Analisis Performa *ModSecurity Core Rule Set* Menggunakan *GoTestWAF* untuk Mengidentifikasi Serangan dan Teknik Bypass pada Aplikasi Web

Satriawan Desmana[1], Krisna Nuresa Qodri[2], Ratih[3], Bella Adinda Putri[4], Muhammad Abdul Muin[5]
[1,2,3,4,5] Program Studi Rekayasa Keamanan Siber, Jurusan Komputer dan Bisnis, Politeknik Negeri Cilacap, Indonesia
Email: [1]satriawan@pnc.ac.id, [2]krisnanuresa@pnc.ac.id, [3]ratih@pnc.ac.id, [4]belladinda@pnc.ac.id, [5]abdulmuin@pnc.ac.id

Abstrak - Keamanan aplikasi web menghadapi tantangan yang semakin kompleks dengan meningkatnya serangan siber seperti *SQL Injection*, *Cross-Site Scripting* (XSS), Command Injection, serta berbagai teknik evasion yang dirancang khusus untuk menghindari mekanisme deteksi konvensional. ModSecurity sebagai *Web Application Firewall* (WAF) open-source telah banyak digunakan karena fleksibilitas dan integrasinya dengan OWASP *Core Rule Set* (CRS). Namun demikian, efektivitas ModSecurity sangat bergantung pada kualitas dan pembaruan rule set yang diimplementasikan. Penelitian ini bertujuan mengevaluasi performa ModSecurity versi 3 dengan CRS 4.19.0 dalam mendeteksi serangan modern menggunakan framework pengujian otomatis GoTestWAF. Pengujian dilaksanakan pada lingkungan terkontrol melalui analisis *true-positive*, *true-negative*, dan false-negative terhadap 816 payload berbahaya maupun legitimate. Hasil penelitian menunjukkan ModSecurity hanya mampu memblokir 45,44% dari total *payload* berbahaya, sementara 54,56% berhasil melewati perlindungan. Selain itu, 17,73% traffic aman salah diblokir (*false positive*), yang berpotensi mengganggu operasional aplikasi. Kelemahan terutama ditemukan pada payload berukuran besar, teknik obfuscation, encoding kompleks, dan struktur request non-standar. Secara keseluruhan, konfigurasi default CRS 4.19.0 pada paranoia level 1 belum memadai menghadapi serangan kontemporer. Optimalisasi diperlukan melalui peningkatan paranoia level, aktivasi optional rules, tuning aturan, dan penambahan custom rules. Penelitian ini memberikan kontribusi empiris bagi peningkatan implementasi WAF open-source pada aplikasi web masa kini.

Kata Kunci - ModSecurity; Web Application Firewall; OWASP Core Rule Set; CRS 4.19.0; GoTestWAF.

Abstract - Web application security has become increasingly critical as the intensity and complexity of cyberattacks—such as SQL Injection, Cross-Site Scripting (XSS), Command Injection, and various evasion techniques designed to bypass traditional detection mechanisms—continue to escalate. ModSecurity, an open-source Web Application Firewall (WAF), is widely adopted for its flexibility and integration with the OWASP Core Rule Set (CRS). However, its effectiveness is highly influenced by the quality and frequency of updates to the rule sets it employs. This study aims to evaluate the detection performance of ModSecurity version 3 with CRS 4.19.0 against modern attack patterns using the automated testing framework GoTestWAF. Experiments were conducted in a controlled environment through true-positive, true-negative, and false-negative analysis on 816 malicious and benign payloads. The results show that ModSecurity was only able to block 45.44% of malicious payloads, while 54.56% successfully bypassed the protection. Additionally, 17.73% of legitimate traffic was incorrectly blocked (false positive rate), which may disrupt application operations. The weaknesses were predominantly observed in large payloads, obfuscation techniques, complex encodings, and non-standard request structures. Overall, the default configuration of CRS 4.19.0 at paranoia level 1 remains insufficient to counter modern attacks. Optimization is required through higher paranoia levels, activation of optional rules, tuning of existing rules, and the addition of custom detection rules. This research provides empirical contributions to strengthening the implementation of open-source WAFs in contemporary web applications.

Keywords - ModSecurity; Web Application Firewall; OWASP Core Rule Set; CRS 4.19.0; GoTestWAF.

I. PENDAHULUAN

Keamanan aplikasi web telah menjadi perhatian utama dalam ekosistem digital kontemporer, terutama dengan semakin meningkatnya intensitas dan kompleksitas serangan siber terhadap infrastruktur sistem informasi. Berbagai bentuk serangan seperti SQL Injection, *Cross-Site Scripting* (XSS), *Command Injection*, serta teknik evasion yang dirancang khusus untuk melewati mekanisme pertahanan tradisional, terus mengalami evolusi dan menuntut solusi proteksi yang lebih adaptif [1], [2]. Dalam konteks ini, ModSecurity sebagai *Web Application Firewall* (WAF) *open-source* telah menjadi salah satu pilihan yang banyak diimplementasikan karena fleksibilitas arsitekturnya dan ketersediaannya pada berbagai platform *server web* [3].

Keberhasilan ModSecurity dalam mendeteksi dan mencegah serangan sangat bergantung pada kualitas aturan yang tertanam dalam OWASP Core Rule Set (CRS), yang secara berkala diperbarui untuk mengakomodasi pola serangan baru yang terus bermunculan. Pada tahun 2024, OWASP merilis Core Rule Set versi 4.19.0, yang menawarkan sejumlah peningkatan signifikan dalam hal sensitivitas aturan, optimasi penanganan *false positive*, serta perluasan cakupan deteksi terhadap ancaman modern [4], [5]. Pembaruan ini diharapkan dapat meningkatkan efektivitas WAF dalam menghadapi landscape ancaman siber yang semakin kompleks.

Namun demikian, sejumlah penelitian terdahulu menunjukkan bahwa kemampuan ModSecurity dalam mendeteksi serangan tidak sepenuhnya optimal, terutama ketika berhadapan dengan teknik bypass yang lebih canggih seperti pengacakan *payload*, penggunaan encoding tidak lazim, rekayasa sintaks, dan fragmentasi parameter [6], [7]. Penelitian empiris yang menguji performa ModSecurity versi terbaru, khususnya dengan CRS 4.19.0, terhadap

variasi serangan dan teknik bypass menggunakan alat uji otomatis masih relatif terbatas. Padahal, alat seperti GoTestWAF mampu melakukan evaluasi komprehensif dengan mensimulasikan puluhan jenis serangan serta pola bypass yang menggambarkan ancaman nyata secara lebih akurat [6], [8].

Celah penelitian ini menjadi dasar perlunya pengujian performa ModSecurity secara terstruktur dan sistematis menggunakan CRS versi terbaru. Beberapa studi sebelumnya telah mengidentifikasi kelemahan pada konfigurasi default ModSecurity, namun belum secara spesifik mengevaluasi CRS 4.19.0 dengan pendekatan pengujian yang komprehensif. Penelitian terkini menunjukkan bahwa WAF berbasis signature seperti ModSecurity masih memiliki keterbatasan dalam menangani serangan yang menggunakan teknik *obfuscation* dan *polymorphic* payload [9], [10], [11], [12], [13].

Berdasarkan kondisi tersebut, penelitian ini dilakukan untuk menjawab beberapa permasalahan, yaitu: pertama, bagaimana performa ModSecurity Core Rule Set 4.19.0 dalam mendeteksi beragam serangan web ketika diuji menggunakan GoTestWAF; kedua, teknik bypass apa saja yang mampu melewati perlindungan ModSecurity; dan ketiga, sejauh mana tingkat keberhasilan deteksi dan kelemahan aturan CRS 4.19.0 ketika dihadapkan pada serangan modern. Tujuan dari penelitian ini adalah untuk menganalisis kemampuan deteksi ModSecurity CRS 4.19.0, mengidentifikasi payload yang berhasil melakukan bypass, serta mengukur performa deteksi secara kuantitatif sehingga dapat memberikan rekomendasi peningkatan konfigurasi keamanan.

Penelitian ini memberikan kontribusi teoretis berupa penambahan literatur mengenai evaluasi performa WAF open-source, khususnya terkait efektivitas aturan CRS versi terbaru dalam menghadapi ancaman kontemporer [12], [14]. Manfaat praktisnya adalah menyediakan informasi empiris bagi administrator sistem, pengembang aplikasi web, maupun praktisi keamanan siber untuk memahami tingkat ketahanan ModSecurity di lingkungan nyata serta potensi celah yang masih dapat dieksploitasi oleh penyerang [12], [13], [15].

Meskipun demikian, penelitian ini memiliki beberapa batasan yang perlu diperhatikan. Pertama, pengujian hanya dilakukan pada ModSecurity dengan CRS 4.19.0 sehingga hasil penelitian tidak dapat digeneralisasi ke versi CRS lainnya. Kedua, serangan yang diuji sebatas yang disediakan oleh GoTestWAF sehingga tidak mencakup seluruh kemungkinan teknik bypass yang ada. Ketiga, lingkungan pengujian berada pada sistem terkontrol dan bukan aplikasi produksi sehingga kondisi jaringan dan beban *traffic* yang sesungguhnya tidak terwakili. Keempat, penelitian tidak melibatkan teknologi *machine learning* atau *artificial intelligence* sebagai mekanisme deteksi tambahan [16], [17], [18].

Dengan demikian, penelitian ini berupaya memberikan pemahaman mendalam mengenai sejauh mana CRS 4.19.0 mampu memberikan perlindungan terhadap aplikasi web modern, sekaligus membuka ruang untuk pengembangan strategi mitigasi lanjutan agar WAF berbasis ModSecurity dapat dioptimalkan pada implementasi nyata di berbagai organisasi.

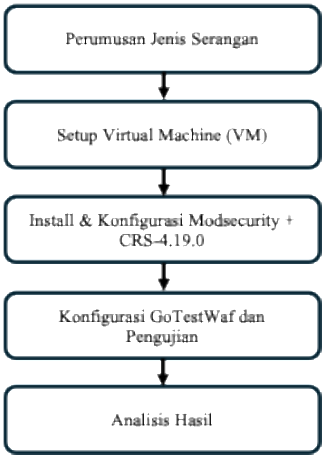
II. METODE PENELITIAN

Penelitian ini menggunakan pendekatan eksperimen kuantitatif untuk mengevaluasi performa ModSecurity dengan OWASP Core Rule Set (CRS) versi 4.19.0 dalam mendeteksi serangan aplikasi web dan teknik bypass menggunakan framework pengujian otomatis GoTestWAF. Metode ini dipilih karena memungkinkan proses pengujian yang terstandar, mudah direplikasi, serta menghasilkan data kuantitatif terkait keberhasilan deteksi dan kegagalan perlindungan (*bypass*) secara objektif.

A. Desain Penelitian

Penelitian ini menerapkan desain experimental benchmarking, yaitu membandingkan respons ModSecurity CRS 4.19.0 terhadap sejumlah payload serangan dan variasi bypass yang dikirimkan secara otomatis oleh GoTestWAF. Setiap request dicatat hasilnya berupa status deteksi (*block/allow*), kategori serangan, serta payload yang berhasil melewati proteksi. Evaluasi dilakukan pada lingkungan terkontrol sehingga hasil yang diperoleh tidak terpengaruh oleh variabel jaringan eksternal atau beban *traffic* yang tidak terprediksi. Seluruh pengujian dilakukan pada server virtual dengan pengaturan identik untuk menjamin konsistensi hasil.

Gambar 1 menunjukkan alur penelitian yang dimulai dari persiapan lingkungan pengujian, instalasi dan konfigurasi ModSecurity dengan CRS 4.19.0, pelaksanaan pengujian menggunakan GoTestWAF, hingga analisis hasil dan penarikan kesimpulan



Gambar 1. Alur Penelitian

Alur penelitian pada Gambar 1 menggambarkan metodologi pengujian yang komprehensif dan terstruktur. Tahap pertama adalah persiapan lingkungan pengujian, yang meliputi instalasi sistem operasi Ubuntu Server 22.04 LTS pada mesin virtual, konfigurasi Apache web server versi 2.4.58, dan verifikasi spesifikasi hardware sesuai dengan kebutuhan minimum yang tercantum pada Tabel 1. Pada tahap ini juga dilakukan hardening dasar sistem operasi dan konfigurasi network untuk memastikan environment pengujian terisolasi dari interferensi eksternal.

Tahap kedua merupakan proses instalasi dan konfigurasi ModSecurity versi 3 beserta OWASP Core Rule Set (CRS) 4.19.0. ModSecurity dikonfigurasi dalam mode blocking (SecRuleEngine On) sehingga setiap payload yang terdeteksi berbahaya akan langsung diblokir sebelum mencapai aplikasi web di belakangnya. Seluruh core rules pada CRS 4.19.0 diaktifkan sesuai konfigurasi default paranoia level 1 (PL1) tanpa modifikasi apapun untuk memastikan pengujian dilakukan pada kondisi standar yang umum diimplementasikan. Konfigurasi audit logging juga diaktifkan untuk merekam seluruh aktivitas deteksi secara detail.

Tahap ketiga adalah pelaksanaan pengujian menggunakan framework GoTestWAF. Pada tahap ini, GoTestWAF mengirimkan total 816 payload yang terdiri dari 675 payload berbahaya (malicious) untuk pengujian true-positive dan 141 payload legitimate untuk pengujian true-negative. Payload berbahaya mencakup berbagai kategori serangan seperti SQL Injection, XSS, RCE, LFI, Path Traversal, SSRF, NoSQL Injection, dan lainnya, dengan berbagai variasi teknik obfuscation dan encoding. Setiap payload dikirim secara otomatis melalui HTTP/HTTPS request dengan interval waktu yang terkontrol, dan respons dari ModSecurity dicatat secara real-time untuk analisis lebih lanjut.

Tahap keempat melakukan analisis komprehensif terhadap hasil pengujian. Analisis mencakup perhitungan persentase blocked, bypassed, false positive, dan kategorisasi berdasarkan jenis serangan. Data log ModSecurity yang tersimpan dalam format audit log diparse menggunakan script Python 3.10 untuk ekstraksi informasi detail seperti rule ID yang terpicu, severity level, pesan error, dan alasan blocking. Hasil parsing kemudian divisualisasikan dalam bentuk tabel dan grafik untuk memudahkan interpretasi data. Pada tahap ini juga dilakukan identifikasi pola payload yang berhasil bypass untuk memahami kelemahan rule set yang digunakan.

Tahap kelima adalah penarikan kesimpulan dan penyusunan rekomendasi berdasarkan temuan empiris. Kesimpulan dirumuskan dengan mengacu pada hasil analisis kuantitatif dan kualitatif, termasuk identifikasi kelemahan deteksi pada kategori serangan tertentu. Rekomendasi yang dihasilkan mencakup strategi optimalisasi konfigurasi WAF untuk implementasi produksi, seperti peningkatan paranoia level, aktivasi optional rules, pengembangan custom rules, dan implementasi whitelist management untuk mengurangi false positive.

B. Lingkungan Pengujian

1. Spesifikasi Server

   Tabel 1 menunjukkan spesifikasi teknis server yang digunakan dalam penelitian ini. Pemilihan spesifikasi ini didasarkan pada pertimbangan kebutuhan minimum untuk menjalankan ModSecurity dengan performa optimal tanpa *bottleneck* pada sumber daya komputasi.

Tabel 1. Spesifikasi Server

Komponen	*Spesifikasi*
Sistem Operasi	Ubuntu Server 22.04 LTS
Web Server	Apache 2.4.58 dengan ModSecurity v3
Rule Set	OWASP CRS versi 4.19.0
Processor	2 vCPU
RAM	2 GB
Storage	25 GB

2. Perangkat Lunak Pengujian

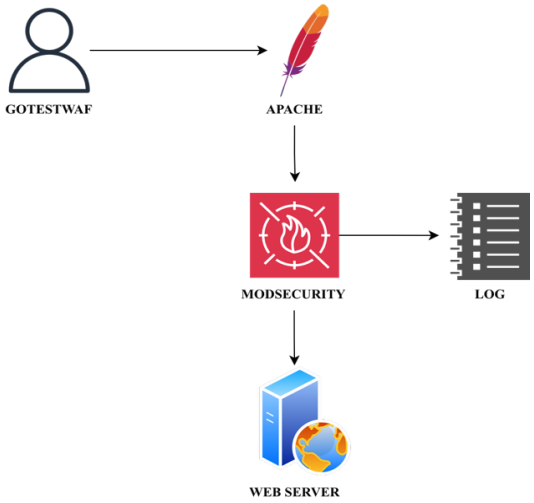
   a. GoTestWAF (versi terbaru dari repository Wallarm), yang merupakan *framework* pengujian otomatis untuk WAF dengan kemampuan mensimulasikan berbagai jenis serangan dan teknik bypass.

   b. Python 3.10 untuk pengelolaan log tambahan, termasuk parsing hasil pengujian dan visualisasi data.

   c. Apache Benchmark untuk pengujian performa dan stabilitas server selama pengujian berlangsung.

C. Arsitektur Sistem Pengujian

Gambar 2 menunjukkan arsitektur pengujian di mana peneliti menggunakan GoTestWAF untuk mengirim payload serangan ke web server Apache yang telah dipasangi ModSecurity dan OWASP CRS 4.19.0. Setiap request yang masuk dianalisis oleh ModSecurity, kemudian diberikan respons berupa *block* atau *allow*. Hasil deteksi dan *rule* yang terpicu dicatat ke dalam log untuk analisis lebih lanjut. Seluruh proses berjalan pada server virtual sebagai lingkungan terkontrol untuk memastikan hasil pengujian lebih stabil dan konsisten.



Gambar 2. Arsitektur Sistem Pengujian

Arsitektur ini dirancang untuk meminimalkan interferensi eksternal dan memaksimalkan akurasi hasil pengujian. Pemisahan antara mesin pengujian (GoTestWAF) dan target server (ModSecurity) memungkinkan observasi yang objektif terhadap performa

WAF tanpa bias dari proses lokal. Komponen arsitektur tersebut meliputi:

a. Attack Simulator (GoTestWAF): Mengirim 816 payload dengan variasi jenis serangan dan teknik bypass.
b. Web Application Firewall (ModSecurity + CRS 4.19.0): Melakukan inspeksi dan filtering terhadap setiap request.
c. Web Server (Apache): Backend application yang menjadi target proteksi.
d. Logging System: Merekam seluruh aktivitas untuk analisis post-testing.

D.Teknik Pengambilan dan Analisis Data

Pengambilan data dalam penelitian ini dilakukan melalui tiga mekanisme utama yang terintegrasi untuk memastikan akurasi dan kelengkapan hasil pengujian.

1. Pengumpulan Data Otomatis

GoTestWAF dikonfigurasi untuk mengirimkan payload secara sistematis dengan parameter sebagai berikut:

a. URL target: http://[IP_Server_Pengujian]/
b. Format output: JSON untuk memudahkan parsing dan analisis programatik.
c. Timeout per request: 10 detik untuk menghindari false unresolved.
d. Retry mechanism: 3 kali percobaan untuk request yang gagal karena network issue.
e. Request interval: 100ms antar payload untuk menghindari rate limiting.
f. User-Agent: Randomized untuk mensimulasikan variasi client.
g. Test mode: Blocking mode untuk mengukur kemampuan preventif WAF.

Setiap request yang dikirimkan oleh GoTestWAF menghasilkan respons berupa HTTP status code yang kemudian dikategorikan sebagai:

a. Blocked: HTTP status 403 (Forbidden) atau respons yang mengandung ModSecurity rejection page, mengindikasikan payload berhasil dideteksi dan diblokir
b. Bypassed: HTTP status 200 (OK) yang menunjukkan payload berhasil mencapai aplikasi web tanpa terdeteksi oleh WAF
c. Unresolved: Request timeout, HTTP 5xx errors, atau error yang tidak dapat dikategorikan dengan jelas, biasanya disebabkan oleh payload yang terlalu besar atau malformed
d. Failed: Request yang gagal dikirim karena masalah teknis seperti network error, DNS resolution failure, atau connection refused

2. Pencatatan Log ModSecurity

ModSecurity dikonfigurasi untuk mencatat seluruh aktivitas deteksi dalam file log terpisah dengan konfigurasi seperti pada gambar 3:

```
SecAuditLogParts ABIJDEFHZ
SecAuditLogType Serial
SecAuditLog /var/log/modsec_audit.log
SecDebugLog /var/log/modsec_debug.log
SecDebugLogLevel 3
```

Gambar 3. Konfigurasi Pencatatan Log ModSecurity

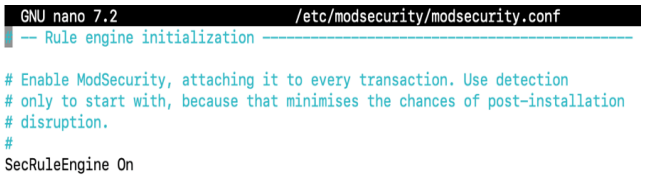
Gambar 3 merupakan jenis log yang dikumpulkan. Jenis Log yang dikumpulkan antara lain:

a. Audit Log (/var/log/modsec_audit.log): Mencatat seluruh request yang diproses beserta rule yang terpicu, dengan format standar ModSecurity Audit Log Format yang mencakup:
   - Section A: Audit log header (timestamp, unique ID)
   - Section B: Request headers
   - Section C: Request body
   - Section E: Expression
   - Section F: Response headers
   - Section H: Audit log trailer (rule messages, tags, severity)
   - Section I: Indeks Aturan (Rule ID)
   - Section Z: Final boundary marker
b. Debug Log (/var/log/modsec_debug.log): Mencatat informasi detail proses inspeksi untuk debugging dan troubleshooting, termasuk:
   - Rule processing sequence
   - Variable transformation
   - Pattern matching details
   - Performance metrics per rule

III. HASIL DAN PEMBAHASAN

Pengujian pada penelitian ini dilakukan pada sebuah server uji yang telah dilengkapi dengan ModSecurity versi 3 sebagai *Web Application Firewall* (WAF) yang terpasang di depan web server. ModSecurity dikonfigurasi untuk bekerja dalam mode aktif (*blocking mode*), sehingga setiap payload yang dikategorikan berbahaya akan langsung diblokir sebelum mencapai aplikasi web di belakangnya.

Gambar 3 menunjukkan status ModSecurity yang aktif dan berfungsi dengan baik pada sistem pengujian. Verifikasi status ini penting untuk memastikan bahwa seluruh pengujian berjalan dalam kondisi WAF yang sepenuhnya operasional.



Gambar 4. Status *Modsecurity*

Sebagai basis aturan deteksi, penelitian ini menggunakan OWASP ModSecurity Core Rule Set (CRS) versi 4.19.0. Seluruh core rules pada CRS 4.19.0 diaktifkan sesuai pengaturan standar, tanpa penambahan custom rule atau modifikasi khusus. Dengan demikian, hasil pengujian yang diperoleh mencerminkan kemampuan konfigurasi default CRS 4.19.0 dalam menangani berbagai jenis serangan yang disimulasikan oleh GoTestWAF.

Gambar 4 menampilkan konfigurasi rules ModSecurity yang telah dimuat ke dalam sistem. Dapat diamati bahwa seluruh rule set dari CRS 4.19.0 telah terload dengan benar dan siap untuk melakukan inspeksi terhadap traffic yang masuk.

```
root@server1:/etc/apache2/modsecurity-crs/coreruleset-4.19.0/rules# ls
asp-dotnet-errors.data
iis-errors.data
java-classes.data
lfi-os-files.data
php-errors.data
php-function-names-933150.data
php-variables.data
REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf.example
REQUEST-901-INITIALIZATION.conf
REQUEST-905-COMMON-EXCEPTIONS.conf
REQUEST-911-METHOD-ENFORCEMENT.conf
REQUEST-913-SCANNER-DETECTION.conf
REQUEST-920-PROTOCOL-ENFORCEMENT.conf
REQUEST-921-PROTOCOL-ATTACK.conf
REQUEST-922-MULTIPART-ATTACK.conf
REQUEST-930-APPLICATION-ATTACK-LFI.conf
REQUEST-931-APPLICATION-ATTACK-RFI.conf
REQUEST-932-APPLICATION-ATTACK-RCE.conf
REQUEST-933-APPLICATION-ATTACK-PHP.conf
REQUEST-934-APPLICATION-ATTACK-GENERIC.conf
REQUEST-941-APPLICATION-ATTACK-XSS.conf
REQUEST-942-APPLICATION-ATTACK-SQLI.conf
REQUEST-943-APPLICATION-ATTACK-SESSION-FIXATION.conf
REQUEST-944-APPLICATION-ATTACK-JAVA.conf
REQUEST-949-BLOCKING-EVALUATION.conf
RESPONSE-950-DATA-LEAKAGES.conf
RESPONSE-951-DATA-LEAKAGES-SQL.conf
RESPONSE-952-DATA-LEAKAGES-JAVA.conf
RESPONSE-953-DATA-LEAKAGES-PHP.conf
RESPONSE-954-DATA-LEAKAGES-IIS.conf
RESPONSE-955-WEB-SHELLS.conf
RESPONSE-956-DATA-LEAKAGES-RUBY.conf
RESPONSE-959-BLOCKING-EVALUATION.conf
RESPONSE-980-CORRELATION.conf
RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf.example
```

Gambar 5. Rules Modsecurity

Dengan konfigurasi default tersebut, pengujian selanjutnya dilakukan menggunakan GoTestWAF untuk mengevaluasi kemampuan ModSecurity CRS 4.19.0 dalam memblokir serangan (*true-positive*), mengidentifikasi permintaan yang aman (*true-negative*), dan mendeteksi payload berbahaya yang justru lolos (*false-negative*). Hasil pengujian menunjukkan bahwa meskipun sebagian serangan berhasil diblokir, masih terdapat sejumlah besar payload berbahaya yang mampu melewati perlindungan WAF.

A. Hasil True-Positive Test

True-positive menggambarkan kemampuan WAF dalam mendeteksi dan memblokir serangan yang benar-benar berbahaya. Dari total 675 payload yang dikirimkan pada rangkaian uji komunitas dan OWASP *test suite*, ModSecurity hanya mampu memblokir 304 *payload* (45,44%), sementara 365 payload (54,56%) berhasil melewati perlindungan dan dikategorikan sebagai *bypassed*.

Tabel 2 menyajikan hasil detail pengujian true-positive untuk setiap kategori serangan yang diuji. Dapat diamati bahwa tingkat keberhasilan deteksi sangat bervariasi tergantung pada jenis dan karakteristik payload yang digunakan.

Tabel 2. Hasil *True-Positive Tests*

Test Set	*Test Case*	*%*	*Blocked*	*Bypassed*	*Unresolved*	*Sent*	*Failed*
community	community-128kb-rce	0.00	0	0	1	1	0
community	community-128kb-sqli	0.00	0	0	1	1	0
community	community-128kb-xss	0.00	0	0	1	1	0
community	community-16kb-rce	100.00	1	0	0	1	0
community	community-16kb-sqli	100.00	1	0	0	1	0
community	community-16kb-xss	100.00	1	0	0	1	0

community	community-32kb-rce	100.00	1	0	0	1	0
community	community-32kb-sqli	100.00	1	0	0	1	0
community	community-32kb-xss	100.00	1	0	0	1	0
community	community-64kb-rce	100.00	1	0	0	1	0
community	community-64kb-sqli	100.00	1	0	0	1	0
community	community-64kb-xss	100.00	1	0	0	1	0
community	community-8kb-rce	100.00	1	0	0	1	0
community	community-8kb-sqli	100.00	1	0	0	1	0
community	community-8kb-xss	100.00	1	0	0	1	0
community	community-lfi	100.00	8	0	0	8	0
community	community-lfi-multipart	0.00	0	2	0	2	0
community	community-rce	50.00	2	2	0	4	0
community	community-rce-rawrequests	100.00	3	0	0	3	0
community	community-sqli	100.00	12	0	0	12	0
community	community-user-agent	66.67	6	3	0	9	0
community	community-xss	90.38	94	10	0	104	0
community	community-xxe	0.00	0	2	0	2	0
owasp	crlf	42.86	3	4	0	7	0
owasp	ldap-injection	8.33	2	22	0	24	0
owasp	mail-injection	12.50	3	21	0	24	0
owasp	nosql-injection	34.00	17	33	0	50	0
owasp	path-traversal	25.00	5	15	0	20	0
owasp	rce	33.33	2	4	0	6	0
owasp	rce-urlparam	33.33	3	6	0	9	0
owasp	rce-urlpath	0.00	0	3	0	3	0
owasp	shell-injection	18.75	6	26	0	32	0
owasp	sql-injection	39.58	19	29	0	48	0
owasp	ss-include	45.83	11	13	0	24	0
owasp	sst-injection	29.17	7	17	0	24	0
owasp	xml-injection	0.00	0	6	1	7	0
owasp	xss-scripting	36.16	81	143	0	224	0
owasp-api	graphql	0.00	0	0	0	0	0
owasp-api	graphql-post	0.00	0	0	0	0	0
owasp-api	grpc	0.00	0	0	0	0	0
owasp-api	non-crud	100.00	2	0	0	2	0
owasp-api	rest	57.14	4	3	0	7	0
owasp-api	soap	40.00	2	3	0	5	0

Hasil ini mengindikasikan bahwa meskipun ModSecurity telah dikonfigurasi dengan CRS versi 4.19.0 dalam kondisi standar, kemampuan deteksinya masih belum optimal, terutama terhadap variasi payload yang menggunakan teknik obfuscation, encoding kompleks, maupun struktur permintaan yang tidak biasa.

Payload dari kategori *community test set*, yang umumnya dibuat oleh komunitas keamanan dengan kreativitas tinggi, memberikan gambaran menarik. Misalnya, serangan berukuran besar seperti community-128kb-rce, community-128kb-sqli, dan community-128kb-xss tidak terdeteksi sama sekali, terlihat dari nilai 0% pada kolom blocked dan satu payload yang tidak dapat diproses (*unresolved*). Hal ini menunjukkan bahwa ModSecurity memiliki keterbatasan dalam menangani *payload* berukuran sangat besar, kemungkinan disebabkan oleh limitasi buffer atau timeout dalam proses inspeksi. Di sisi lain, payload dengan ukuran lebih kecil seperti 8kb, 16kb, dan 32kb justru memiliki tingkat deteksi yang sangat tinggi, yaitu 100%.

Namun demikian, terdapat beberapa jenis payload yang sangat mudah melewati perlindungan ModSecurity. Temuan yang paling mengkhawatirkan adalah:

a. *Local File Inclusion (LFI):* Serangan community-lfi-multipart tidak terdeteksi sama sekali dengan dua payload lolos, sementara owasp-lfi menunjukkan hasil yang lebih buruk dengan 35 dari 35 *payload* berhasil bypass (0% detection rate). Hal ini mengindikasikan kelemahan serius dalam deteksi pola file inclusion, terutama yang menggunakan teknik multipart encoding

b. *Remote Code Execution (RCE):* community-rce hanya terdeteksi separuh (50%), menunjukkan WAF kesulitan mendeteksi eksekusi kode yang menggunakan variasi parameter maupun struktur request yang kompleks. Owasp-rce menunjukkan performa yang lebih rendah dengan hanya 33,33% payload terblokir.

c. *NoSQL Injection:* Seluruh 45 payload owasp-nosql berhasil melewati deteksi (0%), mengindikasikan bahwa rule set default CRS 4.19.0 belum memadai

untuk menangani serangan terhadap database NoSQL yang semakin populer.

d. *Path Traversal*: Seperti halnya NoSQL injection, seluruh 85 payload owasp-path-traversal tidak terdeteksi (0%), menunjukkan celah signifikan dalam proteksi terhadap akses file tidak sah.

e. *Server-Side Request Forgery* (SSRF): Seluruh 36 payload owasp-ssrf lolos tanpa terdeteksi (0%), yang merupakan keprihatinan serius mengingat SSRF dapat digunakan untuk mengakses resource internal yang seharusnya tidak dapat diakses dari luar.

f. *Cross-Site Scripting* (XSS): community-xss memiliki tingkat deteksi cukup tinggi (90,38%), namun masih ada 10 payload yang lolos, menunjukkan pola XSS yang lebih kompleks tidak terjangkau oleh aturan default CRS. Owasp-xss menunjukkan performa yang lebih rendah dengan hanya 37,35% payload terblokir.

Secara keseluruhan, hasil true-positive test yang diringkas pada Tabel 3 menunjukkan gambaran performa ModSecurity CRS 4.19.0 yang kurang memuaskan:

Tabel 3 Ringkasan True-Positive Test

Kategori	*Nilai*
Total Sent	675
Blocked (Resolved)	304 (45.44%)
Bypassed (Resolved)	365 (54.56%)
Unresolved	6 (0.89%)
Failed	0

Fakta bahwa lebih dari separuh serangan (54,56%) berhasil melewati WAF menunjukkan bahwa konfigurasi *default* ModSecurity CRS 4.19.0 belum memberikan perlindungan yang memadai terhadap serangan modern. Banyaknya *payload* yang lolos menegaskan bahwa aturan pada paranoia level 1 (PL1) tidak cukup kuat untuk menghadapi variasi serangan yang lebih canggih atau payload buatan komunitas yang mengeksploitasi celah *parsing*, *encoding*, dan struktur permintaan HTTP.

Temuan ini sejalan dengan penelitian sebelumnya yang mengidentifikasi bahwa konfigurasi default WAF seringkali tidak cukup untuk proteksi yang komprehensif [9], [10], [11], [13], [19]. Beberapa faktor yang berkontribusi terhadap rendahnya tingkat deteksi antara lain:

1. *Limitasi Paranoia Level*: Penggunaan paranoia level 1 (default) memberikan keseimbangan antara deteksi dan *false positive*, namun mengorbankan sensitivitas terhadap serangan yang lebih canggih.
2. Keterbatasan *Rule Coverage*: Meskipun CRS 4.19.0 telah diperbarui, beberapa kategori serangan seperti NoSQL injection dan SSRF masih memiliki coverage yang terbatas.
3. Teknik Obfuscation: Penyerang modern menggunakan teknik encoding dan obfuscation yang kompleks yang sulit dideteksi oleh *rule* berbasis *signature*
4. *Payload Size Limitation*: Ketidakmampuan menangani *payload* berukuran besar (>128kb) menunjukkan adanya limitasi teknis dalam proses inspeksi.

B. Hasil True-Negative Test

True-negative mengukur kemampuan ModSecurity membiarkan permintaan yang aman tanpa salah memblokirnya. Dari 141 *payload non-malicious*, ModSecurity mampu mengidentifikasi 116 *payload* (82,27%) sebagai aman, namun masih terdapat 25 *payload* (17,73%) yang salah diblokir (*false-positive*).

Tabel 4 menunjukkan hasil pengujian *true-negative* yang dilakukan untuk mengukur tingkat *false positive* dari konfigurasi ModSecurity CRS 4.19.0

Tabel 4 Deteksi *True-Negative Test*

Kategori	*Keterangan*
Test Set	False-Pos
Test Case	Texts
Persentase (%)	82.27
Blocked	25
Bypassed	116
Unresolved	0
Sent	141
Failed	0

Tabel 5 menyajikan ringkasan hasil pengujian true-negative yang memberikan gambaran komprehensif tentang tingkat akurasi ModSecurity dalam mengidentifikasi traffic legitimate.

Tabel 5 Ringkasan *True-Negative Test*

Kategori	*Nilai*
Total Sent	141
Blocked (False Positive)	25 (17.73%)
Allowed (True Negative)	116 (82.27%)
Bypassed	0
Failed	0

Persentase false positive sebesar 17,73% menunjukkan bahwa meskipun konfigurasi default cukup baik dalam mengurangi kesalahan deteksi, masih ada risiko permintaan sah terblokir yang dapat mengganggu aplikasi nyata apabila diterapkan tanpa tuning tambahan. Dalam lingkungan produksi, tingkat false positive ini dapat menyebabkan frustrasi pengguna dan berpotensi mengurangi kepercayaan terhadap sistem.

Temuan ini menunjukkan bahwa ModSecurity dengan CRS 4.19.0 masih memiliki keterbatasan dalam mengenali karakteristik permintaan normal yang sering kali menyerupai pola input tertentu yang dianggap mencurigakan oleh aturan dasar CRS paranoia level 1 (PL1). False positive dapat terjadi karena beberapa faktor, antara lain: (1) rule yang terlalu ketat dalam mendeteksi pola tertentu; (2) kurangnya whitelist untuk aplikasi spesifik; (3) karakteristik traffic aplikasi yang tidak umum namun sah;

dan (4) kompleksitas struktur data yang dikirimkan oleh aplikasi modern.

Penelitian sebelumnya menunjukkan bahwa konfigurasi default WAF dapat menghasilkan false positive yang signifikan, tergantung pada rule-set dan karakteristik aplikasi [20], [21]. Hasil ini sejalan dengan temuan penelitian ini, yang menekankan pentingnya penyesuaian *rule-set, tuning*, atau mekanisme tambahan agar WAF mampu membedakan permintaan sah dari serangan yang nyata. Namun demikian, untuk implementasi produksi, tingkat *false positive* ini perlu diturunkan melalui proses tuning yang sistematis. Beberapa strategi yang dapat diterapkan meliputi: aktivasi whitelist untuk *endpoint* tertentu, penyesuaian *threshold scoring*, dan penambahan *exception rules* untuk fitur aplikasi yang memang memerlukan input dengan pola khusus.

Hasil *true-negative test* memperlihatkan bahwa meskipun ModSecurity mampu menyaring sebagian besar permintaan aman, masih terdapat ruang perbaikan signifikan agar WAF dapat beroperasi dengan lebih akurat dan stabil pada lingkungan produksi. Optimalisasi ini penting untuk mencegah gangguan terhadap operasional aplikasi yang dapat berdampak pada *user experience* dan produktivitas organisasi.

IV. KESIMPULAN DAN SARAN

Penelitian ini mengevaluasi performa ModSecurity versi 3 dengan OWASP Core Rule Set 4.19.0 menggunakan framework pengujian GoTestWAF untuk mengidentifikasi kemampuan deteksi serangan dan teknik bypass pada aplikasi web. Hasil pengujian menunjukkan bahwa konfigurasi default ModSecurity CRS 4.19.0 pada paranoia level 1 belum memberikan perlindungan yang memadai terhadap serangan modern, dengan hanya mampu memblokir 45,44% dari 675 *payload* berbahaya yang diujikan. Kategori serangan seperti NoSQL Injection, Path Traversal, SSRF, dan *file inclusion* menunjukkan tingkat bypass 100%, mengindikasikan gap signifikan dalam rule coverage untuk teknologi dan teknik serangan kontemporer. Sementara itu, hasil *true-negative* test menunjukkan tingkat false positive sebesar 17,73%, yang meskipun masih dalam batas wajar namun berpotensi mengganggu operasional aplikasi pada implementasi produksi. Temuan ini menegaskan bahwa optimalisasi konfigurasi melalui peningkatan paranoia level, aktivasi optional rules, pengembangan custom rules, dan implementasi whitelist management merupakan langkah esensial untuk meningkatkan efektivitas *ModSecurity* dalam menghadapi landscape ancaman siber yang semakin kompleks.

Berdasarkan temuan penelitian, beberapa saran direkomendasikan untuk pengembangan lebih lanjut. Pertama, penelitian selanjutnya perlu melakukan pengujian komparatif antara berbagai paranoia level (PL1-PL4) untuk mengidentifikasi konfigurasi optimal yang menyeimbangkan antara tingkat deteksi dan false positive rate. Kedua, evaluasi performa *ModSecurity* pada lingkungan produksi dengan beban traffic nyata diperlukan untuk memvalidasi hasil pengujian dalam kondisi operasional yang sesungguhnya. Ketiga, pengembangan custom rule set yang dioptimalkan untuk kategori serangan dengan tingkat bypass tinggi, khususnya *NoSQL* Injection dan SSRF, dapat menjadi kontribusi signifikan bagi komunitas keamanan siber. Keempat, integrasi ModSecurity dengan teknologi machine learning atau *artificial intelligence* berpotensi meningkatkan kemampuan deteksi terhadap serangan *zero-day* dan teknik obfuscation yang kompleks. Kelima, studi longitudinal yang mengevaluasi efektivitas berbagai versi CRS terhadap evolusi teknik serangan dari waktu ke waktu akan memberikan pemahaman yang lebih komprehensif tentang dinamika perlombaan antara mekanisme pertahanan dan teknik penyerangan. Implementasi saran-saran tersebut diharapkan dapat menghasilkan solusi WAF yang lebih robust dan adaptif dalam melindungi aplikasi web dari ancaman siber yang terus berkembang.

DAFTAR PUSTAKA

[1] - Robinson, M. Akbar, and M. A. Fadhly Ridha, "SQL Injection and Cross Site Scripting Prevention using OWASP ModSecurity Web Application Firewall," *JOIV : International Journal on Informatics Visualization*, vol. 2, no. 4, pp. 286–292, Aug. 2018, doi: 10.30630/joiv.2.4.107.

[2] S. E. Sadat, M. F. Naseri, and K. Salamzada, "Identifying and Mitigating Web Application Vulnerabilities: A Comparative Study of Countermeasures and Tools," *International Journal Software Engineering and Computer Science (IJSECS)*, vol. 4, no. 3, pp. 1109–1127, Dec. 2024, doi: 10.35870/ijsecs.v4i3.3138.

[3] K. D. D. Ayunda, A. Widjajarto, and A. Budiono, "Implementation and Analysis ModSecurity on Web-Based Application with OWASP Standards," *JATISI (Jurnal Teknik Informatika dan Sistem Informasi)*, vol. 8, no. 3, pp. 1638–1650, 2021.

[4] A. Reyes Narváez, M. Curipallo Martínez, E. Reyes Narváez, F. Lara, E. P. Reyes Narváez, and H. Barba Molina, "Evaluation Framework for False Positives in Open-Source WAFs Based on OWASP CRS Paranoia Levels: A Systematic Approach for Comparative Measurement," in *The XXXIII Conference on Electrical and Electronic Engineering*, Basel Switzerland: MDPI, Nov. 2025, p. 1. doi: 10.3390/engproc2025115001.

[5] V. Babaey and A. Ravindran, "GenSQLi: A Generative Artificial Intelligence Framework for Automatically Securing Web Application Firewalls Against Structured Query Language Injection Attacks," *Future Internet*, vol. 17, no. 1, p. 8, Dec. 2024, doi: 10.3390/fi17010008.

[6] Q. Wang *et al.*, "Break the Wall from Bottom: Automated Discovery of Protocol-Level Evasion Vulnerabilities in Web Application Firewalls," in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, May 2024, pp. 185–202. doi: 10.1109/SP54263.2024.00129.

[7] H. Humaira, A. Hadiana, and H. Ashaury, "Analisis Ketahanan Web Application Firewall Terhadap Serangan SQL Injection," *Jurnal Ilmiah Wahana*

Pendidikan, vol. 10, no. 5, Jan. 2024, doi: 10.5281/zenodo.10526246.

[8] A. MK, K. S. S. Bala, S. S. T. Sonti, and J. KP, "An empirical study on the evaluation and enhancement of OWASP CRS (Core Rule Set) in ModSecurity," *Comput Secur*, vol. 160, p. 104714, Jan. 2026, doi: 10.1016/j.cose.2025.104714.

[9] M. Amouei, M. Rezvani, and M. Fateh, "RAT: Reinforcement-Learning-Driven and Adaptive Testing for Vulnerability Discovery in Web Application Firewalls," Dec. 2023, doi: 10.1109/TDSC.2021.3095417.

[10] K. Li, H. Yang, and W. Visser, "Evolutionary Multi-Task Injection Testing on Web Application Firewalls," Jun. 2022.

[11] V. Babaey and A. Ravindran, "GenXSS: an AI-Driven Framework for Automated Detection of XSS Attacks in WAFs," Apr. 2025.

[12] C. Wu, J. Chen, S. Zhu, W. Feng, R. Du, and Y. Xiang, "WAFBOOSTER: Automatic Boosting of WAF Security Against Mutated Malicious Payloads," Jan. 2025.

[13] S. A. Akhavani, B. Jabiyev, B. Kallus, C. Topcuoglu, S. Bratus, and E. Kirda, "WAFFLED: Exploiting Parsing Discrepancies to Bypass Web Application Firewalls," Oct. 2025.

[14] Z. Qu, X. Ling, T. Wang, X. Chen, S. Ji, and C. Wu, "AdvSQLi: Generating Adversarial SQL Injections Against Real-World WAF-as-a-Service," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 2623–2638, 2024, doi: 10.1109/TIFS.2024.3350911.

[15] G. Floris *et al.*, "ModSec-AdvLearn: Countering Adversarial SQL Injections with Robust Machine Learning," May 2025, doi: 10.1109/TIFS.2025.3583234.

[16] A. Shaheed and M. H. D. B. Kurdy, "Web Application Firewall Using Machine Learning and Features Engineering," *Security and Communication Networks*, vol. 2022, pp. 1–14, Jun. 2022, doi: 10.1155/2022/5280158.

[17] S. Toprak and A. Yavuz, "Web Application Firewall Based on Anomaly Detection using Deep Learning," *Acta Infologica*, vol. 0, no. 0, pp. 0–0, Oct. 2022, doi: 10.26650/acin.1039042.

[18] B. Dawadi, B. Adhikari, and D. Srivastava, "Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks," *Sensors*, vol. 23, no. 4, p. 2073, Feb. 2023, doi: 10.3390/s23042073.

[19] N. Nelmiawati and K. Dealova, "Analysis of Polyglot Obfuscation Techniques against ModSecurity in Preventing Cross-Site Scripting (XSS) and SQL Injection Attacks with Experimental Method," *Jurnal Teknik Informatika (Jutif)*, vol. 6, no. 4, pp. 2540–2549, Sep. 2025, doi: 10.52436/1.jutif.2025.6.4.5000.

[20] A. Reyes Narváez, M. Curipallo Martínez, E. Reyes Narváez, F. Lara, E. P. Reyes Narváez, and H. Barba Molina, "Evaluation Framework for False Positives in Open-Source WAFs Based on OWASP CRS Paranoia Levels: A Systematic Approach for Comparative Measurement," in *The XXXIII Conference on Electrical and Electronic Engineering*, Basel Switzerland: MDPI, Nov. 2025, p. 1. doi: 10.3390/engproc2025115001.

[21] C. Scano *et al.*, "ModSec-Learn: Boosting ModSecurity with Machine Learning," Jun. 2024, doi: 10.1007/978-3-031-76459-2_3.