electronics

MDPI

Article

AutoEx: A Log-Driven Framework for Automated Exception Rule Generation in OWASP CRS-Based Web Application Firewalls

Aldrin Reyes Narváez *, Michael Curipallo Martínez and Hernan Barba Molina

Departamento de Electrónica, Telecomunicaciones y Redes de Información, Escuela Politécnica Nacional, Quito 170525, Ecuador; michael.curipallo@epn.edu.ec (M.C.M.); hernan.barba@epn.edu.ec (H.B.M.)
* Correspondence: aldrin.reyesn@epn.edu.ec

Abstract

Web Application Firewalls (WAFs) based on the OWASP Core Rule Set (CRS) are widely utilized to protect web applications; however, higher CRS paranoia levels, while improving attack coverage, often lead to a significant increase in false positives, thus creating substantial operational challenges. To address this issue, this article proposes AutoEx, a systematic framework for the automated generation of secure exclusion rules in CRS-based, rule-driven WAFs. The framework analyzes WAF audit logs and traces of legitimate traffic to identify recurring false-positive patterns and derive exception rules without disabling core detection mechanisms. AutoEx is evaluated across multiple CRS paranoia levels using controlled traffic scenarios, enabling a comparative assessment of its impact on false-positive reduction and detection effectiveness. The results demonstrate that false-positive rates decrease from 100% to mean residual values between 32% and 46% under scenarios involving simple input datasets and to below 2% when sufficiently representative datasets are utilized for exception generation. Additionally, the detection effectiveness remains at 100% when all intentionally introduced attack payloads are correctly identified and blocked, regardless of input dataset complexity or configured paranoia level. Furthermore, the processing latency before and after applying AutoEx is discussed. These findings show that log-driven automated exception rule generation can substantially improve the operational usability of CRS-based WAFs. The proposed framework provides a practical and scalable solution to support secure WAF tuning in complex web applications, reducing manual effort, and minimizing the risk of security degradation caused by overly permissive configurations.

Keywords: Web Application Firewall; OWASP Core Rule Set; paranoia levels; false positives; rule exclusions; WAF tuning; audit log analysis

Check for updates

Academic Editor: Christos J. Bouras

Received: 3 February 2026
Revised: 3 March 2026
Accepted: 9 March 2026
Published: 29 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Web applications constitute a fundamental component of modern electronic and information systems, supporting critical services across multiple sectors. In electronic commerce, web applications enable online transactions and customer data management, making them prime targets for cyberattacks due to the financial impact and sensitive information involved in it [1,2]. In the healthcare sector, web-based systems are widely utilized for electronic health records and clinical information exchange, where security breaches may directly affect patient safety and data privacy [3].

Similarly, industrial control interfaces increasingly rely on web technologies to provide remote monitoring and management of operational processes, exposing Industrial Control Systems (ICSs) to web-based attack vectors that can disrupt critical infrastructure [4].

The increasing exposure of web-based systems to the Internet has made web applications one of the primary targets of cyberattacks. Among the most prevalent application-layer threats are injection-based attacks, such as SQL injection, command injection, and Cross-Site Scripting (XSS), which exploit improper validation of HTTP request parameters. SQL injection attacks manipulate backend databases and remain a prevalent cause of data breaches in web applications [5]. Likewise, command injection attacks allow adversaries to execute arbitrary system commands on the underlying operating system through vulnerable application interfaces, potentially leading to full system compromise [6]. In addition, cross-site scripting (XSS) attacks target the client-side execution context by injecting malicious scripts into trusted web pages, enabling session hijacking and data exfiltration [7]. To mitigate these threats, Web Application Firewalls (WAFs) are widely deployed as an application-layer security control, capable of inspecting HTTP traffic and enforcing security policies beyond the capabilities of traditional network firewalls [8].

Among multiple open-source WAF solutions, such as Modsecurity (https://github.com/owasp-modsecurity/ModSecurity, accessed on 10 January 2026) and Coraza (https://github.com/corazawaf/coraza, accessed on 10 January 2026), the OWASP Core Rule Set (CRS) [9,10] has become a de facto standard for WAF deployments by providing a comprehensive and continuously maintained set of generic attack-detection rules. The CRS has been designed to be engine-agnostic, enabling its deployment across different rule-processing platforms while supporting multiple paranoia levels [11]. The anomaly score-based detection model of the CRS allows individual rules to contribute to an aggregated score which triggers a blocking action once a configurable threshold is exceeded [12]. This design enables flexible tuning through configurable paranoia levels (PLs), allowing security operators to balance detection strictness with operational usability [13]. While higher PLs improve detection coverage against evasive and sophisticated attacks, they also significantly increase the false-positive rate particularly in modern web applications that employ nested parameters and dynamic queries [14,15].

False positives (FPs) represent a challenge not only for WAFs, but also for Intrusion Detection Systems (IDSs) and other security monitoring technologies [16–18]. An excessive volume of false alerts can undermine trust in security mechanisms and lead to alert fatigue, ultimately resulting in weakened protection due to rule deactivation or overly permissive configurations [19]. In WAF environments, the impact is particularly critical as false positives can directly block legitimate user requests, leading to service disruptions and the degradation of system availability [20,21].

To address these operational challenges, WAFs are often deployed in monitoring mode during the initial phases, allowing administrators to observe alerts without enforcing blocking actions while the rule set is being tuned [13]. However, the tuning process is traditionally manual and requires security analysts to inspect audit logs, identify problematic rules and parameters, and iteratively craft exception rules [13,14]. This workflow is time-consuming, error-prone, and difficult to scale especially in complex applications composed of multiple modules and endpoints with heterogeneous request structures [19]. Moreover, improper tuning practices—such as disabling entire rules instead of applying fine-grained exclusions—can significantly weaken the overall security posture [13].

A substantial body of research has explored the utilization of machine learning and data-driven approaches to enhance intrusion detection in web applications and reduce false positives. Several studies have reported supervised learning models for classifying web requests based on labeled datasets, demonstrating improvements in detection accuracy for known attack patterns [22–24]. In parallel, unsupervised approaches have been proposed to detect anomalous behaviors without relying on labeled data, making them suitable for identifying previously unseen attacks and deviations from normal traffic patterns [25].

More recently, adaptive and real-time machine learning techniques have been introduced to dynamically adjust detection thresholds and decision boundaries in WAFs, enabling more responsive protection in highly dynamic web environments [26].

While these approaches demonstrate promising results, they often require labeled datasets and introduce additional computational and operational complexity [22]. Moreover, most machine learning-based approaches focus on improving attack detection rather than addressing the operational challenge of automating the generation of secure exception rules within existing rule-based WAFs.

Beyond web application security, the automation of security policy management has been extensively studied in the context of network firewalls and IDSs [27]. Prior work on firewall rule analysis and policy mining has shown that logs and rule interactions can be systematically analyzed to identify redundancies, conflicts, and overly restrictive configurations [28]. Additionally, data mining techniques have been applied to infer higher-level security policies from low-level rules and traffic logs, enabling more maintainable and optimized firewall configurations [29,30]. These studies, while conceptually related to WAFs, demonstrate the feasibility of log-driven security policy refinement; however, they primarily focus on packet-filtering firewalls and do not account for the semantic complexity of HTTP inspection, endpoint and parameter-level analysis or the anomaly scoring mechanisms employed by modern WAFs [11–13]. These aspects are essential for the safe deployment of exception rules in WAF environments protecting web-based electronic services.

Despite the extensive literature on intrusion detection, false-positive reduction, and security policy optimization, the systematic automation of WAF exception rule generation from audit logs remains largely unexplored, to the best of our knowledge. Prior work has primarily focused on improving attack detection accuracy, tuning rule sensitivity, or applying machine learning-based models for anomaly identification, rather than formalizing a structured methodology for generating secure, endpoint-specific exception rules while preserving the core detection semantics of CRS-based WAFs. This gap highlights the need for a reproducible and implementation-independent approach that bridges WAF telemetry with controlled exception generation.

To address this challenge, this work proposes AutoEx, a framework for the automatic generation of WAF exception rules based on audit logs collected in monitoring mode. AutoEx systematically analyzes and normalizes WAF logs, aggregates false-positive events by endpoint, rule identifier, and affected request components, and generates endpoint-specific exception rules compatible with CRS-based WAF engines. The framework follows a safe-by-design philosophy, prioritizing target-level exclusions and avoiding modifications to core blocking and correlation rules.

It is important to clarify that this framework focuses exclusively on application-layer security mechanisms implemented in OWASP CRS-based WAFs. Volumetric network-layer attacks, such as traditional Denial of Service (DoS) or Distributed Denial of Service (DDoS), require distinct mitigation strategies (e.g. rate limiting or traffic filtering) and are therefore outside the scope of the proposed approach, which is centered on rule-based semantic inspection and exception generation.

The main contributions of this work can be summarized as follows:

1. The design of AutoEx, a log-driven framework for the automated generation of endpoint-specific exception rules in OWASP CRS-based WAFs.
2. A structured methodology for preprocessing and normalizing WAF audit logs to extract rule–variable–endpoint relationships relevant to false-positive mitigation.
3. A safe-by-design exception generation strategy that avoids disabling core detection and anomaly correlation rules, preserving detection effectiveness.

4. An experimental validation across multiple OWASP CRS paranoia levels (PL2–PL4), demonstrating effective false-positive reduction without compromising malicious traffic detection.

This article is organized as follows. Section 2 reviews related work on WAF security mechanisms, including machine learning-based detection approaches, log-driven analysis techniques, and automated rule adaptation strategies, concluding with the identification of the research gap addressed in this study. Section 3 presents the AutoEx framework structure detailing its architectural components, design objectives, and exception generation workflow. Section 4 describes the experimental methodology, including data acquisition, preprocessing, and rule synthesis procedures. Section 5 introduces the framework validation methodology. Section 6 reports and discusses the experimental results analyzing false-positive behavior, detection preservation across CRS paranoia levels, and processing latency. Finally, Section 7 summarizes the main conclusions derived from this work.

2. Related Works

The operational management and optimization of rule-based WAFs, particularly those built upon the OWASP CRS, have been widely studied to balance security effectiveness with operational usability. While increasing paranoia levels strengthens protection against sophisticated attacks, it also raises the likelihood of false positives in dynamic application environments.

This section reviews prior research in machine learning-based detection, log-driven analysis, and automated rule adaptation, positioning AutoEx within the existing body of work.

2.1. Machine Learning-Based WAF Detection

A dominant research direction seeks to augment or replace traditional rule-based WAF engines with machine learning (ML) and deep learning (DL) models. Durmuskaya and Bayrakli [22] demonstrate that classical ML classifiers can achieve high detection accuracy for injection-based threats. Similarly, Rohith et al. [31] propose an ML-driven WAF architecture trained on HTTP traffic datasets to improve attack classification performance.

Deep learning approaches further extend this paradigm. Muttaqin and Sudiana [32] introduce a real-time deep learning-based WAF architecture, while Chindrus and Caruntu [33] explore advanced recurrent architectures such as Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) to enhance detection capabilities. Dawadi et al. [34] reported improved web attack detection using deep learning-enabled models, and Sameh and Selim [35] proposed an Adaptive Dual-Layer Web Application Firewall (ADL-WAF) separating anomaly detection from attack classification to enhance precision.

Although these approaches achieve strong detection performance, they typically operate as black-box classifiers and introduce additional computational overhead. Moreover, they often shift the security paradigm from transparent rule evaluation to probabilistic inference. In contrast, AutoEx operates entirely within the rule-based CRS framework, preserving detection transparency and explainability while addressing the operational burden of false-positive mitigation without introducing model-training dependencies.

2.2. Log-Driven WAF Analysis

Another line of research focuses on leveraging WAF telemetry and audit logs to enhance situational awareness. Bruno et al. [36] apply process mining techniques to Modsecurity logs in order to derive behavioral models of web applications and detect deviations from expected workflows. Sun et al. [37] propose traffic parameter-based monitoring methods that analyze HTTP-session characteristics to establish baselines for normal activity.

Darmawan et al. [38] explore real-time WAF monitoring under the OWASP CRS framework, while Poat et al. [39] demonstrate visualization techniques using the Elasticsearch, Logstash and Kibana (ELK) stack to improve operational oversight of intrusion prevention tools.

These approaches strengthen monitoring, anomaly detection, and visualization capabilities. However, they primarily support administrators in diagnosing problematic rules rather than automating the derivation of corrective configurations. AutoEx extends this log-driven paradigm by transforming structured audit telemetry into actionable, endpoint-specific exception rules, reducing reliance on manual expert interpretation.

2.3. Automated Rule Adaptation

Recent research has explored automation mechanisms for adapting or repairing WAF rule sets. Appelt et al. [40] introduce one of the earliest systematic approaches to automatically repairing WAF rules based on successful SQL injection attacks. Wu et al. [41] propose WAFBooster, which employs mutation operators to generate evasive payloads and strengthen defenses against bypass techniques. Babaey and Ravindran [42] present a generative AI framework, called GenSQLi, that leverages large language models to iteratively generate SQL injection payloads and corresponding defensive rules. Similarly, Scano et al. [43] introduce ModSec-Learn, integrating machine learning to refine Modsecurity behavior based on traffic characteristics. Floris et al. [44] further investigate adversarial learning techniques to enhance WAF robustness against evasive SQL injection patterns.

While these approaches focus on strengthening rule sets against evasions or optimizing detection performance, they primarily target attack-driven enhancement cycles or scoring adjustments. In contrast, AutoEx addresses a different operational challenge: the systematic mitigation of false positives in high-paranoia CRS deployments. By leveraging granular ctl:ruleRemoveTargetById directives at the endpoint level, AutoEx ensures a safe-by-design refinement strategy that suppresses unjustified blocks only where necessary, preserving the global security posture and the original CRS detection semantics.

2.4. Synthesis and Research Gap

Across these categories, prior works either enhance detection capability through ML models, improve monitoring and visualization of WAF behavior, or automate rule strengthening against adversarial payloads. However, none formalizes a reproducible, implementation-independent methodology for transforming structured audit logs into minimal, secure exception rules tailored to specific application endpoints operating under high CRS paranoia levels.

AutoEx uniquely fills this gap by systematizing the operational tuning process itself. Rather than replacing the rule engine or modifying core detection logic, it leverages structured telemetry to automate controlled exception generation, maintaining transparency, compatibility, and upgradeability within CRS-based WAF environments.

3. AutoEx Framework Structure

AutoEx is designed as a log-driven framework for the automated generation of fine-grained exception rules in WAFs operating with the CRS. The proposed framework does not modify the internal detection logic of the CRS or core rule definitions nor does it adjust global anomaly thresholds. Instead, AutoEx operates through a controlled transformation pipeline composed of modular processing components. Its design emphasizes determinism, reproducibility, and safe-by-design exception synthesis.

The primary design objectives of AutoEx are

1. To reduce false positives generated by legitimate traffic under elevated CRS paranoia levels;
2. To preserve the global detection capability of the WAF;
3. To avoid aggressive mitigation strategies such as disabling entire rules;
4. To ensure endpoint-level precision through parameter-specific exclusions.

3.1. Modular Conception

The high-level structure and modular relationships of AutoEx are illustrated in Figure 1. The proposed framework is conceived into four modules whose interactions define a deterministic transformation pipeline from structured WAF audit logs to CRS-compatible exclusion rules.

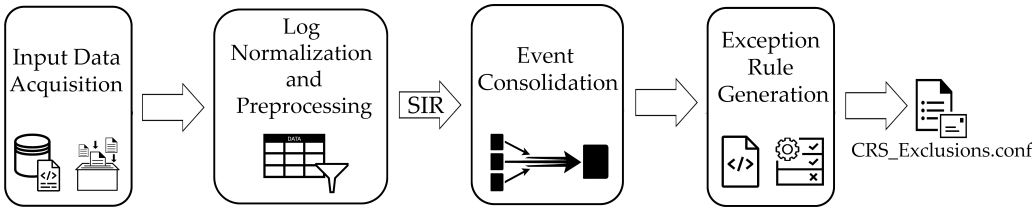


Figure 1. Structure of the AutoEx framework.

3.1.1. Input Data Acquisition Module

This module is responsible for collecting structured WAF audit logs generated during representative application interactions. It operates independently of the underlying WAF engine and provides the initial data source for the AutoEx transformation pipeline.

3.1.2. Log Normalization and Preprocessing Module

The normalization module filters raw audit logs and extracts relevant transaction-level information required for exception analysis. It transforms heterogeneous log formats into structured transaction elements while removing non-essential metadata.

AutoEx defines a canonical Structured Intermediate Representation (SIR), which acts as the internal data abstraction layer of the framework. The SIR serves as a unified data schema that decouples log acquisition and preprocessing from rule consolidation and generation. Modules responsible for data ingestion write normalized transaction elements into this representation while downstream modules operate exclusively on the standardized schema.

This abstraction layer guarantees that consolidation and rule synthesis processes remain independent of the original log source or vendor-specific formatting, thereby enhancing modularity and extensibility.

3.1.3. Event Consolidation Module

The function of this module is to transform normalized rule activation events into structured exception units. It aggregates events by endpoint and rule identifier, consolidates associated variables, and eliminates redundant activations. The output is a deterministic set of structured exception candidates that represent recurrent interference patterns between CRS rules and application endpoints.

3.1.4. Exception Rule Generation Module

The Exception Rule Generation Module synthesizes CRS-compatible exclusion directives based on the structured exception units produced by the consolidation stage. It generates endpoint-scoped SecRule and ctl:ruleRemoveTargetById statements without modifying core rule definitions.

The result is a configuration file (e.g., CRS_Exclusions.conf) that can be integrated into the WAF deployment.

The detailed operational implementation of the exception rule derivation process, including the transformation logic from structured audit logs to CRS-compatible exclusion directives, is described in Section 4.2.

It is important to emphasize that AutoEx does not employ machine learning or neural network-based classification. All transformations within the framework are deterministic and rule-based, relying exclusively on structured audit log analysis and controlled exception synthesis.

3.2. Design Principles

AutoEx is guided by the following architectural principles:

1. Deterministic transformation: The same input dataset yields the same exclusion configuration.
2. WAF engine independence: Through the SIR abstraction layer, downstream modules remain independent of the specific WAF implementation or audit log format.
3. Minimal-impact exclusions: The framework avoids global rule removal and preserves anomaly scoring mechanisms.
4. Execution under demand: AutoEx is intended to be executed per configuration snapshot and may be re-applied when application behavior changes.

4. Experimental Methodology

In this section, the methodological design applied to develop and validate a generic framework is presented. The proposed framework is designed to be adaptable to any combination of a negative security logic-based WAF with reverse proxy and any web application. The framework is specifically oriented toward automating the generation of exception rules in systems based on the OWASP CRS independently of the configured paranoia levels (PL1–PL4).

The framework is grounded in the systematic collection, analysis, and processing of blocking events generated during real web application interactions. This approach enables the identification of false positives and the creation of precise exception rules, ensuring that legitimate traffic is preserved without degrading the attack detection capabilities of the WAF.

Figure 2 illustrates the detailed operational workflow applied during the experimental validation of the AutoEx framework. While Figure 1 presents the high-level modular architecture, the workflow shown in Figure 2 describes the procedural steps followed for audit log preparation, cleansing, structured data extraction, and automated exception rule synthesis.

The process is structured into two main phases: the first phase corresponds to the preparation of input data where either real legitimate audit logs are employed or, when such data are unavailable, representative requests are constructed based on functional web application scenarios; the second phase describes the development of a script which constitutes the core component of the framework. It is specifically designed to analyze WAF audit logs and to transform complex and heterogeneous log records into structured information suitable for the automated generation of exception rules. Both phases are described below in Sections 4.1 and 4.2, respectively.

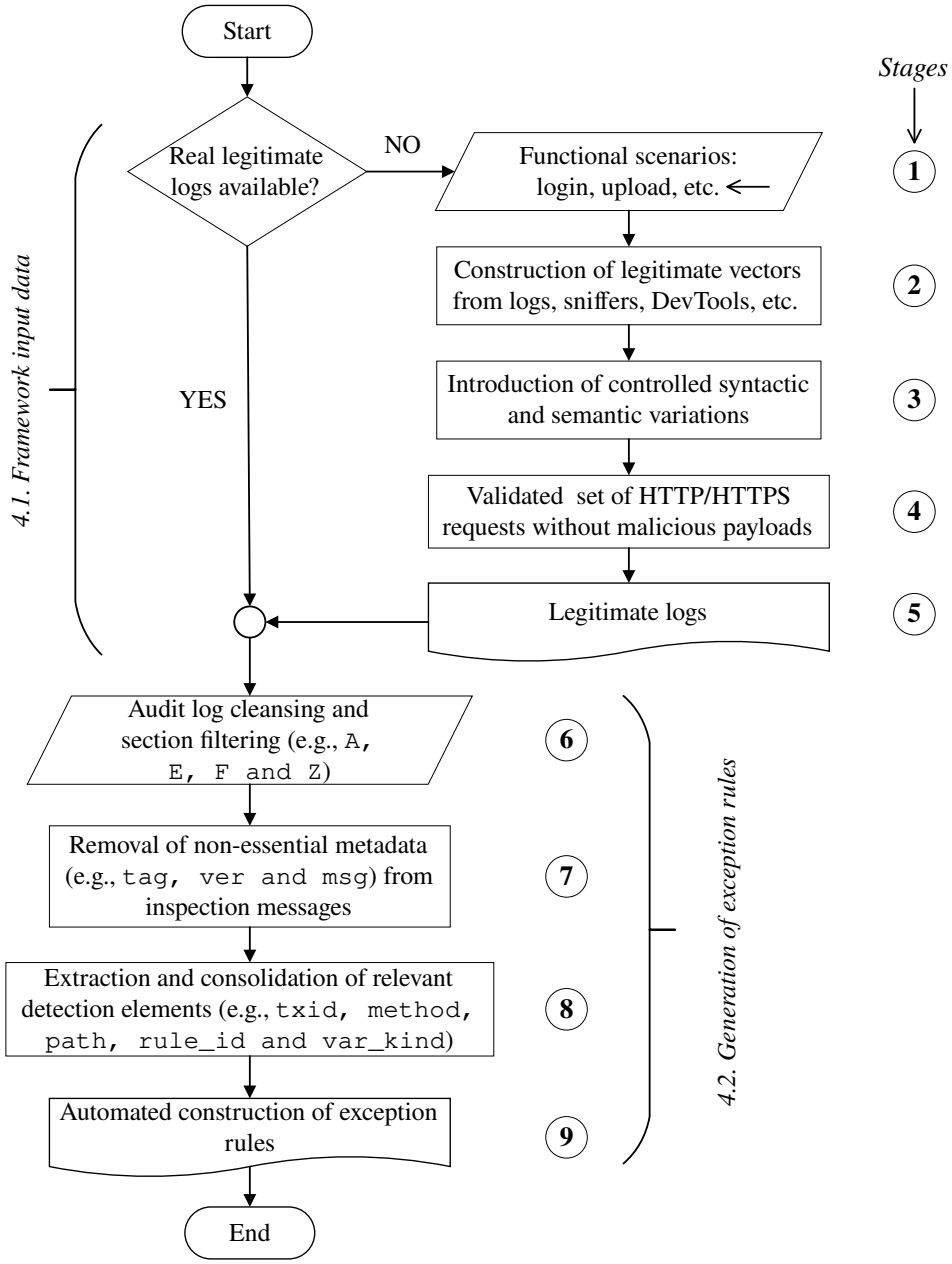


Figure 2. Operational workflow of the AutoEx exception rule generation process during experimental evaluation.

4.1. Framework Input Data

When real legitimate logs are not available, the framework activates a controlled process for constructing representative traffic of the target web application. At stage 1 in Figure 2, a set of POST requests is designed to emulate the routine usage of the web application without any prior configuration of exception rules and under one of the selected paranoia levels for this process. This stage is based on an exhaustive functional navigation of the application to identify all available input fields and interaction points where HTTP/HTTPS requests can be submitted. Based on this exploration, at stage 2, legitimate requests are constructed to ensure successful interaction with the application, covering authentication, navigation, and normal operational workflows.

Subsequently at stage 3, controlled syntactic and semantic variations of the legitimate requests are introduced in a partially random manner in order to increase the diversity and robustness of the resulting logs. At stage 4, all generated requests are verified to ensure that they do not correspond to intentional attacks or malicious payloads, preserving exclusively

legitimate traffic. Finally at stage 5, a consolidated set of legitimate logs that can be reliably utilized in the subsequent stages of the framework is generated.

Nevertheless, this initial phase may be omitted in scenarios where previously collected and sufficiently representative legitimate audit logs are available, allowing us to proceed directly to the second phase, namely data processing and analysis stages. In any scenario, these records are utilized as input for the framework.

The aim of this first phase is not to trigger deliberate attack scenarios but rather to observe WAF behavior in an unconfigured state in which blocking events recorded in the logs arise as a natural consequence of the absence of specific exception rules for the web application. Under these conditions, the observed blocking events correspond to false positives generated by legitimate traffic, thereby establishing a baseline that is essential for identifying which rules and variables interfere with normal application functionality and should be considered in the next phase.

The quality of the input data constitutes a critical factor for the effectiveness of the proposed framework. Therefore, the collected data must be sufficiently representative as greater volume and variability of legitimate traffic contribute to a more accurate identification of rules activated. For this reason, it is recommended that data be submitted across all fields and forms exposed by the application employing multiple POST request variants to capture diverse interaction patterns.

In terms of representativeness and variability, the semantic integrity of the input logs must also be ensured. The logs utilized for exception rule generation should correspond exclusively to verified legitimate traffic. If input data containing malicious payloads is incorporated into this phase, the framework could generate exclusion rules that inadvertently suppress the detection of those attack patterns. Consequently, careful validation and curation of the input dataset are essential to ensure that false-positive mitigation does not compromise the WAF's attack-detection capability.

This process may be conducted through automated scripts, manual testing procedures, dedicated testing tools or as mentioned above, traffic derived from the real operational usage of the application.

4.2. Generation of Exception Rules

The aim of this phase is to transform input data into structured information suitable for the automated generation of exception rules. In general terms, the audit logs generated by a WAF operating under negative logic and using the CRS rule set are organized into different sections identified by letters each associated with a specific stage of HTTP transaction processing, as illustrated in the example in Figure 3.

The audit log structure comprises multiple sections [45], including a mandatory initial Section A, which contains transaction identification information and timestamps; a request headers Section B, where the HTTP method, URI, and headers sent by the client are recorded; and a request-body Section C, which is present only when a request body exists and the WAF is configured to intercept it. Additionally, Modsecurity can log information related to the HTTP response, such as the final response headers (Section F) and, optionally, the intermediary response body (Section E), as long as the inspection engine has the response content capture enabled. Section H constitutes the logical conclusion of the security analysis, as it records the messages generated by the WAF engine, including the triggered rules, their identifiers, the inspected variables, severity levels, and the actions executed. Finally, section Z marks the end of each audit log entry.

```
--<transaction_id>-A--
[timestamp] [time_zone]
[source_ip] [source_port] [destination_ip] [destination_port]

--<transaction_id>-B--
POST /endpoint HTTP/1.1
Host: application_domain
User-Agent: user_agent
Content-Type: application/x-www-form-urlencoded

--<transaction_id>-C--
parameter1=value&parameter2=value&csrf_token=value

--<transaction_id>-F--
HTTP/1.1 response_status
Content-Type: response_content_type
Content-Length: response_length

--<transaction_id>-H--
[id "rule_id"]
[msg "rule_message"]
[data "Matched Data found within COLLECTION:variable"]
[severity "severity_level"]
[action "action"]

--<transaction_id>-Z--
end_of_transaction
```

Figure 3. Example of the structure of a WAF audit log.

Additional optional sections, such as I, J, or K, allow the recording of specific information related to multipart/form-data requests, uploaded files, or the complete list of matched rules, respectively. However, the effective presence of these sections depends on the WAF implementation in use, its level of compatibility with the CRS, and the applied configuration (the level of detail enabled for audit logging).

Although the nomenclature and exact formatting of log sections may vary across different WAF implementations, the proposed framework assumes the availability of information equivalent to the minimum required data necessary to enable the automated generation of exception rules.

The script implements four consecutive stages:

1. Audit log preprocessing: At stage 6 in Figure 2, the irrelevant sections such as A, E, F, and Z are removed as they contain administrative information or transaction delimiters which, although useful for traceability or forensic analysis, do not provide direct information about triggered rules or inspected variables—elements that are essential for exception generation. This early reduction in data volume helps minimize noise and optimize subsequent processing for the automated extraction of endpoints and rules focusing the analysis exclusively on sections that describe the request-submitted parameters and relevant blocking events.
2. Secondary metadata cleanup in Section H: At stage 7 in Figure 2, the descriptive elements that do not provide direct value to the exception generation process are removed. These metadata include labels (e.g. tags), version identifiers (ver), and extensive explanatory texts (msg), as well as severity levels and generic messages associated with the triggered rule. While such fields are useful for classification reporting or forensic analysis, they do not influence the determination of which rule is triggered, on which endpoint the event occurs, or which specific variable is inspected. In contrast, only information strictly related to the blocking event is retained—namely, data that allows an unambiguous association between a transaction identifier of the triggered rule, the inspected variable, and the request context (HTTP method and endpoint).

3. Systematic extraction of critical information associated with each blocking event: At stage 8 in Figure 2, data from multiple sections of the audit log are integrated. From Section B of the audit log structure, the HTTP method and the affected endpoint are extracted. This information allows us to contextualize each rule activation within a specific application functionality. In parallel, Section H of the audit log structure is analyzed to identify the triggered rule identifiers and all variables subjected to inspection by the WAF. Because a single rule may reference multiple variables within the same transaction, the procedure accounts for all detected occurrences across different message formats. This approach ensures full coverage of the inspection points involved.
   Table 1 illustrates the data frame consolidated at stage 8 in Figure 2. The extracted information is initially organized as a set of structured rows per transaction. Each row represents the relationship between an endpoint, a rule, and an inspected collection or variable. This data frame serves as an intermediate structure for analysis and exception generation. During this consolidation process, duplicates resulting from repeated log entries are removed. Key columns are normalized to ensure semantic and syntactic consistency across records. These columns include the transaction identifier (txid), HTTP method (method), accessed endpoint path (path), triggered rule identifier (rule_id), and inspected variable (var_kind).

Table 1. Normalized data frame utilized for automated exception rule generation.

Txid	Method	Path	Rule_id	Var_kind
0733051d	POST	/program-scopes/add	920230	ARGS:_TOKEN[FIELDS].
0733051d	POST	/program-scopes/add	920230	ARGS:_TOKEN[UNLOCKED].
0733051d	POST	/program-scopes/add	949110	TX:ANOMALY_SCORE.
0733051d	POST	/program-scopes/add	980130	TX:INBOUND_ANOMALY_SCORE.
66161b65	POST	/program-scopes/add	931120	ARGS:VERSION.
66161b65	POST	/program-scopes/add	931120	ARGS.
66161b65	POST	/program-scopes/add	949110	TX:ANOMALY_SCORE.
66161b65	POST	/program-scopes/add	980130	TX:INBOUND_ANOMALY_SCORE.
50ea9e4f	POST	/program-scopes/add	941100	ARGS.
50ea9e4f	POST	/program-scopes/add	941100	ARGS:VERSION.
50ea9e4f	POST	/program-scopes/add	941120	ARGS.
50ea9e4f	POST	/program-scopes/add	941120	ARGS:VERSION.
50ea9e4f	POST	/program-scopes/add	949110	TX:ANOMALY_SCORE.
50ea9e4f	POST	/program-scopes/add	980130	TX:INBOUND_ANOMALY_SCORE.
d432df62	POST	/program-scopes/add	920230	ARGS:_TOKEN[FIELDS].
d432df62	POST	/program-scopes/add	920230	ARGS:_TOKEN[UNLOCKED].
d432df62	POST	/program-scopes/add	949110	TX:ANOMALY_SCORE.
d432df62	POST	/program-scopes/add	932200	MATCHED_VAR
d432df62	POST	/program-scopes/add	980130	TX:INBOUND_ANOMALY_SCORE.

   In addition, this stage incorporates a semantic normalization process for variables. The goal is to distinguish between structural elements and request-specific content. Components that represent dynamic values or request-dependent data are removed. These include sequences containing symbols such as “=”, “%”, whitespace, numeric indices, or brackets. Such elements are unsuitable as exception targets due to their high variability. Only the actual structural variable is preserved, e.g., ARGS, REQUEST_BODY, or specific headers. This variable represents the appropriate target for an exclusion rule.
4. Automated generation of the .conf rule file: The rules generated at stage 9 in Figure 2 include SecRule and ctl:ruleRemoveTargetById directives for each endpoint and each problematic rule, as illustrated in Figure 4. All observed variables are grouped into a single exception per rule and per path. This approach preserves an ordered and consistent rule structure.

A strategy based on auxiliary SecRule directives bound to the request URI (REQUES_URI) and ctl:ruleRemoveTargetById actions is deliberately selected. This approach is chosen instead of more aggressive mechanisms such as SecRuleRemoveById, direct modification of CRS rule files, or the use of global anomaly thresholds. During exploratory testing conducted in the development of the framework, it is observed that disabling entire rules through SecRuleRemoveById effectively reduces false positives. However, this comes at the cost of suppressing the detection of relevant attack patterns on other endpoints.

```
SecRule REQUEST_URI "@beginsWith /program−scopes/add" \
 "id:1059001,phase:1,pass,log,\
  ctl:ruleRemoveTargetById=920230;ARGS,\
  ctl:ruleRemoveTargetById=931120;ARGS:VERSION,\
  ctl:ruleRemoveTargetById=931120;ARGS,\
  ctl:ruleRemoveTargetById=941100;ARGS,\
  ctl:ruleRemoveTargetById=941100;ARGS:VERSION,\
  ctl:ruleRemoveTargetById=941120;ARGS,\
  ctl:ruleRemoveTargetById=941120;ARGS:VERSION,\
  ctl:ruleRemoveTargetById=932200;MATCHED_VAR."
```

Figure 4. Example of an automatically generated Modsecurity exception rule file.

A similar effect is identified when relaxing global parameters such as tx.inbound _anomaly_score, anomaly_score, or threshold values, or when removing final correlation rules such as 949110. These actions tend to silence false positives; at the same time, they reduce sensitivity to genuinely malicious traffic. This outcome contradicts the core goal of the framework which is to preserve detection capability. The structure ctl:ruleRemoveTargetById=<rule_id>;<variable> enables the exclusion of only the specific parameters that have been shown to produce recurrent false positives on a given endpoint. Typical examples include selected fields within REQUEST_BODY or ARGS. This method does not disable the rule for the rest of the application and does not alter the overall behavior of the CRS.

The developed script consolidates all observed variables for each ⟨path, rule_id⟩ pair into a single endpoint-level exception. It explicitly excludes final anomaly aggregation rules, namely 949110 and 980130. A prior normalization step is applied to variable collections in order to remove content rather than structure. This design allows precise exclusion of only those variables responsible for false positives on a specific endpoint. As a result, exceptions are restricted to only the strictly necessary parameters, preserving CRS consistency and facilitating long-term management.

5. Framework Validation Methodology

To validate the proposed framework experimentally, the test environment consists of an Apache server configured with Modsecurity as the inspection module, as shown in Figure 5. The setup is complemented with the OWASP Core Rule Set operating at paranoia levels PL2, PL3, and PL4. The Eramba application is selected as the testing platform due to its highly interactive nature.

Figure 5 illustrates the testbed implemented to simulate HTTPS traffic flow toward the application. This configuration ensures that all requests are processed by the WAF under the defined security settings. The experimental setup provides an appropriate framework for log collection, analysis of WAF behavior, and application of the proposed framework under conditions representative of a production environment.

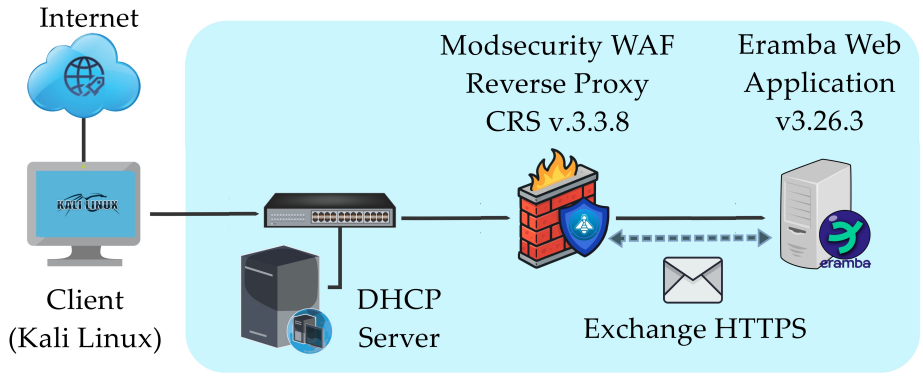


Figure 5. Schema of the experimental topology.

Paranoia Level 1 (PL1) is not considered in this study because its configuration corresponds to a basic inspection mode focused on compatibility, which exhibits a low tendency toward false positives. As a result, it does not require the application of specific exception rules [46]. In contrast, higher paranoia levels increase inspection depth and aggressiveness. These levels thus represent more suitable scenarios for evaluating the necessity and effectiveness of a framework aimed at the systematic identification and mitigation of false positives.

In this context, the WAF is configured in active blocking mode with full audit logging enabled. This configuration ensures the capture of request parameters for each POST request, as well as the triggered rules, executed actions, and blocking reasons.

Once the test environment is defined, the next step involves the controlled generation of inbound traffic. The aim of this is to obtain audit logs that are sufficiently representative to feed the proposed framework. Because the framework's operation depends on the quality and diversity of the information contained in the logs, two request sets are designed. These sets are intended to induce realistic interactions with the application under different levels of input data complexity. The traffic generation process is structured into two distinct stages.

(a) Legitimate traffic without special characters (Stage A). In the first stage, a set of POST requests is utilized to submit simple input values to Eramba fields. These inputs consist of short words without special characters. An automated script repeatedly send these requests in order to generate a sufficient volume of records. This process produces the input log data required by the proposed framework.

(b) Enriched traffic with special characters (Stage B). In the second stage, a new set of POST requests is constructed utilizing more extensive and heterogeneous inputs. The payloads include alphanumeric combinations, special characters, and increased string lengths. This design simulates the type of content that users typically enter in descriptions, comments, or free-text fields within the application. Log records are generated utilizing a large dataset of approximately 5000 requests. These requests are sent randomly and distribute across all available endpoints of the web application. The aim here is to induce the activation of a broader spectrum of OWASP CRS rules under realistic application behavior. It is important to note that these request sets are not used as part of performance evaluation or false-positive counting experiments. They are utilized exclusively as a mechanism to collect representative logs for the generation of endpoint- and paranoia-level–specific exception rules.

Once the generated rules are applied to the WAF, the framework operation is verified through 20 test cycles. Each cycle consisted of 300 randomly generated requests sent to different application endpoints.

6. Results and Discussion

Test execution is classified into two categories: malicious attack vectors as defined by the OWASP CRS, and benign inputs used to evaluate false positives. For the former, payloads sourced from widely utilized public GitHub (https://github.com/Ninja-Yubaraj/SQL-Injection-Payloads-List/tree/main, accessed on 19 October 2025) repositories are employed. The latter consists of over 500 requests incorporating special characters, generated through artificial intelligence to simulate realistic and non-deterministic variations of legitimate user content.

6.1. Detections Associated with Patterns Considered Malicious by the CRS

The results obtained in both stages of the proof of concept demonstrate that the WAF consistently preserved its detection capability against patterns considered malicious by the OWASP CRS. During the initial stage, which is characterized by legitimate traffic without special characters, and during the second stage, which introduces more complex inputs with greater character diversity, all requests containing attack patterns are correctly identified and blocked by the WAF. In no case is malicious traffic allowed to reach the protected application. This outcome confirms that the increase in traffic complexity does not negatively affect detection effectiveness. These results demonstrate that the proposed framework, although primarily aimed at identifying and mitigating false positives, does not compromise the CRS's ability to prevent real attacks. Thus, application integrity is kept throughout the entire experimental process.

6.2. False Positives Triggered by Benign Inputs

The fundamental difference between the two stages (a) and (b) described in Section 5 lies in the type of information utilized to feed the automated exception rule generation process. This distinction allows us to evaluate how the quality and diversity of input data influence the framework's ability to reduce false positives.

As shown in Figure 6, false positives are still observed across all three paranoia levels during the first stage. However, these results do not exhibit a direct relationship with the configured paranoia level. In fact, PL2 shows a higher number of unjustified blocks than PL3 or PL4. Although this behavior may appear counterintuitive from a theoretical perspective, it is not attributed to the effectiveness of the exception rules. Instead, it results from the random nature of the executed test set.

Each experiment consists of multiple repetitions that group a large number of requests. Within these repetitions, the presence or absence of special characters in specific requests directly influences the activation of CRS rules. Thus, the observed variability reflects the specific composition of the requests processed in each repetition rather than differences in restriction levels associated with the configured paranoia.

At the stage (b) in Section 5, the more enriched traffic is incorporated into the exception generation process, the more consistently the reduction in false positives is observed. This effect persists even when requests contain special characters that tend to trigger a broader set of rules. These results confirm that the framework benefits from more representative and diverse logs which enable the generation of more complete and robust exceptions. Overall, the findings indicate that the reduction in false positives is directly related to the quality and diversity of the data utilized to feed the rule generation process. It is not the result of relaxed security policies or changes in the configured paranoia level.

Table 2 reports the percentage of false positives observed before and after applying AutoEx across all paranoia levels and validation stages. Prior to applying the framework, false-positive rates reach 100% under elevated CRS paranoia levels during single-cycle evaluations. After applying AutoEx over 20 validation cycles per PL, the percentage of

false positives decreases substantially as follows: in Stage A, the mean residual false-positive rates are 45.8%, 33.4%, and 32.5% for PL2, PL3, and PL4, respectively; in Stage B, the residual false-positive rates are further reduced to mean values below 2% across all paranoia levels. These results demonstrate that the framework significantly mitigates false positives while preserving the core detection logic of the WAF.

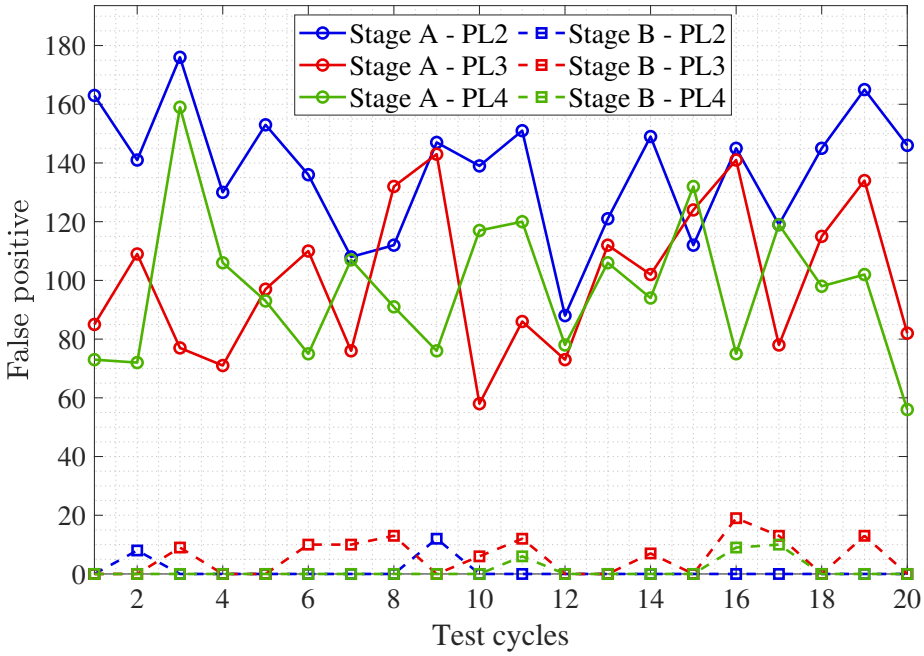


Figure 6. False positives per test cycle across paranoia levels (PL2–PL4) for the two input stages.

Table 2. Percentage of false positives before and after applying the AutoEx framework across stages and paranoia levels. The last row reads the mean value across all test cases.

Before Applying AutoEx (1 Test Cycle per PL)						**After Applying AutoEx (20 Test Cycles per PL)**					
Stage A			**Stage B**			**Stage A**			**Stage B**		
PL2	**PL3**	**PL4**	**PL2**	**PL3**	**PL4**	**PL2**	**PL3**	**PL4**	**PL2**	**PL3**	**PL4**
100%	100%	100%	100%	100%	100%	54.3%	28.3%	24.3%	0.0%	0.0%	0.0%
						47.0%	36.3%	24.0%	2.7%	0.0%	0.0%
						58.7%	25.7%	53.0%	0.0%	3.0%	0.0%
						43.3%	23.7%	35.3%	0.0%	0.0%	0.0%
						51.0%	32.3%	31.0%	0.0%	0.0%	0.0%
						45.3%	36.7%	25.0%	0.0%	3.3%	0.0%
						36.0%	25.3%	35.7%	0.0%	3.3%	0.0%
						37.3%	44.0%	30.3%	0.0%	4.3%	0.0%
						49.0%	47.7%	25.3%	4.0%	0.0%	0.0%
						46.3%	19.3%	39.0%	0.0%	2.0%	0.0%
						50.3%	28.7%	40.0%	0.0%	4.0%	2.0%
						29.3%	24.3%	26.0%	0.0%	0.0%	0.0%
						40.3%	37.3%	35.3%	0.0%	0.0%	0.0%
						49.7%	34.0%	31.3%	0.0%	2.3%	0.0%
						37.3%	41.3%	44.0%	0.0%	0.0%	0.0%
						48.3%	47.0%	25.0%	0.0%	6.3%	3.0%
						39.7%	26.0%	39.7%	0.0%	4.3%	3.3%
						48.3%	38.3%	32.7%	0.0%	0.0%	0.0%
						55.0%	44.7%	34.0%	0.0%	4.3%	0.0%
						48.7%	27.3%	18.7%	0.0%	0.0%	0.0%
100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	45.8%	33.4%	32.5%	0.3%	1.9%	0.4%

Table 3 summarizes the request processing latency measured before and after applying AutoEx. It corresponds to the P2 phase of the Modsecurity stopwatch metric, which measures the HTTP request-body processing time within the WAF engine. This metric isolates the internal inspection and rule evaluation overhead introduced by the security policies, excluding network transmission and backend processing time.

Table 3. Processing latency of the experimental framework validation.

Condition	Evaluation Setting	PL	Logs	FP	Processing Latency			
					$\bar{x}$ (ms)	σ (ms)	min (ms)	max (ms)
Before applying AutoEx	Stage A (1 test cycle per PL)	PL2	5000	5000	4.569	0.982	2.280	11.673
		PL3	5000	5000	5.158	1.289	2.487	36.019
		PL4	5000	5000	5.583	1.215	2.768	15.593
	Stage B (1 test cycle per PL)	PL2	5000	5000	5.262	1.249	2.732	13.354
		PL3	5000	5000	5.944	1.366	2.846	15.518
		PL4	5000	5000	6.408	1.719	3.061	61.274
After applying AutoEx	Stage A (20 test cycles per PL)	PL2	6000	2746	6.385	1.548	3.943	40.556
		PL3	6000	2005	8.092	1.766	4.992	17.696
		PL4	6000	1949	17.239	4.652	2.423	38.126
	Stage B (20 test cycles per PL)	PL2	6000	20	9.886	2.098	5.925	21.631
		PL3	6000	112	16.714	4.537	4.224	45.434
		PL4	6000	25	23.114	5.801	2.512	48.071

The results indicate that latency varies with both the configured paranoia level and the application of AutoEx. As expected, higher CRS paranoia levels naturally increase processing time due to additional rule evaluations. After applying AutoEx, a moderate increase in processing latency is also observed, attributable to the execution of the generated exclusion directives and the additional conditional checks introduced in the inspection flow. Considering the substantial reduction in false positives achieved, the observed latency trade-off is controlled and operationally justified, confirming that AutoEx improves usability without compromising WAF inspection stability.

6.3. Evaluation Environment Limitations

Although the experimental validation demonstrates the effectiveness of AutoEx under controlled conditions, several evaluation environment limitations should be acknowledged.

1. The validation is realized within a controlled testset consisting of a single web application (Eramba) protected by ModSecurity and the OWASP CRS. While this configuration reflects a common deployment scenario, results may vary in environments involving microservice architectures or high-concurrency production traffic conditions.
2. The datasets utilized for exception rule generation are constructed to emulate representative legitimate interactions. Although functional coverage and semantic variability are ensured, the experiments do not incorporate long-term production traffic characterized by heterogeneous user behavior.
3. The experimental validation does not include stress-testing scenarios with deployments protecting multiple heterogeneous applications under the same WAF instance.

Despite these limitations, the deterministic and input-agnostic design of the AutoEx framework, supported by the Structured Intermediate Representation (SIR), enables reproducibility across different WAF deployments and application scenarios. Because the framework operates on normalized audit log abstractions rather than engine-specific configurations, its methodological core remains transferable to alternative infrastructures, provided that equivalent structured logging capabilities are available.

7. Conclusions

This study presented a systematic framework for the identification and mitigation of false positives in WAFs based on Modsecurity and the OWASP CRS. The proposed approach relies on a structured analysis of audit logs and automated generation of endpoint-specific exception rules. The framework was designed to preserve detection capabilities while reducing the unjustified blocking of legitimate traffic.

Experimental results demonstrated that the framework effectively reduced false positives without compromising the detection of malicious patterns defined by the OWASP CRS. Across all evaluated paranoia levels (PL2–PL4), all attack payloads were consistently detected and blocked. No malicious traffic was allowed to reach the protected application. The results demonstrated that precise variable-level exclusions reduced false positives without disabling relevant attack detection mechanisms elsewhere in the application.

In addition to false-positive mitigation, the evaluation included latency measurements based on the Modsecurity stopwatch P2 processing phase, which reflects request-body inspection time. While a moderate increase in processing latency was observed after applying AutoEx—attributable to the evaluation of generated exclusion directives—this overhead remained stable and operationally bounded across paranoia levels. The observed trade-off confirms that AutoEx enhances precision without introducing disproportionate inspection cost.

The study also showed that the effectiveness of false-positive mitigation was strongly influenced by the quality and diversity of the input data utilized during exception generation. When enriched traffic containing realistic variations and special characters was incorporated, the framework produced more robust and comprehensive exception rules. In contrast, exception generation based on limited or overly simplistic inputs resulted in higher variability and a less consistent reduction in false positives. This highlighted the importance of representative log data for reliable exception tuning.

A key contribution of this work is the deliberate avoidance of aggressive mitigation strategies such as disabling complete rules, modifying CRS files, or relaxing global anomaly thresholds. Instead, the framework employed per-path SecRule directives combined with ctl:ruleRemoveTargetById actions. This design enabled precise exclusion of only the variables responsible for recurrent false positives on specific endpoints. As a result, the general behavior and coverage of the CRS were preserved across the rest of the application.

Finally, the proposed framework offers practical advantages in terms of maintainability and scalability. By consolidating all relevant variables for each ⟨path, rule_id⟩ pair into a single exception rule, the approach prevents rule uncontrolled growth and reduces configuration complexity. These characteristics make the framework suitable for deployment in real-world environments, where applications evolve continuously and false-positive management must be both precise and sustainable.

Author Contributions: Conceptualization, A.R.N.; methodology, M.C.M. and A.R.N.; validation, M.C.M. and A.R.N.; investigation, A.R.N.; writing—original draft preparation, M.C.M.; writing—review and editing, M.C.M., A.R.N. and H.B.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank the reviewers for their constructive comments that helped improve the quality of this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

WAF	Web Application Firewall
OWASP	Open Worldwide Application Security Project
PL	Paranoia Level
CRS	Core Rule Set
ICS	Industrial Control Systems
FP	False Positive
IDS	Intrusion Detection Systems
DoS	Denial of Service
DDoS	Distributed Denial of Service
ML	Machine Learning
DL	Deep Learning
GRU	Gated Recurrent Units
LSTM	Long Short-Term Memory
ADL-WAF	Adaptive Dual-Layer Web Application Firewall
ELK	Elasticsearch, Logstash and Kibana
SIR	Structured Intermediate Representation

References

1. Verizon. 2024 Data Breach Investigations Report (DBIR). Available online: https://www.verizon.com/business/resources/reports/dbir/ (accessed on 10 January 2026).
2. European Union Agency for Cybersecurity (ENISA). Threat Landscape for the Financial Sector. Available online: https://www.enisa.europa.eu/publications/enisa-threat-landscape-finance-sector (accessed on 10 January 2026).
3. World Health Organization. Global Strategy on Digital Health 2020–2025. Available online: https://www.who.int/publications/i/item/9789240020924 (accessed on 10 January 2026).
4. National Institute of Standards and Technology (NIST). Guide to Operational Technology (OT) Security; Special Publication 800-82 Revision 3. Available online: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r3.pdf (accessed on 10 January 2026).
5. OWASP Foundation. SQL Injection. Available online: https://owasp.org/www-community/attacks/SQL_Injection (accessed on 10 January 2026).
6. OWASP Foundation. Command Injection. Available online: https://owasp.org/www-community/attacks/Command_Injection (accessed on 10 January 2026).
7. OWASP Foundation. Cross-Site Scripting (XSS). Available online: https://owasp.org/www-community/attacks/xss/ (accessed on 10 January 2026).
8. Clincy, V.; Shahriar, H. Web Application Firewall: Network Security Models and Configuration. In *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*; IEEE: Washington, DC, USA, 2018; pp. 835–836.
9. OWASP Foundation. OWASP Core Rule Set Documentation. Available online: https://coreruleset.org (accessed on 10 January 2026).
10. OWASP Foundation. OWASP ModSecurity Core Rule Set Project. Available online: https://owasp.org/www-project-modsecurity-core-rule-set/ (accessed on 10 January 2026).
11. OWASP CRS Team. Paranoia Levels. OWASP Core Rule Set Documentation. Available online: https://coreruleset.org/docs/concepts/paranoia_levels/ (accessed on 10 January 2026).
12. OWASP CRS Team. Anomaly Scoring. OWASP Core Rule Set Documentation. Available online: https://coreruleset.org/docs/2-how-crs-works/2-1-anomaly_scoring/ (accessed on 10 January 2026).
13. OWASP Core Rule Set Project. False Positives and Tuning. Available online: https://coreruleset.org/docs/2-how-crs-works/2-3-false-positives-and-tuning/ (accessed on 11 January 2026).

14. Reyes Narváez, A.; Curipallo Martínez, M.; Reyes Narváez, E.; Lara, F.; Reyes Narváez, E.P.; Barba Molina, H. Evaluation Framework for False Positives in Open-Source WAFs Based on OWASP CRS Paranoia Levels: A Systematic Approach for Comparative Measurement. *Eng. Proc.* **2025**, *115*, 1. [CrossRef]
15. Anuvarshini, M.K.; Kommuri, S.S.B.; Sonti, S.S.T.; Jevitha, K.P. An Empirical Study on the Evaluation and Enhancement of OWASP CRS (Core Rule Set) in ModSecurity. *Comput. Secur.* **2026**, *160*, 104714.
16. Ho, C.-Y.; Lai, Y.-C.; Chen, I.-W.; Wang, F.-Y.; Tai, W.-H. Statistical Analysis of False Positives and False Negatives from Real Traffic with Intrusion Detection/Prevention Systems. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 1257–1271. [CrossRef]
17. Coulibaly, K. An Overview of Intrusion Detection and Prevention Systems. *arXiv* **2020**, arXiv:2004.08967. [CrossRef]
18. Gupta, N.; Jindal, V.; Bedi, P. A Survey on Intrusion Detection and Prevention Systems. *SN Comput. Sci.* **2023**, *4*, 439. [CrossRef]
19. Tariq, S.; Baruwal Chhetri, M.; Nepal, S.; Paris, C. Alert Fatigue in Security Operations Centres: Research Challenges and Opportunities. *ACM Comput. Surv.* **2025**, *57*, 224. [CrossRef]
20. Chakir, O.; Sadqi, Y.; Maleh, Y. Evaluation of Open-source Web Application Firewalls for Cyber Threat Intelligence. In *Big Data Analytics and Intelligent Systems for Cyber Threat Intelligence*; River Publishers: Gistrup, Denmark, 2023; pp. 1–14. Available online: https://www.taylorfrancis.com/chapters/edit/10.1201/9781003373384-3 (accessed on 12 January 2026).
21. Singh, J.J.; Samuel, H.; Zavarsky, P. Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. In *Proceedings of the 2018 IEEE 1st International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*; IEEE: Yogyakarta, Indonesia, 2018; pp. 1–6. Available online: https://ieeexplore.ieee.org/abstract/document/8367754/references#references (accessed on 12 January 2026).
22. Durmuskaya, M.E.; Bayrakli, S. Web Application Firewall Based on Machine Learning Models. *PeerJ Comput. Sci.* **2025**, *11*, e2975. [CrossRef]
23. Shaheed, A.; Kurdy, M.H.D.B. Web Application Firewall Using Machine Learning and Features Engineering. *Secur. Commun. Netw.* **2022**, *2022*, 5280158. [CrossRef]
24. Otero-Mosquera, J.; López-Bravo, C.; Tubío-Figueira, P.; García de la Iglesia, A.I. Improving WAF Detection Capabilities Through Machine Learning Algorithms in Open-Source Technologies. *Secur. Commun. Netw.* **2025**, *2025*, 6021296. [CrossRef]
25. Mani, K.; Shenoy, A.K.B. Machine Learning Models in Web Applications: A Comprehensive Review. *ICT Express* **2025**, *11*, 1110–1119. [CrossRef]
26. Kumar, A.; Simha, J.B.; Agarwal, R. Machine Learning-Based Web Application Firewall for Real-Time Threat Detection. In *Proceedings of the 2024 IEEE Conference on Engineering Informatics (ICEI), Melbourne, Australia, 20–28 November 2024*; IEEE: Piscataway, NJ, USA, 2024. [CrossRef]
27. Kilincer, I.F.; Ertam, F.; Sengur, A. Machine learning methods for cyber security intrusion detection: Datasets and comparative study. *Comput. Netw.* **2021**, *188*, 107840. [CrossRef]
28. Andalib, A.; Babamir, S.M. Anomaly Detection of Policies in Distributed Firewalls Using Data Log Analysis. *J. Supercomput.* **2023**, *79*, 19473–19514. [CrossRef]
29. Pyke, M.S.C.; Meng, W.; Lampe, B. Security on Top of Security: Detecting Malicious Firewall Policy Changes via K-Means Clustering. In *Machine Learning for Cyber Security*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2024; Volume 14541, pp. 145–162. [CrossRef]
30. Brighenti, D.; Marchetto, G.; Sisto, R.; Valenza, F.; Yusupov, J. Automated Firewall Configuration in Virtual Networks. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 1559–1576. [CrossRef]
31. Rohith, R.; Athief, R.; Kishore, N.; Paranthaman, R.N. Web Application Firewall Using Machine Learning. In *Proceedings of the 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 9–10 May 2024*; IEEE: New York, NY, USA , 2024; pp. 1–6. [CrossRef]
32. Muttaqin, R.Z.; Sudiana, D. Design of Realtime Web Application Firewall on Deep Learning-Based to Improve Web Application Security. *J. Penelit. Pendidik. IPA (JPPIPA)* **2024**, *10*, 11121–11129. [CrossRef]
33. Chindrus, C.; Caruntu, C.F. Improving WAF Performance with Advanced ML Models: From RNN to GRU and LSTM. In *Proceedings of the 2025 29th International Conference on System Theory, Control and Computing (ICSTCC), Cluj-Napoca, Romania, 9–11 October 2025*; IEEE: New York, NY, USA, 2025. [CrossRef]
34. Dawadi, B.R.; Adhikari, B.; Srivastava, D.K. Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks. *Sensors* **2023**, *23*, 2073. [CrossRef] [PubMed]
35. Sameh, A.; Selim, S. Adaptive Dual-Layer Web Application Firewall (ADL-WAF) Leveraging Machine Learning for Enhanced Anomaly and Threat Detection. *arXiv* **2025**, arXiv:2511.12643. [CrossRef]
36. Bruno, M.; Ibáñez, P.; Techera, T.; Calegari, D.; Betarte, G. Exploring the Application of Process Mining Techniques to Improve Web Application Security. In *Proceedings of the 2021 XLVII Latin American Computing Conference (CLEI), Cartago, Costa Rica, 25–29 October 2021*; IEEE: New York, NY, USA, 2021. [CrossRef]

37. Sun, Y.; Zhou, P.; Peng, J.; Dai, D.; Wu, Y.; Feng, J. Research on Network Attack Monitoring Based on Application HTTP Traffic Parameter Analysis. In *Proceedings of the 2025 IEEE 8th Information Technology and Mechatronics Engineering Conference (ITOEC), Chongqing, China, 14–16 March 2025*; IEEE: New York, NY, USA, 2025. [CrossRef]
38. Darmawan, I.; Nuridwan, A.; Rahmatulloh, A.; Gunawan, R.; Rizal, R. Real-time Web Application Firewall Monitoring uses the OWASP CRS Framework. In *Proceedings of the 2024 Ninth International Conference on Informatics and Computing (ICIC), Medan, Indonesia, 24–25 October 2024*; IEEE: New York, NY, USA, 2024. [CrossRef]
39. Poat, M.D.; Lauret, J.; Fedele, D. Flexible visualization of a 3rd party Intrusion Prevention (Security) tool: A use case with the ELK stack. *J. Phys. Conf. Ser.* **2023**, *2438*, 012040. [CrossRef]
40. Appelt, D.; Panichella, A.; Briand, L. Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks. In *Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017*; IEEE: New York, NY, USA, 2017; pp. 28–38. [CrossRef]
41. Wu, C.; Chen, J.; Zhu, S.; Feng, W.; He, K.; Du, R. WAFBooster: Automatic Boosting of WAF Security Against Mutated Malicious Payloads. *IEEE Trans. Dependable Secur. Comput.* **2025**, *22*, 1118–1131. [CrossRef]
42. Babaey, V.; Ravindran, A. GenSQLi: A Generative Artificial Intelligence Framework for Automatically Securing Web Application Firewalls Against Structured Query Language Injection Attacks. *Future Internet* **2025**, *17*, 8. [CrossRef]
43. Scano, C.; Floris, G.; Montaruli, B.; Demetrio, L.; Valenza, A.; Compagna, L.; Ariu, D.; Piras, L.; Balzarotti, D.; Biggio, B. ModSec-Learn: Boosting ModSecurity with Machine Learning. In *Proceedings of the 21st International Conference on Distributed Computing and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2025; pp. 23–33. Available online: https://link.springer.com/chapter/10.1007/978-3-031-76459-2_3 (accessed on 27 February 2026).
44. Floris, G.; Scano, C.; Montaruli, B.; Demetrio, L.; Valenza, A.; Compagna, L. ModSec-AdvLearn: Countering Adversarial SQL Injections With Robust Machine Learning. *IEEE Trans. Inf. Forensics Secur.* **2025**, *20*, 6693–6705. [CrossRef]
45. OWASP ModSecurity Project. ModSecurity Reference Manual (v2.x): Audit Log. Available online: https://github.com/owasp-modsecurity/ModSecurity/wiki/Reference-Manual-(v2.x)#audit-log (accessed on 13 November 2025).
46. Curipallo Martínez, M.; Guevara-Vega, A.; Reyes Narváez, A.; Raura, G.; Molina, H.; Barba Molina, H. Web Application Protection Optimization Through Coraza WAF: Performance Assessment Against OWASP Risks in Reverse Proxy Configurations. *Eng. Proc.* **2025**, *115*, 17. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.