan XSS: Reflected, Stored, dan DOM-Based, baik secara otomatis maupun manual. Hasil pengujian menunjukkan bahwa serangan Reflected XSS dapat terdeteksi dengan confidence tinggi oleh XSStrike, sedangkan Stored dan DOM-Based XSS memerlukan teknik pengujian manual karena keterbatasan alat. Setelah implementasi validasi input, encoding output, dan Content Security Policy (CSP), injeksi skrip tidak lagi dieksekusi oleh browser dan hanya ditampilkan sebagai teks biasa. Kesimpulan dari penelitian ini adalah bahwa penerapan kombinasi teknik preventif secara signifikan dapat meningkatkan ketahanan aplikasi PHP Native terhadap berbagai bentuk serangan XSS.

Kata kunci: Cross-Site Scripting, XSStrike, PHP Native, Keamanan Web, CSP

I. PENDAHULUAN

Fenomena penggunaan AI, seperti Chat GPT dan alat-alat generatif lainnya, semakin marak digunakan oleh mahasiswa dan programmer pemula untuk menghasilkan kode. Dalam banyak kasus, alat ini mampu membuat kode yang sesuai dengan prompt yang diberikan. Namun, jika prompt tidak spesifik atau mengabaikan aspek keamanan, kode yang dihasilkan rentan terhadap berbagai celah keamanan, termasuk XSS. Studi menunjukkan bahwa penggunaan alat seperti Chat GPT dapat menghasilkan kode PHP yang rentan terhadap serangan XSS, terutama jika pengguna kurang memahami praktik pengkodean yang aman atau hanya mengandalkan hasil generasi tanpa validasi lebih lanjut.

Penelitian oleh Oh enunjukkan bahwa AI-powered coding tools dapat mengarahkan pengguna pada penggunaan kode yang tidak aman ketika prompt mereka kurang spesifik. Hal ini diperparah dengan rendahnya kesadaran pengguna akan risiko keamanan terkait kode generatif dari AI [1].

Aplikasi berbasis PHP (Hypertext Preprocessor), yang banyak digunakan untuk pengembangan web [2], rentan terhadap serangan XSS [3][4][5][6], ditambah banyaknya pengguna PHP pemula sering meminta bantuan Chat GPT untuk mengenerate kan skrip kode PHP, ini dikemukakan oleh T'oth menemukan bahwa dari 2.500 situs web berbasis PHP yang dihasilkan oleh GPT-4, sebanyak 26% mengandung setidaknya satu kerentanan yang dapat dieksploitasi, termasuk XSS. Hal ini menunjukkan bahwa ketergantungan pada kode AI tanpa uji keamanan tambahan dapat membawa risiko signifikan [7]

Selain itu, studi oleh Khoury nunjukkan bahwa meskipun ChatGPT menyadari potensi kerentanan dalam kode yang dihasilkan, kode yang dihasilkan tetap sering tidak tahan terhadap serangan seperti XSS. Penelitian ini juga mencatat bahwa pengujian tambahan dan prompt

yang lebih spesifik dapat membantu mengurangi risiko ini [8].

Secara keseluruhan, fenomena ini menyoroti pentingnya edukasi tentang keamanan dalam pengkodean dan perlunya pengujian menyeluruh pada kode yang dihasilkan oleh AI, terutama bagi pengguna pemula yang kurang berpengalaman.

XSS merupakan salah satu bentuk serangan siber yang paling umum dan berbahaya terhadap aplikasi web. Serangan ini memungkinkan penyerang untuk menyisipkan skrip berbahaya ke dalam halaman web yang diakses oleh pengguna, dengan tujuan untuk mencuri informasi sensitif, seperti cookie, data sesi, atau informasi pribadi lainnya, serta melakukan aktivitas jahat lainnya seperti phishing atau mengubah konten halaman web [9].

Salah satu dampak paling umum dari serangan XSS adalah pencurian data sensitif [10]. Penyerang dapat menggunakan skrip berbahaya untuk mencuri cookie sesi pengguna, yang dapat berisi informasi penting seperti kredensial login atau token autentikasi [11]. Dengan akses ke cookie ini, penyerang dapat melakukan session hijacking, yaitu mengambil alih sesi pengguna dan bertindak atas nama pengguna tersebut tanpa sepengetahuan mereka [12][13][14].

Keamanan aplikasi web adalah salah satu aspek yang paling krusial dalam era digital saat ini. Dengan semakin meningkatnya penggunaan aplikasi web untuk berbagai keperluan, mulai dari transaksi finansial, komunikasi, hingga manajemen data sensitif, ancaman terhadap keamanan web menjadi semakin nyata dan berbahaya. XSS adalah salah satu ancaman paling serius yang dihadapi oleh pengembang web saat ini, dan statistik menunjukkan bahwa serangan XSS merupakan salah satu dari sepuluh ancaman keamanan web teratas menurut Open Web Application Security Project (OWASP) [14][15].

Urgensi penelitian ini terletak pada meningkatnya insiden serangan XSS dan dampaknya yang signifikan terhadap keamanan dan integritas data pengguna. Menurut studi yang dilakukan oleh Sethi serangan XSS dapat digunakan untuk mencuri data pribadi pengguna, mengakses informasi rahasia, dan bahkan mengendalikan akun pengguna tanpa sepengetahuan mereka [4]. Sementara itu, penelitian lain [3] menunjukkan bahwa masih banyak aplikasi web yang rentan terhadap serangan ini karena kurangnya penerapan teknik mitigasi yang memadai. Oleh karena itu, penelitian ini diharapkan dapat memberikan kontribusi yang signifikan dalam meningkatkan kesadaran dan pemahaman tentang pentingnya penerapan teknik keamanan yang lebih efektif, terutama dalam konteks penggunaan PHP.

Dalam konteks ini, penting untuk mengembangkan teknik dan strategi yang lebih efektif untuk mendeteksi dan mencegah serangan XSS pada aplikasi PHP. Salah satu solusi potensial adalah penggunaan alat pengujian otomatis seperti XSStrike [17], yang dirancang khusus untuk mendeteksi dan mencegah kerentanan XSS. XSStrike dapat melakukan analisis pola skrip berbahaya dan pemindaian otomatis untuk mengidentifikasi dan mengurangi potensi serangan XSS pada aplikasi web [14]. Namun, meskipun alat ini memiliki potensi besar, belum banyak penelitian yang mengevaluasi efektivitas penggunaannya dalam konteks aplikasi PHP Native (bukan framework seperti Laravel, Codeigniter dan lainnya).

XSStrike memiliki kemampuan dalam mendeteksi skrip berbahaya melalui analisis pola yang dilakukan secara otomatis. Namun, seperti alat pengujian lainnya, XSStrike memiliki keterbatasan dalam mengidentifikasi jenis serangan XSS yang lebih kompleks atau yang memanfaatkan teknik obfuscation tertentu. Penelitian ini akan mengevaluasi sejauh mana alat ini mampu mengidentifikasi dan mencegah berbagai jenis serangan XSS dalam konteks aplikasi PHP, serta mengeksplorasi strategi mitigasi serangan XSS pada aplikasi yang dibuat menggunakan PHP Native.

Penelitian ini tidak bertujuan untuk memodifikasi bahasa PHP itu sendiri, melainkan untuk mengevaluasi bagaimana aplikasi berbasis PHP yang sudah ada dapat diproteksi dari serangan XSS. Oleh karena itu, penelitian ini akan fokus pada penerapan teknik mitigasi yang kompatibel dengan PHP, serta memanfaatkan XSStrike untuk mendeteksi potensi kerentanan dari bahasa PHP Native. Dengan kata lain, penelitian ini bekerja dalam batasan yang ada pada PHP, yang berarti bahwa solusi yang diusulkan akan bersifat tambahan pada pengembangan aplikasi, bukan pada bahasa pemrograman PHP Native itu sendiri.

Penelitian ini juga akan mengkaji bagaimana penerapan praktik terbaik dalam pengembangan aplikasi PHP, seperti sanitasi input, validasi data, dan penggunaan fungsi keamanan bawaan PHP. Dengan mengkombinasikan temuan kerentanan yang dihasilkan XSStrike dan fungsi bawaan (built-in) PHP Native pendekatan ini diharapkan dapat memberikan saran bagi pengguna PHP khususnya secara native serta aplikasi PHP dapat memiliki ketahanan yang lebih tinggi terhadap serangan XSS. Praktek ini bertujuan untuk melindungi aplikasi dari serangan, sementara XSStrike akan membantu mendeteksi kerentanan.

II. METODE PENELITIAN

Penelitian ini menggunakan pendekatan eksperimen untuk menguji efektivitas alat pengujian otomatis XSStrike dalam mendeteksi serangan XSS pada aplikasi PHP Native. Metode eksperimen dipilih karena memungkinkan pengujian langsung terhadap berbagai jenis serangan XSS di lingkungan yang terkendali dan memberikan data empiris mengenai performa XSStrike. Tujuan utama dari eksperimen ini adalah untuk mengevaluasi kemampuan XSStrike dalam mengidentifikasi kerentanan XSS serta memberikan rekomendasi untuk pengembangan teknik preventif yang lebih efektif dalam konteks aplikasi PHP.

Pada tahap ini, lingkungan pengujian disiapkan untuk menguji berbagai jenis aplikasi PHP terhadap serangan XSS. Aplikasi PHP yang rentan terhadap serangan XSS akan diidentifikasi dan diatur dalam server lokal yang terisolasi. Aplikasi ini akan dipilih berdasarkan kerentanannya terhadap serangan XSS dan relevansinya dengan skenario serangan yang umum terjadi. Selain itu, XSStrike akan diinstal dan dikonfigurasi sesuai dengan dokumentasi resmi alat tersebut. Lingkungan pengujian akan mencakup berbagai komponen.

Pengujian penetrasi dilakukan untuk mengidentifikasi dan mengevaluasi kerentanan XSS pada aplikasi PHP menggunakan XSStrike. Setiap aplikasi PHP yang telah dipersiapkan akan dikenakan berbagai skenario serangan XSS, termasuk Reflected XSS, Stored XSS, dan DOM-based XSS. Proses ini mencakup penggunaan XSStrike untuk melakukan fuzzing input, mengidentifikasi titik masuk yang rentan, dan menganalisis respons dari aplikasi untuk mendeteksi adanya eksekusi skrip berbahaya. Pengujian ini akan dilakukan secara berulang untuk memastikan hasil yang konsisten dan akurat. Selama pengujian, data seperti waktu yang dibutuhkan untuk deteksi, jumlah serangan yang berhasil diidentifikasi, dan akan dicatat untuk dianalisis lebih lanjut.

Pengujian akan dilakukan dalam lingkungan lokal yang terisolasi untuk memastikan bahwa setiap aplikasi PHP diuji tanpa adanya gangguan dari faktor eksternal. Lingkungan lokal ini akan dikonfigurasi menggunakan server web lokal dan database lokal untuk mengelola aplikasi PHP yang diuji. Pengujian ini akan berfokus pada deteksi kerentanan dalam skenario yang terkendali, di mana aplikasi dapat diakses melalui jaringan internal yang terbatas. Tujuan dari pengujian di lingkungan lokal adalah untuk meminimalkan risiko keamanan selama pengujian awal, serta untuk memastikan bahwa XSStrike dapat mendeteksi berbagai jenis serangan XSS dalam kondisi yang aman dan terkontrol.

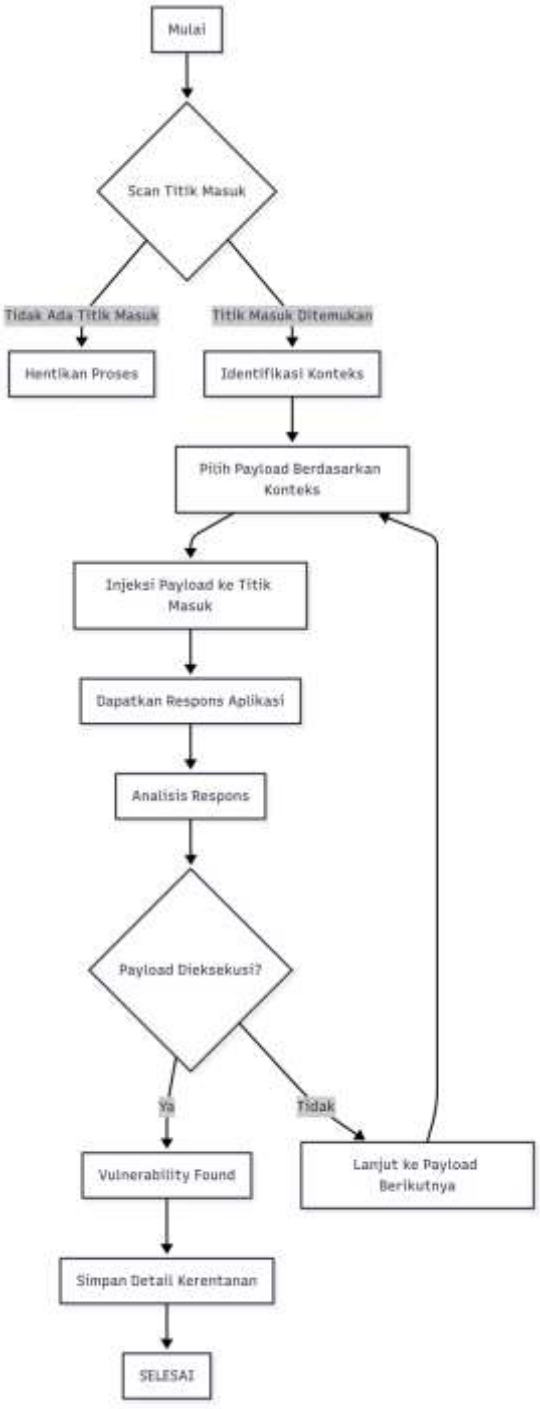
Penelitian ini akan mengembangkan rekomendasi praktis bagi pengembang web mengenai cara meningkatkan keamanan aplikasi mereka terhadap serangan XSS. Rekomendasi yang dikembangkan mencakup penggunaan teknik sanitasi input yang tepat serta penerapan kebijakan keamanan yang relevan guna mengurangi risiko eksekusi skrip berbahaya. Dalam proses pengujian dan analisis, beberapa alat dan infrastruktur pendukung digunakan. Salah satu alat utama yang dimanfaatkan adalah XSStrike, sebuah alat pengujian otomatis yang dirancang khusus untuk mendeteksi dan mencegah kerentanan XSS. XSStrike dipilih karena memiliki kemampuan mendeteksi berbagai jenis serangan, termasuk Reflected XSS, Stored XSS, dan DOM-Based XSS, serta keunggulannya dalam melakukan teknik fuzzing secara cerdas terhadap input aplikasi web [14]. Selain XSStrike, penelitian ini juga menggunakan aplikasi berbasis PHP yang secara sengaja dibuat rentan terhadap serangan XSS sebagai target pengujian. Aplikasi ini memungkinkan peneliti mensimulasikan berbagai skenario serangan untuk mengevaluasi efektivitas alat dan teknik pencegahan yang diterapkan. Lingkungan pengujian didukung oleh server web yang dikonfigurasi secara khusus untuk menjalankan aplikasi PHP tersebut. Infrastruktur ini digunakan untuk mensimulasikan kondisi operasional nyata dari sebuah aplikasi web, sehingga hasil pengujian dapat merefleksikan situasi yang lebih representatif terhadap potensi ancaman di dunia nyata.

Dalam penelitian ini, yang digambarkan pada gambar 1, XSStrike digunakan sebagai alat utama untuk mendeteksi kerentanan XSS pada aplikasi berbasis PHP. XSStrike bekerja dengan cara memindai titik-titik masuk potensial dalam aplikasi web, seperti parameter input dan form, lalu menyuntikkan berbagai jenis payload yang dirancang khusus untuk menguji kemungkinan adanya kerentanan. Alat ini mengadopsi pendekatan fuzzing, yakni teknik pengujian otomatis yang menggunakan berbagai kombinasi input dengan tujuan untuk memicu dan mengekspos area yang rentan terhadap serangan. Proses kerja XSStrike dimulai dengan melakukan pemindaian terhadap aplikasi untuk menemukan titik-titik masuk yang mungkin rentan. Apabila titik masuk ditemukan, proses dilanjutkan ke tahap identifikasi konteks, yaitu menentukan bagaimana dan di mana input tersebut akan dieksekusi, apakah dalam elemen HTML, atribut, atau konteks JavaScript. Berdasarkan hasil identifikasi ini, XSStrike akan memilih payload yang paling relevan dan sesuai dengan konteks tersebut, lalu menyuntikkannya ke dalam titik masuk yang ditentukan.

XSStrike memiliki sejumlah fitur utama yang menjadikannya alat unggulan dalam pengujian kerentanan XSS pada aplikasi web. Alat ini mampu melakukan pemindaian terhadap serangan Reflected XSS dan DOM-Based XSS secara otomatis dan menyeluruh. Selain itu, XSStrike dilengkapi dengan kemampuan crawling multi-threaded yang memungkinkan pemetaan aplikasi web secara efisien dan cepat. Salah satu keunggulan pentingnya adalah kemampuan analisis konteks, yang memungkinkan alat ini menghasilkan payload yang relevan dan tepat sasaran sesuai dengan posisi serta cara input digunakan dalam struktur halaman. XSStrike juga mendukung protokol HTTP secara penuh dan memiliki fitur evasion terhadap Web Application Firewall (WAF), yang berguna untuk menguji aplikasi di bawah kondisi perlindungan keamanan tingkat lanjut. Mesin fuzzing yang disematkan dalam XSStrike sangat canggih, mampu melakukan brute force terhadap berbagai kombinasi payload untuk menembus lapisan sanitasi input yang lemah. Di samping itu, XSStrike juga dapat mendeteksi pustaka JavaScript yang sudah usang, yang mungkin mengandung kerentanan tersembunyi. Dengan generator dan encoder payload yang cerdas, XSStrike mampu menciptakan berbagai variasi skrip uji yang kompleks untuk mengeksplorasi celah XSS dalam berbagai skenario serangan. Melalui pendekatan dan fitur-fitur tersebut, XSStrike mampu mendeteksi berbagai jenis serangan XSS dan memberikan gambaran menyeluruh tentang kondisi keamanan aplikasi terhadap ancaman ini. Mekanisme kerjanya yang disusun dalam bentuk pseudocode turut membantu menjelaskan alur logika proses pendeteksian kerentanan secara sistematis dan efisien pada aplikasi berbasis PHP.

Setelah pengujian serangan dilakukan, alat ini mengirimkan permintaan ke aplikasi dan menganalisis respons yang diterima untuk mendeteksi apakah payload yang dikirimkan berhasil dieksekusi. Jika skrip dalam payload terdeteksi berjalan di sisi klien, maka XSStrike mencatatnya sebagai kerentanan. Jika tidak, maka proses pengujian dilanjutkan dengan payload lain yang berbeda. Ketika suatu kerentanan berhasil diidentifikasi, XSStrike menyimpan informasi secara detail, termasuk titik masuk yang digunakan, jenis payload yang disuntikkan, serta bentuk respons dari aplikasi. Proses ini kemudian berulang kembali untuk mengevaluasi titik masuk lainnya

hingga seluruh aplikasi selesai dipindai secara menyeluruh. Dengan alur kerja ini, XSStrike mampu memberikan pemetaan menyeluruh terhadap potensi kerentanan XSS dalam aplikasi web berbasis PHP Native, sekaligus menunjukkan efektivitas alat ini sebagai bagian dari proses pengujian keamanan aplikasi.



Gambar 1. XSStrike Workflow

III. HASIL PENELITIAN

Pengujian kerentanan terhadap serangan XSS pada aplikasi web berbasis PHP Native merupakan langkah penting dalam menjamin keamanan sistem. Dalam penelitian ini, digunakan alat otomatis XSStrike untuk mengidentifikasi dan mengevaluasi kerentanan XSS, dengan fokus utama pada efektivitas alat dalam mendeteksi berbagai jenis serangan.

1. Konfigurasi Lingkungan Uji

Lingkungan pengujian dirancang agar menyerupai kondisi produksi namun tetap aman secara lokal. PHP versi 8.2.4 dipilih sebagai interpreter karena stabilitas dan dukungannya terhadap fitur pemrograman modern. Sistem operasi yang digunakan adalah macOS Darwin 21.6.0 dengan arsitektur x86_64, yang mendukung kestabilan dan performa tinggi. Untuk layanan web, digunakan Apache 2.0 Handler yang fleksibel dan mendukung modul-modul penting dalam pengujian keamanan.

Alat pengujian utama adalah XSStrike versi 3.1.5 [17], yang dikenal mampu menghasilkan berbagai payload kompleks untuk menguji Reflected, Stored, dan DOM-Based XSS. Kombinasi perangkat lunak ini menciptakan lingkungan pengujian yang ideal untuk mensimulasikan dan menganalisis skenario serangan dengan akurasi tinggi dan dapat direplikasi.

2. Pengujian dan Target Aplikasi

Objek uji yang digunakan adalah OSTE Vulnerable Web Application, aplikasi berbasis PHP yang secara sengaja dibuat memiliki berbagai celah keamanan untuk keperluan pembelajaran dan pengujian. OSTE menawarkan berbagai skenario kerentanan, termasuk SQL Injection, XSS, dan OS Command Injection, dengan struktur database sederhana yang memudahkan eksplorasi dan eksperimen.

Penggunaan OSTE memungkinkan evaluasi menyeluruh terhadap performa XSStrike dalam mendeteksi XSS. Fitur-fitur seperti generator payload cerdas dan kemampuan crawling multi-threaded dari XSStrike dimanfaatkan untuk menguji sejauh mana alat ini efektif dalam menemukan titik-titik kerentanan pada aplikasi yang rentan, sekaligus menguji ketahanan aplikasi setelah penerapan teknik mitigasi.

3. XSStrike dalam Identifikasi Kerentanan XSS

Pengujian keamanan aplikasi web berperan penting dalam mengidentifikasi potensi celah yang rentan terhadap serangan. Dalam penelitian ini, fokus utama diarahkan pada evaluasi efisiensi alat deteksi otomatis seperti XSStrike dalam mengidentifikasi serangan XSS. Efisiensi dinilai berdasarkan kemampuan alat dalam mendeteksi titik kerentanan, menghasilkan payload yang efektif, dan memberikan tingkat kepercayaan (confidence) terhadap keberhasilan eksploitasi. Dengan pendekatan ini, penelitian bertujuan memahami sejauh mana XSStrike dapat mendukung upaya mitigasi risiko keamanan dalam pengembangan aplikasi web berbasis PHP.

3.1 Reflected XSS

Salah satu jenis serangan yang diuji adalah Reflected XSS, yakni serangan di mana input berbahaya yang tidak disanitasi dikembalikan langsung oleh server dalam respons halaman web. Serangan ini umumnya terjadi ketika penyerang menyisipkan skrip berbahaya ke dalam parameter URL, yang kemudian direfleksikan dalam halaman web tanpa validasi atau encoding yang memadai.

Ketika korban mengakses URL tersebut, skrip langsung dijalankan di sisi browser, memungkinkan penyerang mencuri informasi seperti cookie atau data sesi [21]. Alur serangan ini ditunjukkan secara visual pada Gambar 2, yang menggambarkan tahapan eksploitasi Reflected XSS mulai dari injeksi hingga pencurian informasi oleh penyerang.

Gambar 2. Reflected XSS flow [22]

Pengujian Reflected XSS dalam penelitian ini dilakukan pada endpoint http://localhost/oste/XSS/page1.php, menggunakan parameter username sebagai media injeksi. Alat XSStrike secara otomatis menghasilkan dan menguji berbagai payload terhadap parameter tersebut. Salah satu contoh payload yang digunakan adalah ><d3v%09ONPOinTEreNTeR%09=%09confirm()>v3dm0s, yang terbukti mampu mengeksekusi skrip di sisi klien. Dalam pengujian ini, XSStrike menghasilkan total 3.096 payload, dan salah satunya memiliki tingkat efisiensi 100% dengan confidence level 9, menandakan deteksi kerentanan yang sangat tinggi.

Gambar 3 menampilkan output pengujian Reflected XSS menggunakan alat XSStrike versi 3.1.5 pada parameter username dari endpoint http://localhost/oste/XSS/page1.php.

Dalam tangkapan layar ini, XSStrike menghasilkan total 3.096 payload untuk menguji kerentanan Reflected XSS, dan melakukan eksplorasi secara sistematis terhadap kemungkinan eksekusi skrip berbahaya di sisi klien. Beberapa contoh payload yang ditampilkan dalam gambar, seperti:

- ><d3v%09ONPOinTEreNTeR%09=%09confirm()>v3dm0s
- ><dETaILs/+oNtOgGLE%0d=%0d(prompt)//
- ><HTMl%0dONPOinTErenter%09=%09(prompt)^%0dx//

menunjukkan kombinasi karakter Unicode, obfuscation menggunakan whitespace (seperti %09, %0d, %0a), dan pemanggilan fungsi JavaScript seperti prompt() atau confirm(). Ini adalah teknik untuk menghindari filter dasar dan menguji sanitasi input dari aplikasi. Masing-masing payload dihasilkan dan dievaluasi oleh XSStrike berdasarkan dua metrik utama:

Efficiency: Seberapa efektif payload dalam memicu respons yang diharapkan (di sini selalu 100%, artinya payload tersebut berhasil memicu respons sistem).

Confidence: Level keyakinan XSStrike terhadap keberhasilan eksploitasi XSS (ditunjukkan dengan nilai 9 dari skala 10, artinya sangat tinggi).

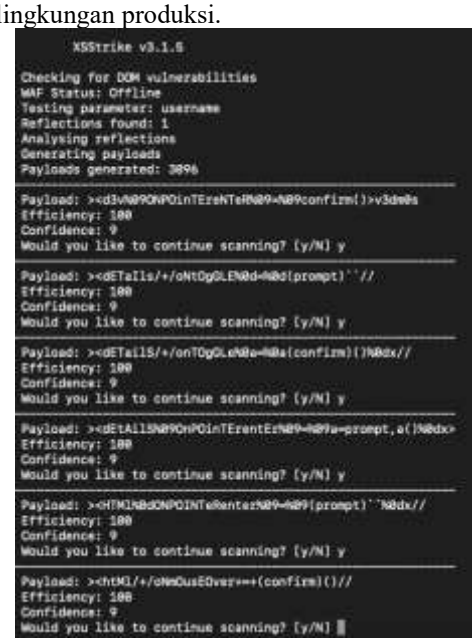
Setelah setiap payload diuji, XSStrike memberikan opsi interaktif:

"Would you like to continue scanning? [y/N]"

yang menunjukkan bahwa alat ini menjalankan proses semi-otomatis, memungkinkan peneliti untuk mengevaluasi setiap respons aplikasi sebelum lanjut ke payload berikutnya. Ini memberikan fleksibilitas tinggi untuk eksplorasi mendalam terhadap lapisan keamanan aplikasi.

Secara keseluruhan, Gambar 3 menunjukkan bahwa aplikasi yang diuji benar-benar rentan terhadap Reflected XSS, karena beberapa payload dapat dieksekusi dengan confidence dan efisiensi penuh. Ini menegaskan perlunya penerapan teknik preventif seperti validasi input, encoding output, dan pembatasan sumber daya melalui CSP.

Alat ini juga menunjukkan fleksibilitas dengan meminta konfirmasi pengguna sebelum melanjutkan ke pengujian payload berikutnya, yang memungkinkan eksplorasi lebih dalam terhadap lapisan pertahanan aplikasi. Temuan ini menunjukkan bahwa XSStrike memiliki kemampuan tinggi dalam mendeteksi celah Reflected XSS secara cepat dan akurat. Dengan menghasilkan payload yang relevan dan tingkat kepercayaan tinggi terhadap potensi eksploitasi, XSStrike terbukti efektif sebagai alat bantu dalam proses evaluasi keamanan aplikasi web. Oleh karena itu, penggunaan alat otomatis seperti XSStrike menjadi langkah penting dalam proses pengujian sebelum aplikasi diimplementasikan pada lingkungan produksi.

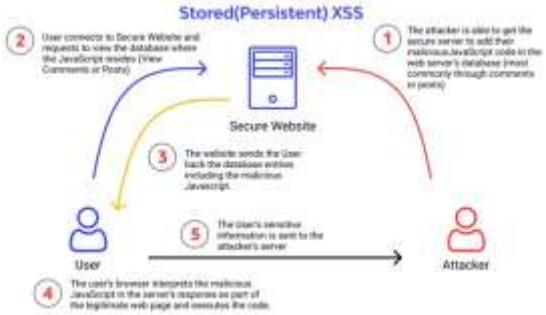


Gambar 3. Hasil pengujian Reflected XSS pada OSTE

3.2 Reflected XSS

Jenis serangan Stored XSS merupakan bentuk serangan yang lebih berbahaya dibandingkan dengan Reflected XSS karena input berbahaya dari penyerang tidak hanya diproses secara langsung, tetapi juga disimpan di sisi server dan ditampilkan kembali saat halaman diakses oleh pengguna lain. Alur serangan ini dijelaskan secara visual pada Gambar 4, yang

menunjukkan bagaimana penyerang dapat menyisipkan skrip JavaScript berbahaya melalui form input seperti komentar, yang kemudian disimpan dalam basis data server web. Ketika halaman yang berisi data tersebut diakses oleh pengguna lain, skrip berbahaya akan dikirim kembali dan dieksekusi oleh browser tanpa disadari. Hal ini memungkinkan penyerang mencuri data sensitif pengguna, seperti cookie, sesi login, atau informasi pribadi lainnya.



Gambar 4. Store XSS flow [22]

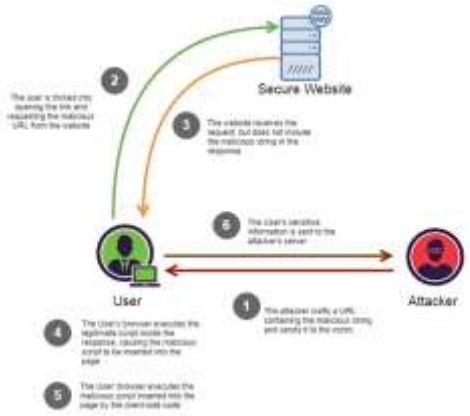
Pengujian Stored XSS dalam penelitian ini menggunakan payload yang sebelumnya berhasil digunakan pada skenario Reflected XSS. Payload berupa ><d3v%09ONPOinTEreNTeR%09=%09confirm()>v3dm0s disisipkan melalui parameter username pada halaman http://localhost/oste/XSS/page1.php. Setelah input dikirim melalui form dan halaman dimuat ulang, hasil pengujian menunjukkan bahwa data yang dimasukkan ditampilkan kembali di halaman tanpa melalui proses validasi atau encoding HTML yang memadai. Hal ini tampak jelas pada Gambar 5, di mana karakter berbahaya seperti < dan > tetap tampil dalam bentuk aslinya dan bukan sebagai entitas HTML yang aman, sehingga memungkinkan skrip untuk dieksekusi secara langsung oleh browser pengguna.

Gambar 5. Hasil pengujian Store XSS pada OSTE

Temuan ini mengindikasikan bahwa parameter username menyimpan input pengguna secara langsung dan menampilkannya kembali tanpa proses pengamanan, yang menjadikannya titik masuk yang sangat rentan terhadap serangan. Kondisi ini membuka peluang bagi penyerang untuk menyisipkan skrip berbahaya yang akan aktif setiap kali halaman diakses oleh pengguna lain, menjadikan serangan bersifat persisten. Potensi ancaman dari Stored XSS tidak hanya terbatas pada pencurian data, tetapi juga mencakup serangan phishing melalui tampilan halaman palsu, manipulasi antarmuka pengguna, serta kerusakan reputasi aplikasi akibat tampilan atau perilaku yang tidak diinginkan. Oleh karena itu, hasil pengujian ini menegaskan pentingnya penerapan validasi input dan sanitasi data secara ketat sebagai bentuk pertahanan utama terhadap serangan Stored XSS dalam pengembangan aplikasi web.

3.2 DOM-Base XSS

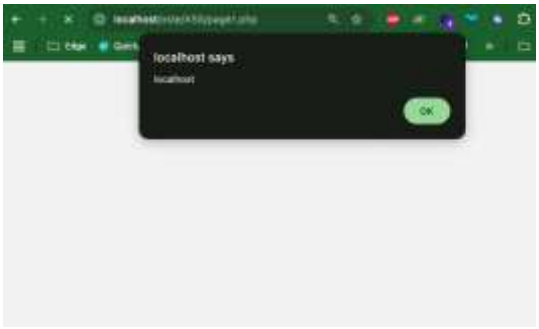
Serangan DOM-Based XSS merupakan jenis yang sepenuhnya terjadi di sisi klien, di mana skrip berbahaya dieksekusi melalui manipulasi langsung terhadap Document Object Model (DOM) menggunakan JavaScript di browser pengguna. Berbeda dengan Reflected maupun Stored XSS, jenis serangan ini tidak melibatkan pengiriman atau respons dari server. Alur serangan DOM-Based XSS ditunjukkan pada Gambar 6, yang menggambarkan bagaimana penyerang menyusun URL berbahaya yang mengandung skrip atau string manipulatif, lalu mengirimkannya kepada korban melalui media seperti email phishing atau pesan langsung. Setelah korban mengakses URL tersebut, JavaScript di dalam halaman web memproses input dari URL tanpa validasi, lalu menyisipkan skrip berbahaya langsung ke dalam struktur DOM. Proses ini biasanya dilakukan melalui fungsi seperti document.write(), innerHTML, atau metode serupa, tanpa melalui penyaringan atau encoding yang memadai. Skrip yang telah disisipkan kemudian dijalankan oleh browser, dan memungkinkan penyerang mencuri informasi sensitif seperti cookie, token sesi, atau data pribadi lainnya, yang kemudian dikirimkan ke server yang mereka kontrol.



Gambar 6. DOM-Base XSS flow [22]

Dalam pengujian yang dilakukan, payload yang digunakan adalah document.write('<script>alert(document.domain)</script>');. Payload ini secara langsung disisipkan dalam halaman uji page1.php, dan hasilnya menunjukkan bahwa skrip berhasil dieksekusi oleh browser, ditandai dengan munculnya popup berisi teks "localhost". Bukti keberhasilan eksekusi ini ditampilkan pada Gambar 7, yang menunjukkan bahwa aplikasi rentan terhadap DOM-Based XSS karena tidak adanya validasi atau encoding terhadap input pengguna sebelum dimasukkan kembali ke dalam halaman melalui DOM. Kerentanan ini membuka

peluang serius bagi penyerang, karena mereka dapat menjalankan skrip berbahaya di browser pengguna, baik untuk mencuri data, menampilkan halaman phishing, maupun memanipulasi tampilan dan konten halaman secara dinamis.



Gambar 7. Hasil pengujian DOM-Base XSS pada OSTE

Hasil ini menegaskan pentingnya penerapan praktik preventif dalam pengembangan aplikasi PHP Native. Karena input yang tidak aman dapat dimanipulasi dengan mudah di sisi klien, langkah seperti validasi dan sanitasi input harus diterapkan secara ketat sejak tahap awal pengembangan. Praktik ini memastikan bahwa semua data yang diterima dari pengguna diperiksa dan dibersihkan secara menyeluruh sebelum digunakan dalam DOM atau ditampilkan kembali di halaman web. Dengan pendekatan ini, aplikasi akan memiliki ketahanan yang lebih baik terhadap ancaman DOM-Based XSS yang sifatnya sangat tersembunyi namun berbahaya.

4. Penerapan Praktik Preventive XSS Bagi Pengembangan PHP Native

Dalam pengembangan aplikasi web berbasis PHP Native, serangan XSS menjadi salah satu ancaman utama yang dapat mengganggu integritas sistem dan membahayakan data pengguna. XSS memungkinkan penyerang menyisipkan skrip berbahaya yang dieksekusi di sisi klien melalui input yang tidak aman. Oleh karena itu, penerapan praktik preventif yang menyeluruh sangat penting dan harus dimulai sejak tahap awal pengembangan. Salah satu komponen fundamental dalam upaya ini adalah validasi dan sanitasi input, yang berfungsi untuk memastikan bahwa data yang diterima dari pengguna tidak mengandung elemen berbahaya sebelum digunakan atau ditampilkan kembali dalam halaman web.

4.1 DOM-Base XSS

Validasi input difokuskan pada pemeriksaan terhadap nilai yang dimasukkan oleh pengguna agar sesuai dengan format atau aturan yang telah ditentukan, seperti tipe data, panjang karakter, atau pola khusus. Sementara itu, sanitasi input bertujuan untuk membersihkan karakter-karakter yang berpotensi digunakan dalam injeksi skrip, sehingga mencegah kemungkinan skrip berbahaya dieksekusi di browser pengguna. Dengan kombinasi dua pendekatan ini, potensi eksploitasi melalui input yang tidak valid dapat diminimalkan secara signifikan.

Skrip 1. Penerapan Validasi Input

BEFORE

```
<?php
  if(isset($\_POST\['username'\])) {
    echo "<h1>Welcome To OSTE Vulnerable web application <3";
    echo $\_POST\['username'\];
    echo "</h1>";
  }
?>
```

#AFTER

```
<?php
if(isset($\_POST\['username'\])&&!empty($\_POST\['username'\])) {
    echo "<h1>Welcome To OSTE Vulnerable web application <3";
    echo $\_POST\['username'\];

    echo "</h1>";
  } else {
    echo "<h1>Username tidak boleh kosong!</h1>";
  }
?>
```

Penerapan teknik ini ditunjukkan pada Skrip 1, di mana kondisi sebelum penerapan validasi menunjukkan bahwa aplikasi secara langsung menampilkan nilai dari parameter username tanpa pengecekan apapun. Kondisi ini memungkinkan injeksi skrip berbahaya secara langsung ke dalam halaman. Setelah dilakukan modifikasi, skrip diperbarui dengan logika pemeriksaan menggunakan fungsi isset() dan !empty() untuk memastikan bahwa nilai parameter username benar-benar ada dan tidak kosong sebelum ditampilkan. Jika input tidak memenuhi kondisi tersebut, sistem akan menampilkan pesan kesalahan yang sesuai. Dengan demikian, implementasi ini tidak hanya meningkatkan keamanan, tetapi juga memberikan umpan balik yang lebih jelas kepada pengguna terkait kesalahan input.

4.2 Encoding

Encoding output merupakan salah satu langkah krusial dalam mencegah serangan XSS pada aplikasi web. Proses ini bertujuan untuk memastikan bahwa data yang dikirimkan dari server ke browser dikonversi ke dalam format aman sebelum ditampilkan, sehingga karakter spesial seperti <, >, ', " dan & tidak diinterpretasikan sebagai bagian dari kode HTML atau JavaScript yang berbahaya. Tanpa mekanisme encoding, input pengguna yang mengandung karakter tersebut berpotensi langsung dieksekusi oleh browser sebagai skrip aktif, yang dapat digunakan penyerang untuk mencuri data atau melakukan manipulasi terhadap halaman web.

Dalam pengembangan berbasis PHP, encoding dapat diterapkan menggunakan fungsi bawaan seperti htmlspecialchars() dan htmlentities(). Fungsi htmlspecialchars() digunakan untuk mengkonversi karakter khusus menjadi entitas HTML yang aman, sementara htmlentities() dapat mengubah seluruh karakter yang memiliki representasi HTML. Penggunaan flag ENT_QUOTES sangat dianjurkan agar kutip tunggal (') dan kutip ganda (") juga diubah menjadi entitas aman (' dan "). Selain itu, penggunaan encoding karakter UTF-8 membantu mencegah eksploitasi berbasis multi-byte encoding yang dapat digunakan dalam serangan lanjutan.

Skrip 2. Penerapan Encoding

```
# BEFORE
<?php
if(isset($_POST['username'])&&!empty($_POST['username'])) {
    echo "<h1>Welcome To OSTE Vulnerable web application <3";
    echo $_POST['username'];
    echo "</h1>";
  } else {
    echo "<h1>Username tidak boleh kosong!</h1>";
  }
?>

#AFTER
<?php
if(isset($_POST['username'])&&!empty($_POST['username'])) {
    $username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
    echo "<h1>Welcome To OSTE Vulnerable web application <3, " . $username . "</h1>";
  } else {
    echo "<h1>Username tidak boleh kosong!</h1>";
  }
?>
```

Penerapan encoding ditunjukkan pada Skrip 2, di mana perbandingan antara kode sebelum dan sesudah implementasi encoding menggambarkan perubahan signifikan dalam penanganan input. Sebelumnya, aplikasi menampilkan langsung nilai dari parameter username tanpa proses penyaringan. Setelah diperbaiki, skrip menggunakan htmlspecialchars() dengan parameter ENT_QUOTES dan 'UTF-8' untuk memastikan setiap karakter spesial diubah menjadi representasi aman. Misalnya, karakter < dikonversi menjadi <, > menjadi >, ' menjadi ', dan " menjadi ". Hasilnya, browser akan menampilkan input tersebut sebagai teks biasa, bukan sebagai skrip aktif yang dapat dijalankan.

Ketika seorang pengguna memasukkan input berbahaya seperti <script>alert('XSS')</script>. Tanpa encoding, browser akan menafsirkan input tersebut sebagai perintah JavaScript dan menampilkan popup bertuliskan “XSS”. Namun, setelah diterapkan encoding output, browser hanya akan menampilkan teks literal seperti <script>alert('XSS')</script>, sehingga tidak menimbulkan eksekusi skrip.

Encoding output merupakan bentuk pertahanan lapis terakhir yang sangat penting terutama dalam konteks tampilan konten dinamis dari pengguna. Meskipun validasi dan sanitasi input telah diterapkan, encoding tetap harus dilakukan pada setiap data yang ditampilkan kembali di halaman untuk mencegah potensi eksekusi yang lolos dari tahap awal. Dengan penggunaan htmlspecialchars() atau htmlentities(), aplikasi web dapat secara signifikan mengurangi risiko serangan XSS dan meningkatkan keandalan serta keamanan interaksi antara pengguna dan sistem.

4.3 Penerapan Content Security Policy (CSP)

CSP merupakan mekanisme keamanan berbasis header HTTP yang dirancang untuk melindungi aplikasi web dari berbagai ancaman, khususnya serangan XSS dan injeksi konten berbahaya lainnya [18][19][20]. Dengan CSP, pengembang dapat mengontrol jenis dan sumber konten yang diizinkan untuk dimuat dan dieksekusi oleh browser. Mekanisme ini bekerja dengan mendefinisikan kebijakan eksplisit yang membatasi pemuatan sumber daya seperti skrip, gaya (CSS), gambar, dan elemen tertanam (iframe), hanya dari sumber yang dianggap sah. Sebagai contoh, CSP memungkinkan pengembang untuk mengizinkan hanya skrip dari domain lokal ('self'), memblokir eksekusi JavaScript inline, serta mencegah penggunaan fungsi berbahaya seperti eval() [23].

CSP menjadi pelengkap penting dalam pencegahan XSS karena tetap dapat membatasi eksekusi skrip meskipun input berbahaya berhasil disisipkan ke halaman. Implementasi CSP dapat dilakukan dengan menambahkan header Content-Security-Policy pada response HTTP. Header ini mendefinisikan berbagai directive, seperti default-src untuk sumber default semua jenis konten, script-src untuk skrip, style-src untuk CSS, img-src untuk gambar, dan frame-src untuk kontrol konten iframe.

Skrip 3. Penerapan CSP

```
# BEFORE
<?php
if(isset($_POST['username'])&&!empty($_POST['username'])) {
    $username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
    echo "<h1>Welcome To OSTE Vulnerable web application <3, " . $username . "</h1>";
  } else {
    echo "<h1>Username tidak boleh kosong!</h1>";
  }
?>

#AFTER
<?php
  header("Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self'; style-src 'self'; frame-src 'none';");


if(isset($_POST['username'])&&!empty($_POST['username'])) {
```

```
    $username = htmlspecialchars($_POST['username'], ENT_QUOTES, 'UTF-8');
    echo "<h1>Welcome To OSTE Vulnerable web application <3, " . $username . "</h1>";
  } else {
    echo "<h1>Username tidak boleh kosong!</h1>";
  }
?>
```

Penerapan CSP ditunjukkan dalam Skrip 3. Sebelum CSP diterapkan, aplikasi menampilkan input pengguna secara langsung, meskipun telah melalui proses encoding menggunakan htmlspecialchars(). Setelah modifikasi, ditambahkan baris header("Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self'; style-src 'self'; frame-src 'none';"); di awal skrip PHP. Kebijakan ini menetapkan bahwa semua konten hanya boleh dimuat dari domain yang sama ('self'), sehingga konten dari domain asing akan diblokir.

Secara spesifik, direktif script-src 'self' hanya mengizinkan pemuatan skrip dari domain lokal dan secara eksplisit memblokir semua skrip inline atau yang berasal dari sumber eksternal. Hal ini sangat efektif untuk mencegah eksekusi skrip injeksi seperti <script>alert('XSS')</script>. Direktif img-src 'self' dan style-src 'self' berfungsi untuk membatasi pemuatan gambar dan stylesheet hanya dari sumber lokal, sehingga risiko penyisipan konten berbahaya seperti malware tersembunyi atau stylesheet berbahaya dapat dicegah. Sementara itu, frame-src 'none' sepenuhnya menonaktifkan penggunaan iframe, menutup celah terhadap serangan berbasis clickjacking atau konten tertanam lainnya.

Dalam pengujian, ketika penyerang mencoba memasukkan skrip seperti <script>alert('XSS')</script>, hasilnya menunjukkan bahwa meskipun input tersebut ditampilkan kembali (karena telah di-sanitize dengan htmlspecialchars()), browser tetap memblokir eksekusi skrip tersebut berdasarkan kebijakan CSP. Konsol browser akan menampilkan pesan seperti "Refused to execute inline script because it violates the Content Security Policy directive 'script-src 'self'." Pesan ini mengonfirmasi bahwa CSP berhasil mencegah eksekusi skrip yang tidak diotorisasi.

Dengan demikian, penerapan CSP memberikan lapisan perlindungan tambahan yang sangat efektif terhadap XSS, di samping validasi input dan encoding output. Ketika dikombinasikan, ketiga pendekatan ini membentuk sistem keamanan berlapis yang memperkuat pertahanan aplikasi web terhadap eksploitasi berbasis skrip.

4.4 Menghindari Penggunaan Fungsi Berbahaya

Dalam pengembangan aplikasi web berbasis PHP, pengembang perlu mewaspadai sejumlah fungsi bawaan yang dapat menimbulkan risiko keamanan serius. Fungsi-fungsi seperti eval(), exec(), system() dalam PHP, serta penggunaan innerHTML atau document.write() dalam JavaScript sering kali menjadi celah eksploitasi yang digunakan penyerang untuk menyisipkan skrip berbahaya. Fungsi-fungsi tersebut bekerja dengan mengeksekusi input sebagai kode program secara langsung tanpa validasi yang memadai, sehingga membuka peluang terjadinya serangan seperti XSS dan Remote Code Execution (RCE).

Bahaya dari fungsi-fungsi ini terletak pada kemampuannya dalam mengeksekusi kode yang dikendalikan oleh pengguna, terutama ketika input berasal dari sumber yang tidak terpercaya. Dalam konteks keamanan, hal ini merupakan praktik yang sangat berisiko karena aplikasi tidak dapat mengontrol perintah apa yang akan dijalankan, sehingga menimbulkan potensi manipulasi sistem atau pengambilan informasi sensitif. Oleh karena itu, pengembang disarankan untuk sepenuhnya menghindari penggunaan fungsi-fungsi berbahaya tersebut, dan menggantinya dengan pendekatan yang lebih aman dan terkendali.

Beberapa langkah preventif yang direkomendasikan antara lain adalah dengan tidak menggunakan eval(), exec(), system() dan fungsi serupa dalam pemrosesan input pengguna. Sebagai gantinya, pengembang dapat menggunakan Prepared Statements saat berinteraksi dengan basis data untuk menghindari SQL Injection, serta menggunakan htmlspecialchars() untuk menampilkan output dengan aman di halaman web. Dalam JavaScript, penghindaran terhadap manipulasi DOM menggunakan innerHTML atau document.write() juga penting, karena metode ini memungkinkan injeksi skrip. Sebagai alternatif, digunakan properti seperti textContent atau innerText yang menampilkan data sebagai teks biasa tanpa interpretasi sebagai kode HTML.

Skrip 4. Penggunaan Skrip Berbahaya

```
<?php
  if (isset($_GET['code'])) {
    $code = $_GET['code'];
    eval($code);
  }
    ?>
```

Contoh penggunaan fungsi berbahaya ditunjukkan pada Skrip 4, di mana fungsi eval() digunakan untuk mengeksekusi isi parameter code yang dikirim melalui metode GET. Ketika pengguna mengakses URL dengan parameter seperti ?code=phpinfo();, maka kode tersebut akan langsung dijalankan oleh server, sehingga dapat menampilkan informasi sistem yang sensitif. Dalam skenario yang lebih ekstrem, penyerang dapat menyisipkan kode destruktif seperti perintah penghapusan file atau eksekusi perintah sistem yang dapat mengancam integritas server.

Sebaliknya, pada skrip yang aman, input dari pengguna terlebih dahulu diproses menggunakan htmlspecialchars() yang mengonversi karakter spesial seperti <, >, ', dan " menjadi entitas HTML. Input kemudian hanya ditampilkan sebagai teks, bukan dieksekusi sebagai kode. Dengan pendekatan ini, skrip berbahaya yang dikirimkan oleh pengguna tidak akan dijalankan oleh aplikasi, sehingga potensi eksploitasi dapat dicegah secara efektif.

Penerapan prinsip ini penting untuk memastikan bahwa aplikasi tidak hanya berfungsi dengan baik, tetapi juga tahan terhadap upaya eksploitasi yang memanfaatkan kelemahan fungsi-fungsi eksekusi dinamis. Dengan menerapkan teknik pemrosesan input yang aman dan menghindari penggunaan fungsi berbahaya, pengembang dapat meningkatkan ketahanan

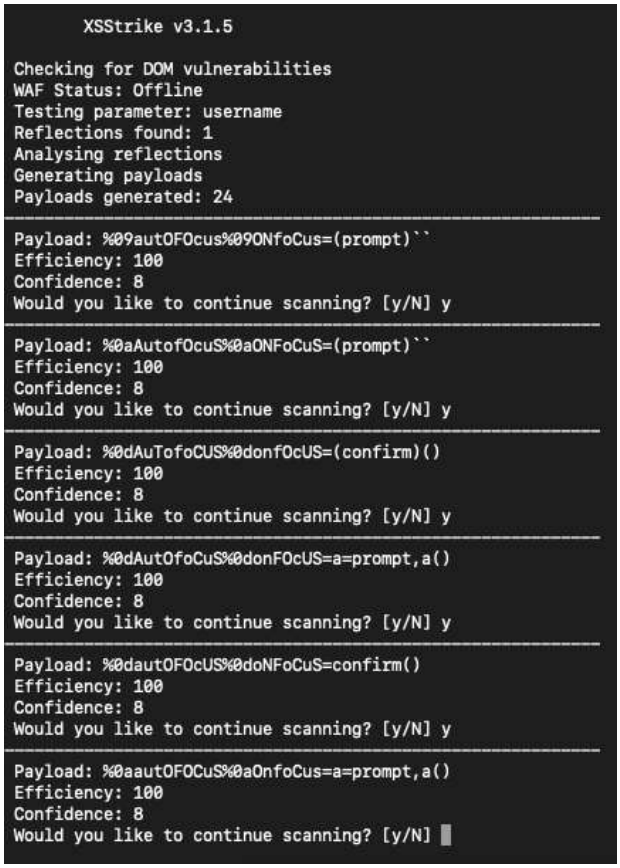
aplikasi terhadap berbagai ancaman keamanan yang umum terjadi dalam pengembangan web modern.

IV. PEMBAHASAN

Setelah penerapan berbagai mekanisme pengamanan terhadap serangan XSS, dilakukan pengujian ulang untuk mengevaluasi efektivitas perlindungan yang telah diimplementasikan. Pengujian ini dilakukan menggunakan XSStrike versi 3.1.5, dengan menjalankan skrip sebagai berikut:

```
python xsstrike.py -u
"http://localhost/oste/XSS/page1.php" --data
"username=<script>alert('Reflected XSS')</script>"
```

Tujuan dari pengujian ini adalah untuk menilai sejauh mana mekanisme seperti encoding output (htmlspecialchars()), validasi input, dan penerapan CSP mampu mencegah eksploitasi pada parameter input terhadap serangan Reflected XSS, Stored XSS, dan DOM-Based XSS.



Gambar 8. Hasil pengujian Reflected XSS pada OSTE dengan PHP yang diamankan

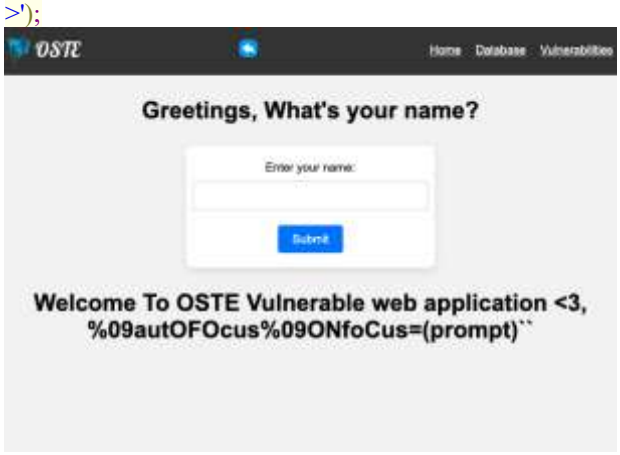
Hasil awal yang ditunjukkan pada Gambar 8 memperlihatkan bahwa XSStrike mampu mendeteksi adanya refleksi input pada parameter username, dengan total 24 payload yang dihasilkan. Payload ini memiliki tingkat efisiensi sebesar 100% dan tingkat kepercayaan (confidence level) sebesar 9, yang menunjukkan adanya potensi refleksi meskipun sudah terdapat skrip pengamanan. Payload yang dihasilkan termasuk variasi encoding karakter seperti:

- %09autOFOcus%09ONfoCus=(prompt)
- %0aAutofOcuS%0aONFoCuS=(prompt)
- %0dAuTofoCUS%0donfOcUS=(confirm)()

Payload semacam ini dirancang untuk mengecoh mekanisme deteksi sederhana dan tetap mencoba memicu eksekusi JavaScript. Namun, setelah penerapan pengamanan, skrip injeksi tidak lagi dieksekusi oleh browser, sebagaimana ditunjukkan pada Gambar 9, di mana input yang sebelumnya berbahaya kini ditampilkan sebagai plain text. Perubahan ini merupakan hasil dari penggunaan fungsi htmlspecialchars() yang mengonversi karakter berisiko seperti <, >, dan " menjadi entitas HTML yang aman, seperti <, >, dan ".

Selanjutnya, pengujian juga dilakukan terhadap potensi serangan DOM-Based XSS menggunakan skrip:

```
document.write('<script>alert(document.domain)</script>');
```



Gambar 9. Skrip injeksi berhasil diubah menjadi plain text setelah implementasi pengamanan

Gambar 10. Hasil Pengujian Injeksi Skrip dengan document.write()

Pengujian ini bertujuan untuk memastikan bahwa DOM aplikasi tidak lagi rentan terhadap manipulasi berbasis skrip klien. Hasilnya ditampilkan pada Gambar 10, yang menunjukkan bahwa input tidak dieksekusi oleh browser. Sebaliknya, isi skrip ditampilkan apa adanya sebagai teks biasa di dalam halaman. Ini menandakan bahwa selain proses encoding output berhasil mencegah interpretasi skrip, kebijakan CSP yang diterapkan dengan direktif script-src 'self' turut memperkuat keamanan dengan menolak eksekusi skrip inline yang tidak sah. Browser secara aktif memblokir skrip tersebut dan menampilkan pesan error di konsol, mengonfirmasi bahwa kebijakan telah diterapkan dengan benar.

Secara keseluruhan, pengujian pasca-implementasi ini menunjukkan bahwa kombinasi mekanisme validasi input, encoding output, serta kebijakan CSP memberikan perlindungan berlapis yang efektif terhadap berbagai jenis serangan XSS. Dengan tidak adanya eksekusi skrip meskipun input masih direfleksikan, sistem kini jauh lebih aman dan resisten terhadap eksploitasi umum yang sering ditemukan dalam aplikasi web berbasis PHP.

V. KESIMPULAN

Penelitian ini berhasil mengevaluasi efektivitas berbagai teknik preventif dalam mencegah serangan XSS pada aplikasi berbasis PHP Native, dengan menggunakan alat uji otomatis XSStrike sebagai media pengujian utama. Pengujian dilakukan secara menyeluruh terhadap tiga jenis serangan XSS-Reflected, Stored, dan DOM-Based baik sebelum maupun sesudah penerapan mekanisme pengamanan.

Pada tahap awal, XSStrike menunjukkan bahwa aplikasi memiliki kerentanan serius terhadap Reflected XSS, dengan tingkat confidence sebesar 9 dan keberhasilan eksekusi skrip berbahaya seperti <script>alert('XSS')</script> di sisi klien. Namun, XSStrike belum mampu secara otomatis mendeteksi Stored dan DOM-Based XSS, sehingga pengujian dua jenis serangan tersebut dilakukan secara manual. Pengujian lanjutan membuktikan bahwa aplikasi juga rentan terhadap Stored XSS melalui input form dan DOM-Based XSS melalui manipulasi document.write() di browser.

Setelah penerapan teknik pengamanan seperti validasi dan sanitasi input, encoding output dengan htmlspecialchars(), serta kebijakan CSP, pengujian ulang menunjukkan peningkatan ketahanan aplikasi. Tingkat confidence XSStrike menurun dari 9 menjadi 8, dan skrip injeksi tidak lagi dieksekusi oleh browser melainkan ditampilkan sebagai teks biasa. Dengan demikian, langkah-langkah pengamanan yang diterapkan terbukti efektif dalam menutup celah eksekusi XSS dan meningkatkan keamanan aplikasi terhadap berbagai jenis serangan, meskipun deteksi Stored dan DOM-Based XSS tetap membutuhkan pendekatan manual.

Secara keseluruhan, penelitian ini menyimpulkan bahwa kombinasi dari praktik terbaik keamanan mulai dari validasi input, encoding output, kebijakan CSP, hingga penghindaran fungsi berbahaya seperti eval() dan innerHTML dapat secara signifikan mengurangi risiko serangan XSS. Pengujian lanjutan juga berhasil menunjukkan bahwa payload seperti document.write('<script>alert(document.domain)</script>') tidak lagi dieksekusi di browser, melainkan ditampilkan dalam bentuk aman.

Sebagai hasil akhir, penelitian ini tidak hanya berhasil menjawab rumusan masalah terkait deteksi kerentanan XSS dengan XSStrike, tetapi juga merumuskan panduan praktis untuk penerapan teknik preventif dalam pengembangan aplikasi web PHP yang lebih aman. Berdasarkan temuan ini, disarankan agar pengembang aplikasi PHP, khususnya yang berbasis native, secara konsisten menerapkan validasi input, encoding output, dan kebijakan Content Security Policy (CSP) sebagai mekanisme pertahanan berlapis terhadap XSS. Selain itu, penggunaan alat uji otomatis seperti XSStrike dapat dimanfaatkan secara berkala dalam proses pengujian keamanan sebelum aplikasi di-deploy ke lingkungan produksi. Untuk penelitian selanjutnya, disarankan untuk mengeksplorasi integrasi antara alat deteksi otomatis dan framework keamanan berbasis AI guna meningkatkan kemampuan deteksi terhadap varian XSS yang lebih kompleks dan tersembunyi.

VI. REFERENSI

[1] Oh, S., Lee, K., Park, S., Kim, D., & Kim, H. (2023). Poisoned ChatGPT Finds Work for Idle Hands: Exploring Developers' Coding Practices with Insecure Suggestions from Poisoned AI Models. 2024 IEEE Symposium on Security and Privacy (SP), 1141-1159. https://doi.org/10.1109/SP54263.2024.00046.

[2] Khazal, I., & Hussain, M. (2021). Server Side Method to Detect and Prevent Stored XSS Attack. Iraqi Journal for Electrical and Electronic Engineering. https://doi.org/10.37917/ijeee.17.2.8.

[3] Su, H., Xu, L., Chao, H., Li, F., Yuan, Z., Zhou, J., & Huo, W. (2022). A Sanitizer-centric Analysis to Detect Cross-Site Scripting in PHP Programs. 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), 355-365. https://doi.org/10.1109/ISSRE55969.2022.00042.

[4] Sethi, M., Verma, J., Snehi, M., Baggan, V., , V., & Chhabra, G. (2023). Web Server Security Solution for Detecting Cross-site Scripting Attacks in Real-time Using Deep Learning. 2023 International Conference on Artificial Intelligence and Applications (ICAIA) Alliance Technology Conference (ATCON-1), 1-5. https://doi.org/10.1109/ICAIA57370.2023.10169255.

[5] Su, H., Li, F., Xu, L., Hu, W., Sun, Y., Sun, Q., Chao, H., & Huo, W. (2023). Splendor: Static Detection of Stored XSS in Modern Web Applications. Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. https://doi.org/10.1145/3597926.3598116.

[6] Kumar, A., Gupta, A., Mittal, P., Gupta, P., & Varghese, S. (2021). Prevention of XSS attack using Cryptography & API integration with Web Security. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3833910.

[7] T'oth, R., Bisztray, T., & Erdodi, L. (2024). LLMs in Web-Development: Evaluating LLM-Generated PHP code unveiling vulnerabilities and limitations. ArXiv, abs/2404.14459. https://doi.org/10.48550/arXiv.2404.14459.

[8] Khoury, R., Avila, A., Brunelle, J., & Camara, B. (2023). How Secure is Code Generated by ChatGPT?. 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2445-2451. https://doi.org/10.1109/SMC53992.2023.10394237.

[9] Mishra, A., & Juneja, S. (2023). Prevention of Website from Cross Site Scripting. 2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN), 471-473. https://doi.org/10.1109/cictn57981.2023.10140659.

[10] Dora, J., & Nemoga, K. (2021). Ontology for Cross-Site-Scripting (XSS) Attack in Cybersecurity. J. Cybersecur. Priv. 1, 319-339. https://doi.org/10.3390/JCP1020018.

[11] Tolkachova, A., & Zhuravchak, D. (2024). Automate the verification of session cookie attributes. Collection "Information Technology and Security". https://doi.org/10.20535/2411-1031.2024.12.1.306260.

[12] Rathod, P., Gowda, D., M, K., Talekar, P., Daddi, N., Bhairanallikar, A., & G, G. (2024). The Cross-Site Scripting (XSS) Attack: A Comprehensive Review. International Journal of Advanced Research in Science, Communication and Technology. https://doi.org/10.48175/ijarsct-19230.

[13] Karthika, S., Padmavathi, G., Roshni, A., & Varshini, S. (2024). Detecting Cross-Site Scripting Attack using Machine Learning Algorithms. 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), 991-995. https://doi.org/10.23919/INDIACom61295.2024.10499119.

[14] Pardomuan, C., Kurniawan, A., Darus, M., Ariffin, M., & Muliono, Y. (2023). Server-side Cross-site Scripting Detection Powered by HTML Semantic Parsing Inspired by XSS Auditor. Pertanika Journal of Science and Technology. https://doi.org/10.47836/pjst.31.3.14.

[15] Pazos, J., Légaré, J., & Beschastnikh, I. (2021). XSnare: Application-specific client-side cross-site scripting protection. 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 154-165. https://doi.org/10.1007/s10664-023-10323-w.

[16] KirstenS, Manico, J., Williams, J., Wichers, D., Weidman, A., Roman, Jex, A., Smith, A., Knutson, J., Imifos, Yalon, E., Kingthorin, Khanna, V., & Ongers, G. (n.d.). Cross Site Scripting (XSS). Retrieved December 16, 2024, from https://owasp.org/www-community/attacks/xss.

[17] s0md3v. (n.d.). XSStrike: Advanced XSS Detection Suite. Retrieved December 16, 2024, from https://github.com/s0md3v/XSStrike.

[18] Roy, R., Mandal, S., Kavya, S., Dedeepya, S., Singh, A., Reddy, E., Patil, R., & Ramamoorthi, J. (2023). Real-time XSS Vulnerability Detection. 2023 3rd International Conference on Intelligent Technologies (CONIT), 1-6. https://doi.org/10.1109/CONIT59222.2023.10205698.

[19] Arian, L., , N., Nugraha, L., & , ". (2024). Website Penetration Analysis Against XSS Attacks using Payload Method. Journal of Innovation Information Technology and Application (JINITA). https://doi.org/10.35970/jinita.v6i1.2225.

[20] Blancaflor, E., Araullo, E., Corcuera, J., Rivera, J., & Velarde, L. (2023). Vulnerability Assessment on Cross-site scripting attack in a simulated E-commerce platform using BeEF and XSStrike. 2023 13th International Conference on Software Technology and Engineering (ICSTE), 1-6. https://doi.org/10.1109/ICSTE61649.2023.00008.

[21] Anchan, A., Patil, A., S, S., N, S., & S, N. (2023). Dual-Layered Defence Mechanism For Prevention of XSS Attack. 2023 International Conference on Computer, Electronics & Electrical Engineering & their Applications (IC2E3), 1-6. https://doi.org/10.1109/IC2E357697.2023.10262414.

[22] Chandran, A. (2022, May 1). XSStrike: A tool to detect XSS. Medium. https://medium.com/@aswinchandran274/xsstrike-a-tool-to-detect-xss-e6b54b5f6f5b

[23] Alaoui, R., & Nfaoui, E. (2023). Generative Adversarial Network-based Approach for Automated Generation of Adversarial Attacks Against a Deep-Learning based XSS Attack Detection Model. International Journal of Advanced Computer Science and Applications. https://doi.org/10.14569/ijacsa.2023.0140797.