**Jurnal Teknik Informatika (JUTIF)**
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000

# Analysis of Polyglot Obfuscation Techniques against ModSecurity in Preventing Cross-Site Scripting (XSS) and SQL Injection Attacks with Experimental Method

**Nelmiawati[1], Kessy Dealova*[2]**

[1,2]Cyber Security Engineering, Batam State Polytechnic, Indonesia

Email: [2]kessy.dealova@students.polibatam.ac.id

Received : Jun 30, 2025; Revised : Aug 18, 2025; Accepted : Aug 18, 2025; Published : Sep 2, 2025

**Abstract**

Internet use has increased every year, as shown by the percentage of internet users in Indonesia reaching 79.50% in 2024. However, security is something that cannot be ignored, especially with the growing number of Cross-Site Scripting (XSS) and SQL Injection Attacks in web platforms. According to OWASP Top 10 report, these two attacks were listed in 2017 and appeared again in the 2021 version, showing that they are still relevant today. In fact, in June 2024, XSS and SQL Injection vulnerabilities were found in a company, PT. XYZ. One way to mitigate these attacks is by using a Web Application Firewall (WAF) such as ModSecurity, which can protect websites from exploitation. However, previous research found that older versions of ModSecurity had weaknesses that could be bypassed with simple obfuscation techniques. This study aims to analyze the effectiveness of the built-in rules in ModSecurity Core Rule Set (CRS) version 4.7 in handling XSS and SQL Injection payloads with polyglot obfuscation, a method that uses complex character encoding to avoid WAF detection. The research was conducted using an experimental method. This study contributes to improve WAF security by testing against modern obfuscation-based attacks, so that security does not rely solely on the default WAF configuration. The results show that all payloads were detected and blocked by ModSecurity with an HTTP 403 response, proving that the CRS 4.7 built-in rules can effectively protect against XSS and SQL Injection threats.

***Keywords :*** *Core Rule Set (CRS), Cross-Site Scripting, ModSecurity, Polyglot obfuscation, SQL Injection, Web Application Firewall (WAF)*



## 1. PENDAHULUAN

Perkembangan teknologi saat ini semakin pesat. Berbagai perkembangan teknologi dapat membantu pekerjaan manusia dalam kegiatan sehari-hari, salah satunya Internet. Internet merupakan media penghubung jutaan individu dan perangkat di seluruh dunia. Hal ini dibuktikan dengan survei penggunaan Internet di dunia yang mencapai 5.3 miliar pengguna, yang berarti dua pertiga populasi atau setara dengan 68% terhubung secara global ke World Wide Web [1]. Demikian pula, peningkatan pengguna Internet di Indonesia mengalami kenaikan pada tahun 2024, mencapai 79.50% dari total populasi penduduk Indonesia tahun 2023 yang terkoneksi Internet [2]. Tingginya penggunaan Internet ini juga terdata dalam statistik penggunaan platform website seperti Google.com, yang mencatat sebanyak 1.97 miliar [3]. Ini menunjukkan penggunaan Internet memang merupakan kebutuhan penting yang dapat salah satunya untuk mendukung industri atau perusahaan untuk meningkatkan kualitas kinerja karyawan [4]. Di sisi lain, peningkatan ini memicu meningkatnya kerentanan terhadap berbagai bentuk serangan siber yang menargetkan aplikasi web [5].

Berdasarkan penggunaan Internet yang meningkat kemudahan dan kecepatan akses tidaklah cukup [6]. Faktor penting yang perlu diperhatikan agar pengguna merasa nyaman dan aman dalam ber-Internet, yaitu Keamanan Siber [7],[8]. Meskipun bukan hal yang baru, saat ini masih banyak ditemukan insiden serangan siber di Indonesia, salah satunya adalah web *defacement* judi online yang menggunakan teknik serangan seperti kerentanan terhadap serangan *Cross-Site Scripting* (XSS) dan SQL *Injection* [9]. Serangan ini juga diperkuat oleh sebuah penelitian yang menunjukkan bahwa XSS menempati peringkat pertama dengan persentase 27,6%, diikuti oleh SQL Injection sebesar 9,9% [10].

PT. XYZ mendapati log kerentanan serangan XSS melalui alat pemantauan perusahaan yang berasal dari aktivitas pengujian keamanan jaringan dari pihak internal pada bulan Juni 2024. Serangan XSS merupakan sebuah eksploitasi keamanan dimana penyerang menyisipkan kode berbahaya (dalam bentuk Javascript) di sisi klien ke suatu halaman web [11]. Serangan XSS bekerja dengan menyerang pengguna aplikasi, untuk mendapat data dari web *client* melalui tindakan POST, GET, COOKIES dan lain-lain dengan menyuntikkan kode ke dalam output aplikasi web [12]. Apabila skrip tersebut berhasil dijalankan, dapat melewati akun serta keamanan [13]. Selain itu, saat ini skrip serangan juga mengalami peningkatan seperti "*code confusion*" yang membuat skrip sulit dibaca untuk melewati deteksi keamanan [14].

Disamping itu, pada bulan November 2024, serangan SQL *injection* juga ditemukan pada alat pemantauan perusahaan PT. XYZ. Menurut laporan National Vulnerability Database NIST pada 27 September 2024, terdapat kerentanan serangan injeksi pada aplikasi Dingfanzu dengan ID CVE-2024-9294, yang memanfaatkan parameter nama pengguna untuk menyebabkan SQL *injection* [15]. Dari laporan ini menunjukkan bahwa serangan XSS dan SQL *injection* masih menjadi ancaman serius.

Berdasarkan pembahasan di atas, keamanan situs web menjadi salah satu ancaman yang perlu diwaspadai. Salah satu alternatif yang dapat mengatasi permasalahan tersebut yaitu dengan menggunakan Web Application Firewall (WAF) pada web server. WAF membantu penyaringan kueri yang berpotensi berbahaya bagi keamanan web, melakukan penyaringan paket, memblokir lalu lintas HTTP, dan pencatatan (*logging*) [16]. Sedangkan apabila tidak menggunakan WAF, serangan yang terjadi tidak dapat terdeteksi karena tidak adanya pencatatan insiden yang membuat sistem menjadi terbuka terhadap serangan yang berpotensi menimbulkan kerugian [17]. Salah satu aplikasi WAF yang digunakan yaitu ModSecurity. Dimana, ModSecurity merupakan salah satu aplikasi WAF yang *open source* dengan kemampuan yang paling stabil dan efektif dalam menangkal serangan pada aplikasi web [18]. ModSecurity memiliki aturan konfigurasi yang disebut 'SecRules' untuk memantau lalu lintas HTTP(S) secara real-time, melakukan pencatatan (*logging*), dan melakukan penyaringan pada komunikasi HTTP(S) [19].

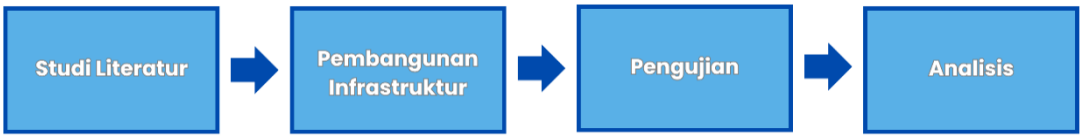
Selanjutnya, muncul celah keamanan yang memungkinkan payload SQL Injection atau XSS melewati deteksi keamanan pada aturan CRS sebelumnya versi 3, sehingga menimbulkan potensi risiko. Hal ini disebut sebagai kerentanan keamanan, dimana kelemahan pada sistem atau mekanisme perlindungan dapat dimanfaatkan penyerang untuk mendapatkan akses tidak sah [20]. Celah tersebut memanfaatkan teknik *obfuscation* seperti karakter Unicode atau fungsi WEIGHT_STRING(). Teknik ini membuat *payload* tampak tidak mencurigakan karena tidak mengandung kata kunci seperti SELECT atau UNION. *Payload obfuscation* tersebut, yang pada penelitian terdahulu berhasil melewati CRS versi 3, sudah berhasil diblokir pada CRS versi 4.7. Dari teknik yang digunakan untuk melewati deteksi keamanan tersebut menunjukkan bahwa serangan XSS dan SQL *injection* mengalami peningkatan kompleksitas yang membuat serangan menjadi lebih menantang [21]. Disamping itu, penelitian sebelumnya menyebutkan bahwa WAF ModSecurity tidak memiliki pembaruan otomatis terhadap kerentanan baru [19], sehingga pengguna harus secara manual mencari informasi terbaru mengenai isu atau celah keamanan pada aturan tersebut.

Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000

Salah satu cara meningkatkan keamanan aplikasi web adalah dengan menggunakan penyamaran kode (*obfuscation*) untuk mempersulit penyerang menganalisis kode dalam upaya melakukan eksploitasi [22]. Akan tetapi, teknik *obfuscation* juga dapat dimanfaatkan oleh penyerang, misalnya untuk melewati keamanan yang diterapkan pada web. Oleh karena itu, studi ini bertujuan untuk melakukan pengujian terhadap aturan bawaan ModSecurity CRS versi 4.7 menggunakan teknik *obfuscation payload* dengan teknik *polyglot* untuk menilai efektivitas deteksi keamanan WAF. *Payload* jenis ini memiliki kemampuan yang sulit dideteksi dan dapat dengan mudah menipu sistem deteksi keamanan berbasis pola tradisional [23]. Selain itu, *payload* ini memungkinkan penggabungan dua atau lebih pola atau perilaku eksploitasi dalam satu input, seperti halnya penggabungan skrip SQL *Injection* dan XSS, sehingga berpotensi membingungkan deteksi WAF [24]. Parameter yang diperhatikan dalam studi ini adalah kode respon HTTP dan ID aturan yang terpicu. Studi ini memberikan kontribusi terhadap evaluasi efektivitas WAF dalam menghadapi ancaman serangan web modern, yang tidak hanya bergantung pada konfigurasi bawaan WAF [25].

## 2. METODE PENELITIAN

Metode penelitian yang digunakan dalam studi ini adalah metode eksperimental. Dalam metode ini, menggunakan empat tahapan yaitu studi literatur, pembangunan infrastruktur, pengujian, dan analisis. Tahapan yang dilakukan dapat dilihat pada Gambar 1.




Gambar 1. Tahapan penelitian

### 2.1. Studi Literatur

Pada tahapan ini, data dan informasi terkait WAF dikumpulkan. Salah satu WAF yang banyak digunakan dalam memblokir serangan melalui protokol HTTP yaitu ModSecurity [26]. Studi ini mengumpulkan referensi dari jurnal, dokumentasi ModSecurity, serta sumber lain terkait teknik *obfuscation polyglot* yang akan digunakan dalam pengujian ModSecurity [27].

### 2.2. Pembangunan Infrastruktur

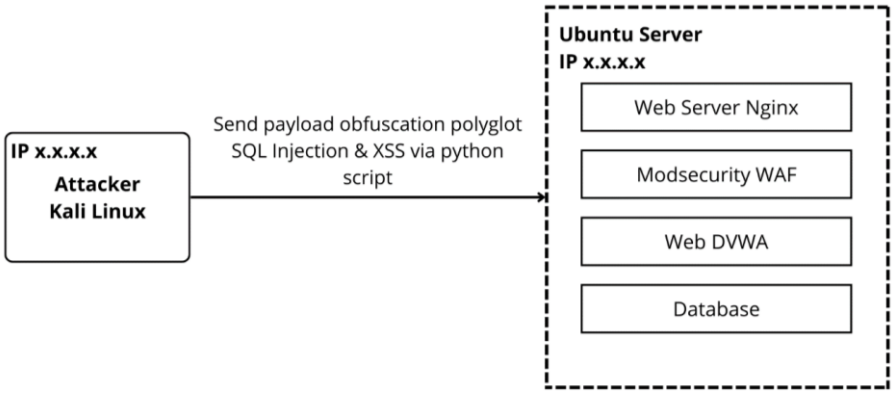
Pembangunan infrastruktur dilakukan dengan menggunakan Damn Vulnerable Web Application (DVWA) sebagai target serangan dan ModSecurity dengan CRS 4.7 sebagai WAF. Server dikonfigurasi agar dapat mencatat log, memantau ID aturan yang terpicu, serta menguji payload secara otomatis dengan Selenium WebDriver.

**a. Implementasi Sistem**

Implementasi sistem merupakan pemasangan ModSecurity pada web server NGINX beserta Web aplikasi DVWA pada server Ubuntu dapat dilihat pada Gambar 2.

Terdapat dua mesin virtual yang digunakan dalam pengujian yaitu mesin virtual Ubuntu server dan mesin virtual Kali Linux. Pada mesin virtual Ubuntu Server terdapat empat komponen utama, yaitu Web Server Nginx, WAF ModSecurity versi 4.7, aplikasi Web (DVWA) sebagai target pengujian dan database untuk menyimpan data DVWA. Sedangkan pada mesin virtual Kali Linux digunakan hanya untuk menjalankan skrip otomatisasi berbasis Python untuk pengiriman payload secara otomatis ke web aplikasi DVWA.

**Jurnal Teknik Informatika (JUTIF)**
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000




Gambar 2 Implementasi Sistem

Pengujian dilakukan dengan cara *attacker* (Kali Linux) mengirimkan *payload* serangan yang telah di-obfuskasi menggunakan teknik *polyglot* ke web server Nginx. Web Server menerima permintaan *payload* serangan tersebut sebelum diteruskan ke aplikasi web DVWA. ModSecurity yang terpasang pada Web Server akan memproses dan menyaring *payload* serangan tersebut untuk menentukan apakah permintaan tersebut akan diterima atau ditolak. Pada studi ini, aturan bawaan WAF ModSecurity diuji secara langsung tanpa modifikasi aturan. Berikut daftar komponen yang dapat dilihat pada Tabel 1. Daftar Komponen

Tabel 1. Daftar Komponen

| No | Nama Komponen | Deskripsi |
|---|---|---|
| 1. | Attacker | Mengirimkan payload serangan melalui kali linux versi 2024.3 |
| 2. | Ubuntu Server | OS versi 24.04 |
| 3. | Nginx | Web server versi 1.26.2 |
| 4. | ModSecurity | WAF versi 3.0.13 dengan aturan CRS 4.7 |
| 5. | DVWA | Aplikasi web yang digunakan untuk pengujian versi 23.0.5 |
| 6. | Payload Polyglot | Teknik *obfuscation* yang digunakan |

**b. Konfigurasi Pengujian**

Konfigurasi pengujian dilakukan dengan memastikan konfigurasi pada file nginx.conf sesuai, pengaturan tingkat keamanan web DVWA, mengaktifkan deteksi ModSecurity, menyiapkan file log, serta menyiapkan skrip otomatisasi pengiriman payload serangan ke web aplikasi target DVWA.

Langkah pertama dalam konfigurasi pengujian adalah penyesuaian file nginx.conf. File ini berfungsi menjalankan web server DVWA dan mengatur integrasi dengan modul keamanan ModSecurity. Pada file ini, ditambahkan perintah *modsecurity on* untuk mengaktifkan WAF, serta menambahkan lokasi file *modsecurity.conf* untuk mengaktifkan aturan CRS dalam mendeteksi serangan. Selain itu, ditambahkan lokasi file log yaitu *modsec_audit.log* untuk mengarahkan agar seluruh aktivitas dicatat pada file ini. Jika terjadi kesalahan, sistem akan mencatatnya ke dalam file *error_example.log* untuk mempermudah melakukan mitigasi apabila terjadi kesalahan dapat dilihat Gambar 3.

Setelah itu, dilakukan pengaturan tingkat keamanan (*security*) pada DVWA pada level *"Low",* dengan tujuan agar perlindungan terhadap serangan hanya bergantung pada ModSecurity bukan dari deteksi keamanan lain. Hal ini dapat dilihat pada Gambar 4. Pastikan juga ModSecurity telah diaktifkan dengan mode *SecRuleEngine On* agar berfungsi sebagai WAF. Hal ini dapat dilihat pada Gambar 5. Terakhir siapkan file log *"modsec_audit.log"* yang dikosongkan terlebih dahulu agar

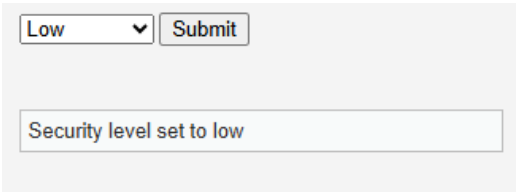
Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000

proses analisis lebih mudah dilakukan, khususnya untuk menelusuri parameter analisis seperti *payload* dan ID yang terpicu dalam file log tersebut.

```
server {
    listen       80;
    server_name  localhost;
    modsecurity  on;
    modsecurity_rules_file /usr/local/nginx/conf/modsecurity.conf;
    access_log /var/log/nginx/modsec_audit.log;
    error_log  /var/log/nginx/error_example.log;
    root   /var/www/dvwa;
    index  index.php index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }
```

Gambar 3 Konfigurasi nginx.conf




Gambar 4 Tingkat keamanan DVWA

```
# -- Rule engine initialization ----------------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine On
```

Gambar 5 Mengaktifkan mode deteksi ModSecurity

### 2.3 Pengujian

Pengujian dilakukan pada kolom input web DVWA XSS yang memiliki tiga jenis kolom input yaitu : XSS reflected yang memanfaatkan skrip berbahaya untuk memantulkan permintaan berbahaya kepada pengguna website yang sudah terinfeksi [28], XSS dom adalah serangan yang terjadi sepenuhnya di sisi klien dengan mengubah struktur Document Object Model (DOM) menggunakan skrip berbahaya [28], dan XSS-stored adalah serangan XSS yang memanfaatkan sebuah kolom pada sebuah website yang dapat menyimpan data secara permanen contohnya kolom komentar [29].

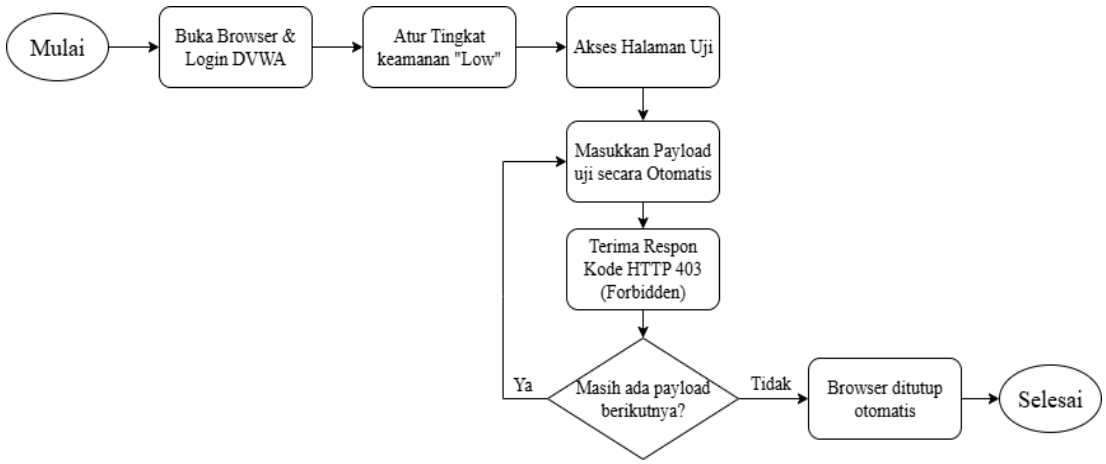
Pada serangan lain pada web DVWA terdapat dua kolom input untuk serangan SQL Injection yaitu: SQL injection yang memungkinkan penyerang menyisipkan *payload* atau kode SQL berbahaya pada kolom input untuk mendapatkan akses ke dalam basis data. Kerentanan ini dapat memberikan akses tidak sah ke data sensitif [26], dan SQL blind serangan yang memanfaatkan pesan kesalahan pada web aplikasi DVWA untuk memungkinkan adanya kerentanan SQL dan mengungkapkan data sensitif [30].

Pada tahap ini, pengujian dilakukan dengan mengirimkan *payload obfuscation polyglot* menggunakan skrip automatisasi pada kali linux ke web aplikasi DVWA. Skrip ini akan mengirimkan payload tersebut pada halaman input SQL injection biasa, SQL blind, XSS reflected, XSS dom, dan XSS Stored. Langkah skrip otomatisasi pengujian dilakukan dengan menggunakan selenium. Skrip dimulai dengan membuka browser dan menampilkan halaman login web DVWA. Halaman login akan diisi dengan *username* “admin” dan *password* “password”. Setelah itu, skrip akan mengarahkan ke

Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000

halaman security pada web aplikasi dvwa dan memilih tingkat level “low” karena hanya berfokus pada keamanan dari aturan WAF ModSecurity.

Tahap selanjutnya yaitu mengakses halaman yang akan diuji. Skrip akan mengirimkan *payload* uji secara otomatis dan setelah dikirimkan akan menerima respon kode HTTP 403 (*Forbidden*). Payload tersebut dikirim satu persatu. Setelah menerima respon kode HTTP, skrip akan kembali memasukkan *payload* uji, selanjutnya hingga seluruh *payload* dilakukan uji. Terakhir setelah seluruh payload dikirimkan skrip akan menutup browser secara otomatis yang artinya pengujian selesai dilakukan. Alur ini dapat dilihat pada Gambar 6.




Gambar 6 Alur pengujian

### 2.4 Analisis

Untuk memastikan keberhasilan studi ini, pengujian terhadap aturan ModSecurity menggunakan *payload obfuscation polyglot* dengan parameter yang diidentifikasi telah dilakukan. Hal ini dapat dilihat pada Tabel 2. Parameter yang dianalisis

Tabel 2. Parameter yang dianalisis

| No | Parameter | Deskripsi |
|---|---|---|
| 1. | Payload obfuscation Polyglot | Jenis payload obfuscation yang diuji |
| 2. | Respon kode HTTP | Memantau status kode HTTP pada log ModSecurity yaitu 403 (*forbidden*) atau 200 (OK) |
| 3. | ID aturan yang terpicu | ID aturan yang terpicu dalam aturan serangan *XSS* dan *SQL Injection* |

Analisis dilakukan dengan mengevaluasi tingkat deteksi ModSecurity terhadap *payload* yang diuji. Kode respons HTTP digunakan sebagai indikator keberhasilan deteksi, dan log dianalisis untuk mengidentifikasi ID rule yang terpicu. Pada studi ini, seluruh pengujian menghasilkan kode HTTP 403, sehingga tidak ditemukan kasus kode HTTP 200. Validasi dilakukan dengan memastikan konsistensi antara hasil yang tampil pada web browser dan log ModSecurity, sehingga setiap deteksi oleh WAF terkonfirmasi pada kedua sisi.

## 3. HASIL

Pengujian terhadap efektivitas aturan bawaan ModSecurity dengan CRS versi 4.7 dalam mendeteksi dan memblokir serangan menggunakan payload obfuscation polyglot yaitu gabungan skrip

Vol. 6, No. 4, August 2025, Page. 2540-2549
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.4.5000

SQL Injection dan XSS telah dilakukan. Pengujian dilakukan pada web DVWA dengan mengirimkan *payload* ke beberapa halaman jenis serangan pada web DVWA yaitu: SQL injection, SQL blind, XSS reflected, XSS stored, dan XSS dom. Hasil pengujian menunjukkan bahwa seluruh *payload* pengujian berhasil diblokir dengan respon HTTP 403 (Forbidden). Hal ini dapat dilihat pada Tabel 3 dan penjelasan ID aturan yang terpicu dapat dilihat pada Table 4 di lampiran. Selain respon kode HTTP, studi ini juga memperhatikan parameter berupa aturan ID yang terpicu selama pengujian. Setiap *payload* umumnya memicu sekitar 15–20 aturan ID, dan satu aturan ID dapat terpicu lebih dari lima kali dalam satu jenis serangan. Kondisi ini disebabkan karena ModSecurity mendeteksi berdasarkan *payload* yang diterima, bukan jenis serangannya, sementara seluruh skenario pengujian menggunakan *payload* yang sama. Oleh karena itu, hasil menunjukkan deskripsi aturan ID yang terpicu secara keseluruhan, yang dapat dilihat pada **Kesalahan! Sumber referensi tidak ditemukan.** di lampiran.

Tabel 3. Hasil Respon Kode HTTP pada *Payload* Uji

| No | Teknik penymaranan payload polyglot | Hasil respon kode HTTP | | | | |
|---|---|---|---|---|---|---|
| | | SQL Injection | SQL blind | XSS reflected | XSS stored | XSS dom |
| 1. | Extraneous Open Brackets | 403 | 403 | 403 | 403 | 403 |
| 2. | Malformed A Tags | 403 | 403 | 403 | 403 | 403 |
| 3. | Spaces and Meta Chars Before the JavaScript in Images for XSS | 403 | 403 | 403 | 403 | 403 |
| 4. | URL Encoding | 403 | 403 | 403 | 403 | 403 |
| 5. | Hex Entity Encoding | 403 | 403 | 403 | 403 | 403 |
| 6. | End Title Tag | 403 | 403 | 403 | 403 | 403 |
| 7. | Protocol Resolution Bypass | 403 | 403 | 403 | 403 | 403 |
| 8. | fromCharCode | 403 | 403 | 403 | 403 | 403 |
| 9. | Malformed IMG Tags | 403 | 403 | 403 | 403 | 403 |
| 10. | Embedded Tab | 403 | 403 | 403 | 403 | 403 |
| 11. | Default SRC Tag by Leaving it Empty | 403 | 403 | 403 | 403 | 403 |
| 12. | Half Open HTML/JavaScript XSS Vector | 403 | 403 | 403 | 403 | 403 |
| 13. | SVG and Non Alphabet Character Obfuscation | 403 | 403 | 403 | 403 | 403 |
| 14. | Embedded Encode Tab | 403 | 403 | 403 | 403 | 403 |
| 15. | Escaping JavaScript Escapes | 403 | 403 | 403 | 403 | 403 |
| 16. | INPUT Image | 403 | 403 | 403 | 403 | 403 |
| 17. | Break Out Injection via Comment | 403 | 403 | 403 | 403 | 403 |
| 18. | Break Up XSS Attack with Embedded Carriage Return | 403 | 403 | 403 | 403 | 403 |
| 19. | Default SRC Tag by Leaving it out Entirely | 403 | 403 | 403 | 403 | 403 |
| 20. | Default SRC Tag to Get Past Filters that Check SRC Domain | 403 | 403 | 403 | 403 | 403 |

## 4. PEMBAHASAN

Hasil pengujian menunjukkan bahwa seluruh *payload obfuscation polyglot* berhasil dideteksi dan diblokir oleh aturan bawaan ModSecurity dengan Core Rule Set (CRS) versi 4.7. Semua serangan, baik SQL Injection maupun XSS menghasilkan respon HTTP 403 (Forbidden) dari server target. Hal ini

menandakan bahwa CRS 4.7 memiliki kapabilitas yang baik dalam mengenali dan menanggapi pola serangan yang telah disamarkan menggunakan berbagai teknik *obfuscation*.

Teknik *obfuscation* yang digunakan dalam payload mencakup ASCII encoding, URL encoding, karakter hexadecimal, manipulasi struktur HTML, entitas HTML, serta penyamaran dengan JavaScript. Sistem deteksi ModSecurity mampu mengenali pola tersembunyi tanpa memerlukan proses decoding tambahan. Keberhasilan ini menunjukkan bahwa CRS versi terbaru telah mengalami peningkatan efektivitas dalam menangani varian serangan yang kompleks dibanding versi sebelumnya.

Beberapa *payload* pada penelitian sebelumnya yang berhasil melewati filter ModSecurity dengan aturan CRS versi 3.x. juga dilakukan pengujian pada CRS versi penelitian yaitu 4.7. Sebagai contoh, *payload* yang menggunakan fungsi WEIGHT_STRING(), karakter Unicode, dan Whitespace untuk membuat *payload* tampak tidak mencurigakan karena tidak mengandung SELECT, UNION [31], selanjutnya sumber lain menggunakan junk karakter seperti +, - [32], penggunaan karakter komentar [33], dan teknik non-alphanumeric [34]. *Payload* tersebut juga berhasil diblokir oleh aturan CRS versi 4.7.

Salah satu temuan yang paling menonjol adalah aturan ID yang menggunakan pustaka LibInjection dalam mendeteksi struktur serangan. Rule ID 941100 untuk XSS dan 942100 untuk SQL Injection merupakan aturan yang paling sering terpicu selama pengujian. LibInjection tidak hanya mendeteksi berdasarkan kata kunci spesifik serangan, tetapi juga berdasarkan struktur umum serangan, sehingga mampu mengenali pola yang telah dimanipulasi dengan *obfuscation*. Ini menjadikan LibInjection sebagai komponen kunci dalam lapisan deteksi awal ModSecurity.

Hasil studi membantu meningkatkan desain aturan WAF di masa depan dengan memberikan pemahaman lebih mendalam mengenai teknik *obfuscation* yang digunakan oleh penyerang. Dengan pemahaman ini, aturan deteksi dapat dibuat lebih adaptif, dan strategi mitigasi serangan pada aturan WAF menjadi lebih efektif. Disamping itu, studi ini juga menyoroti skenario *bypass* yang gagal, yaitu *payload* yang tidak berhasil lolos deteksi pada CRS 4.7. Analisis terhadap payload gagal ini menunjukkan bahwa aturan terbaru tidak hanya mampu menangkal serangan sederhana, tetapi juga serangan *polyglot* yang menggabungkan dua jenis serangan sekaligus dalam satu input. Temuan ini memperkuat bukti bahwa CRS 4.7 memiliki kemampuan deteksi yang lebih baik dan mampu menghadapi ancaman yang telah disamarkan secara kompleks dibanding versi sebelumnya.

## 5. KESIMPULAN

Kerentanan XSS dan SQL Injection bukanlah jenis serangan yang baru. Namun, dua kerentanan ini masih banyak ditemukan dalam serangan terhadap aplikasi web, termasuk yang terdeteksi melalui log pemantauan SIEM di lingkungan PT. XYZ. Studi ini berhasil menguji efektivitas aturan bawaan ModSecurity dengan OWASP Core Rule Set (CRS) versi 4.7 dalam mendeteksi dan memblokir serangan XSS dan SQL Injection yang telah disamarkan menggunakan teknik obfuscation polyglot. Hasil dari studi ini menunjukkan bahwa CRS versi 4.7 mampu mendeteksi ancaman lebih awal dan efektif dalam mengenali berbagai bentuk serangan yang telah disamarkan. Hal ini terlihat pada seluruh *payload obfuscation polyglot* yang merupakan penggabungan dua jenis serangan berhasil diblokir oleh aturan bawaan ModSecurity CRS 4.7 dalam satu input tanpa perlu memisahkan jenis serangannya terlebih dahulu. Selanjutnya, teknik obfuscation yang diuji meliputi URL *encoding*, ASCII, karakter *hexadecimal*, HTML entity, manipulasi struktur HTML, dan JavaScript. Semua teknik ini terdeteksi oleh WAF tanpa perlu proses *decoding* terlebih dahulu, sesuai dengan log audit ModSecurity. Berdasarkan hasil log, terdapat aturan yang paling sering terpicu yaitu Pustaka LibInjection, yaitu rule ID 941100 untuk XSS dan 942100 untuk SQL Injection. Hal ini menunjukkan bahwa LibInjection memiliki efektivitas dalam mengenali serangan berbasis struktur, termasuk yang telah diobfuskasi.

Meskipun hasil pengujian menunjukkan efektivitas yang tinggi, teknik *obfuscation* tetap merupakan tantangan serius dalam dunia keamanan siber. *Obfuscation* memungkinkan penyerang menyamarkan *payload* berbahaya agar tampak tidak mencurigakan oleh sistem deteksi. Sebagai pengembangan ke depan, studi ini dapat selanjutnya diperluas dengan menguji efektivitas teknik *obfuscation* lain terhadap aplikasi WAF yang berbeda. Sehingga cakupan analisis meningkat dan ketahanan sistem terhadap ancaman yang terus berkembang dapat lebih terjamin keamanannya.

## REFERENSI

[1] rebootonline.com, "Website Statistics Report 2024," Reboot. Accessed: Nov. 14, 2024. [Online]. Available: https://www.rebootonline.com/website-statistics/

[2] M. Arif, "Press Conference Hasil Survei Penetrasi Internet Indonesia 2024," 2024. [Online]. Available: https://apjii.or.id/berita/d/apjii-jumlah-pengguna-internet-indonesia-tembus-221-juta-orang

[3] A. D. Riyanto, "Hootsuite (We are Social): Data Digital Indonesia 2024," Feb. 2024. [Online]. Available: https://andi.link/hootsuite-we-are-social-data-digital-indonesia-2024/

[4] M. Irfan, "Analisis Implementasi Kerentanan Website Laboratiorium Jurusan Teknik Informatika dan Komputer Menggunakan OpenVAS dan Acunetix Vulnerabilty Scanner," *Repos. Politek. Negeri Jakarta*, 2024, [Online]. Available: https://repository.pnj.ac.id/id/eprint/21267/

[5] S. Alazmi and D. C. De Leon, "A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners," *IEEE Access*, vol. 10, pp. 33200–33219, 2022, doi: 10.1109/ACCESS.2022.3161522.

[6] R. Riska and H. Alamsyah, "Penerapan Sistem Keamanan Web Menggunakan Metode Web Aplication Firewall," *J. Amplif. J. Ilm. Bid. Tek. Elektro Dan Komput.*, vol. 11, no. 1, pp. 37–42, 2021, doi: 10.33369/jamplifier.v11i1.16683.

[7] M. Annas, R. T. Adek, and Y. Afrillia, "Web Application Firewall (WAF) Design to Detect and Anticipate Hacking in Web-Based Applications," *J. Adv. Comput. Knowl. Algorithms*, vol. 1, no. 3, p. 52, 2024, doi: 10.29103/jacka.v1i3.16315.

[8] Andreya E., "Antisipasi Bersama Tingkatkan Sistem dan Cegah Serangan Siber," Sep. 2022. [Online]. Available: https://aptika.kominfo.go.id/2022/09/antisipasi-bersama-tingkatkan-sistem-dan-cegah-serangan-siber/

[9] I. R. Team, "WEB DEFACEMENT : JUDI ONLINE," JAKARTA, Jun. 2023. Accessed: Nov. 14, 2024. [Online]. Available: https://www.bssn.go.id/langkah-langkah-penanggulangan-insiden-web-defacement-judi-online/

[10] D. Lee, B. Steed, Y. Liu, and O. Ezenwoye, "Tutorial: A Lightweight Web Application for Software Vulnerability Demonstration," *Proc. - 2021 IEEE Secur. Dev. Conf. SecDev 2021*, pp. 5–6, 2021, doi: 10.1109/SecDev51306.2021.00014.

[11] A. A. Chandra, A. Turmudi Zy, and A. Nugroho, "PENERAPAN TEKNIK PENETRATION TESTING TERHADAP CROSS SITE SCRIPTING (XSS) DALAM PENGEMBANGAN WEBSITE," *Rabit J. Teknol. dan Sist. Inf. Univrab*, vol. 9, no. 2, pp. 262–270, Jul. 2024, doi: 10.36341/rabit.v9i2.4822.

[12] S. Suroto and A. Asman, "Ancaman Terhadap Keamanan Informasi Oleh Serangan Cross-Site Scripting (Xss) Dan Metode Pencegahannya," *Zo. Komput.*, vol. 11, no. 1, pp. 11–19, 2021, doi: https://doi.org/10.37776/zk.v11i1.658.

[13] A. S. Hakim, T. A. Cahyanto, and H. Azizah, "Serangan cross-site scripting (XSS) berdasarkan base metric CVSS V.2," *J. Smart Teknol.*, vol. 2, no. 1, 2020, [Online]. Available: http://jurnal.unmuhjember.ac.id/index.php/JST/article/view/3839

[14] M. Liu, B. Zhang, W. Chen, and X. Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities," *IEEE Access*, vol. 7, pp. 182004–182016, 2019, doi: 10.1109/ACCESS.2019.2960449.

[15] nvd.nist.gov, "CVE-2024-9294 Detail," National Vulnerability Database. Accessed: Nov. 14, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-9294

[16] Hardianto and T. Subari, "Analisis Cyber Crime handling pada Aplikasi Web dengan WAF

ModSecurity," *Petir*, vol. 16, no. 1, pp. 91–99, Apr. 2023, doi: 10.33322/petir.v16i1.1910.
[17] G. E. Cárdenas Rosero, C. P. Guevara Vega, and P. Landeta-López, "Website Protection: An Evaluation of the Web Application Firewall," *Data Metadata*, vol. 4, 2025, doi: 10.56294/DM2025190.
[18] R. A. Muzaki, O. C. Briliyant, M. A. Hasditama, and H. Ritchi, "Improving Security of Web-Based Application Using ModSecurity and Reverse Proxy in Web Application Firewall," *2020 Int. Work. Big Data Inf. Secur. IWBIS 2020*, pp. 85–90, 2020, doi: 10.1109/IWBIS50925.2020.9255601.
[19] S. D. Utama, F. Ferdian, A. Bahtiar, E. Pratama, and N. Arista, "Web Application Firewall Menggunakan ModSecurity," *IET Inf. Secur.*, vol. 17, no. 1, pp. 900–926, 2019, [Online]. Available: http://www.sicherheitsforschung-magdeburg.de/uploads/journal/MJS_061_Bijjou_Bypassing.pdf
[20] N. Tewari and G. Datt, "A Study on the Systematic Review of Security Vulnerabilities of Popular Web Browsers," *Proc. Int. Conf. Technol. Adv. Innov. ICTAI 2021*, pp. 314–318, 2021, doi: 10.1109/ICTAI53825.2021.9673463.
[21] G. Areo, "Advanced Cybersecurity Strategies for Detecting and Preventing Cross-Site Scripting (XSS) Attacks," no. November, p. 8, 2024, [Online]. Available: https://www.researchgate.net/publication/385492153
[22] R. A. AlSufaian, K. H. AlQahtani, R. M. AlAjmi, R. A. AlMoussa, R. A. AlGhamdi, and N. A. Saqib, "Web Application Security Using Obfuscation," no. June, pp. 0–6, 2012.
[23] D. Garg, "Uncovering XSS Polyglot Payload Detection with Machine Learning : Advancing Web Security Against Complex Threats," pp. 0–19, 2024, doi: https://doi.org/10.21203/rs.3.rs-5564100/v1.
[24] L. Koch *et al.*, "On the Abuse and Detection of Polyglot Files," *WWW 2025 - Proc. ACM Web Conf.*, pp. 4810–4822, 2025, doi: 10.1145/3696410.3714814.
[25] M. Alagoz, M. S. Tok, and K. Bicakci, "Exploring and Improving the Usability of ModSecurity Web Application Firewall," *14th Int. Conf. Inf. Secur. Cryptology, ISCTURKEY 2021 - Proc.*, no. December, pp. 51–56, 2021, doi: 10.1109/ISCTURKEY53027.2021.9654294.
[26] E. Mulyo, "Analisa WAF (Web Application Firewall) Menggunakan Nginx Terhadap Serangan Sql Injection," 2021, [Online]. Available: http://digilib.mercubuana.ac.id/
[27] OWASP, "XSS Filter Evasion Cheat Sheet." [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html
[28] E. Malays and S. Sakti, "Analisis pengamanan Website dari Serangan Cross Site Script ( XSS ) dengan htmlspecialchars dan strip _ tags," vol. 25, no. 1, pp. 177–183, 2024, doi: https://doi.org/10.37817/tekinfo.v25i1.
[29] D. G. Sembiring, "Apa itu Serangan Cross-Site Scripting (XSS)?," 2024, *Information Technology Certification Center, Jakarta Barat*. [Online]. Available: https://itcc.itpln.ac.id/apa-itu-serangan-cross-site-scripting-xss/
[30] M. D. Khoiroh *et al.*, "PENETRATION TESTING UNTUK MENGUJI KERENTANAN SISTEM INFORMASI PEMERINTAH DAERAH," *J. Sist. dan Teknol. Inf.*, vol. 06, no. 2, pp. 1–5, 2024.
[31] S. Tahiri, "How we bypassed libModSecurity aka ModSecurity," Linkedin Article. Accessed: Nov. 16, 2024. [Online]. Available: https://www.linkedin.com/pulse/how-we-bypassed-libmodsecurity-aka-modsecurity-soufiane-tahiri/
[32] W. El Labban, "Bypassing a Web Application Firewall," 2024. Accessed: Nov. 15, 2024. [Online]. Available: https://wissam-labban.com/Projects/Modesecurity WAF Bypassing.pdf
[33] qazbnm456, "Bypass the latest CRS v3.1.0 rules of SQL injection," github.com.
[34] A. Chowdhary, K. Jha, and M. Zhao, "Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications," *Sensors*, vol. 23, no. 18, Sep. 2023, doi: 10.3390/s23188014.