Contents lists available at IdNNS
Indonesian Journal of Cybersecurity and Emerging Risks
Volume 1 ; Issue 1
journal homepage: https://journal.idnns.org/index.php/ijcyber

# Enhancing Web Defense Through Machine Learning and Active Response Mechanism Integration in Web Application Firewalls

Antonio Varialle[a,*], Gerard Borg[b] and Agung Prasetiawan[c]

[a]Blue5 Group, Italy
[b]Departement of Engineering ANU, Australia
[c]School of Computer Science, Nusa Putra University, Sukabumi, Indonesia

ARTICLE INFO

*Keywords*:
Web Application Security
Machine Learning
Web Application Firewall
WAF
Dynamic Security Framework
Zero-day Attack
Active Response

ABSTRACT

Web applications are increasingly exposed to attacks such as SQL injection, cross-site scripting, path traversal, and request-forgery attempts. Conventional Web Application Firewalls often rely on static rules and signatures, which can be less effective against modified, obfuscated, or previously unseen payloads. This study proposes a dynamic Web Application Firewall framework that integrates machine-learning classification, attack-keyword filtering, regex-based matching, anomaly handling, an attack-pattern library, and active response. Several models were evaluated, including Support Vector Machine, Random Forest, Logistic Regression, Convolutional Neural Network, and Recurrent Neural Network. Random Forest achieved the highest accuracy in the model comparison with 0.9954, followed closely by CNN and RNN. In a separate deployed-model test, the selected model achieved 85.1% accuracy, 85.1% precision, 100% recall, and 91.9% F1-score, showing strong attack coverage but also false positives. The attack-pattern library achieved 93% accuracy, 100% precision, 93% recall, and 96% F1-score. These results indicate that the proposed hybrid framework is promising for adaptive WAF detection and response, but broader benign traffic, comparison with baseline WAF configurations, and production-scale testing are still required.

## 1. Introduction

Web applications have become an essential part of modern digital infrastructure because they provide fast, flexible, and scalable access to services, information, and online transactions. However, the increasing dependence on web-based systems has also expanded the attack surface for cyber threats. Attacks such as SQL injection, cross-site scripting (XSS), distributed denial-of-service (DDoS), path traversal, and payload injection continue to threaten the confidentiality, integrity, and availability of web applications (Dawadi et al., 2023; Riera et al., 2022; Surendhar et al., 2023).

A Web Application Firewall (WAF) is commonly used as a security layer to filter, monitor, and block malicious HTTP traffic before it reaches the protected application. Traditional WAF solutions generally depend on static rules and signature-based detection mechanisms. Although these approaches are effective for known attack patterns, they often struggle to detect new, modified, obfuscated, or unknown attacks because the incoming request may not match existing signatures (Applebaum et al., 2021; Sepczuk, 2022). This limitation becomes more critical when attackers develop zero-day payloads or use evasion techniques that bypass predictable rule-based defenses (Sepczuk, 2022; Calvo and Beltrán, 2022).

Recent developments in artificial intelligence and machine learning provide opportunities to improve WAF adaptability. Machine learning can learn from historical traffic data, extract attack-indicating features, and classify web requests as benign or malicious. Previous studies have shown that machine learning and deep learning techniques can improve web-attack detection through feature engineering, HTTP traffic analysis, and neural-network-based classification (Tiwari et al., 2023; Dawadi et al., 2023; Shaheed and Kurdy, 2022). In addition, CNN-based and smart WAF designs show that deep learning can be applied directly to WAF detection tasks (Wang et al., 2022; Jemal et al., 2022).

However, detection alone is not sufficient if the system cannot respond quickly and continuously adapt to new threats. Existing research has explored mitigation, attacker profiling, deception, and centralized response mechanisms to improve protection after malicious activity is detected (Shahid et al., 2022b,a; Manaseer and Al Hwaitat, 2018). Active response and adaptive anomaly detection are therefore important for WAF systems because they allow suspicious requests to be

*Corresponding author
av@blu5labs.eu (A. Varialle); Gerard.Borg@anu.edu.au (G. Borg); agung.prasetiawan@nusaputra.ac.id (A. Prasetiawan)
ORCID(s):

blocked, isolated, logged, and used as learning material for future detection updates (Xuan et al., 2020; Calvo and Beltrán, 2022).

This research proposes an enhanced WAF framework that integrates machine learning with an active response mechanism. The proposed system is designed not only to detect malicious requests but also to respond to threats by blocking known attack patterns, isolating suspicious queries, recording anomalies, and updating an attack-pattern library. By combining known attack detection, anomaly detection, regex-based filtering, attack keyword analysis, and adaptive learning, the proposed framework is intended to support stronger detection and response against selected existing and emerging web application threats (Calvo and Beltrán, 2022; Xuan et al., 2020; Mohammadi Rouzbahani et al., 2020).

The main objective of this study is to improve web application defense by developing a dynamic WAF concept that is capable of learning from attack data and responding to malicious activities in real time. The proposed approach is evaluated using several machine learning models, including Support Vector Machine (SVM), Random Forest, Logistic Regression, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). The use of multiple models supports comparative evaluation and helps identify the most effective model for deployment in the WAF architecture (Dawadi et al., 2023; Wang et al., 2022; Shaheed and Kurdy, 2022).

The contribution of this research is threefold. First, it presents a machine learning-based WAF approach for detecting malicious web requests. Second, it integrates an active response mechanism to support real-time mitigation and anomaly isolation. Third, it introduces an adaptive attack-pattern library that can support future attack prediction and improve WAF resilience against zero-day threats.

## 2. Related Work

Web application security has become an important research area because modern web systems are frequently targeted by attacks that exploit weaknesses in application logic, user input handling, and HTTP request processing. Traditional Web Application Firewalls (WAFs) commonly rely on predefined signatures and rule-based filtering to identify malicious requests. These mechanisms remain useful for detecting known attacks, but they have limitations when facing obfuscated payloads, new attack variants, and zero-day threats (Applebaum et al., 2021; Sepczuk, 2022).

Recent studies have proposed machine learning as a way to improve web application defense. Tiwari et al. investigated the integration of artificial intelligence and machine learning into web application development and defense, showing that AI/ML can support vulnerability prediction and improve security-oriented web development (Tiwari et al., 2023). Shaheed and Kurdy proposed a WAF based on machine learning and feature engineering, demonstrating that extracted request features can be used to classify malicious and benign web traffic (Shaheed and Kurdy, 2022). Surendhar et al. also applied machine learning to payload injection detection in firewall traffic, supporting the idea that payload-based attacks can be detected through learned traffic patterns rather than only static signatures (Surendhar et al., 2023).

Deep learning has also been explored for WAF and web-attack detection. Dawadi et al. developed a deep-learning-enabled WAF architecture for detecting attacks such as DDoS, SQL injection, and cross-site scripting by comparing normal HTTP traffic and attack traffic (Dawadi et al., 2023). Wang et al. designed a WAF system using convolutional neural networks and deep learning, showing the relevance of CNN-based architectures for web request classification (Wang et al., 2022). Similarly, Jemal et al. proposed SWAF, a smart WAF based on convolutional neural networks, which further supports the use of deep learning for intelligent web-attack detection (Jemal et al., 2022).

Several studies focus not only on detection but also on mitigation, attacker profiling, and adaptive defense. Shahid et al. proposed an enhanced deep-learning framework for web attack detection, mitigation, and attacker profiling, showing that detection systems can be connected with response mechanisms to reduce the impact of attacks (Shahid et al., 2022b). In another study, Shahid et al. introduced a personalized deception system assisted by deep learning, where malicious HTTP requests are redirected to deception environments and attacker profiles are used to support future defense (Shahid et al., 2022a). Manaseer and Al Hwaitat proposed a centralized WAF security system that distributes attack information across connected clients so that multiple hosts can react to the same attacker in real time (Manaseer and Al Hwaitat, 2018).

Adaptive WAF research addresses the weakness of fixed rule configuration. Calvo and Beltrán proposed an adaptive WAF that changes its configuration according to context and risk indicators, because a WAF cannot be configured once and remain effective against changing attack conditions (Calvo and Beltrán, 2022). Xuan et al. proposed an adaptive anomaly request detection framework based on dynamic web application profiles, where abnormal requests can be detected and attack data can be semi-automatically updated during web application access (Xuan et al., 2020). These studies show that dynamic profiling, anomaly detection, and continuous updates are important for improving WAF resilience.

Dataset quality and attack classification are also important in machine-learning-based WAF research. Riera et al. introduced a multi-label dataset for web attack classification based on CAPEC and evaluated machine learning methods for classifying different web attack types (Riera et al., 2022). Feature-driven analysis is also relevant for cybersecurity data sources; Tundis et al. showed that features can be used to automate the assessment of open-source cyber threat sources, which is useful for future threat intelligence integration and model updating (Tundis et al., 2022).

For SQL injection and pattern-based detection, previous research has used both machine learning and string-matching approaches. Kar et al. proposed SQLiGoT, which detects SQL injection attacks using graph-of-token representation

and SVM classification (Kar et al., 2016). In addition, Aho-Corasick-based multiple-pattern matching is relevant to efficient keyword and signature matching because it can match many attack patterns in a single process (Sun and Li, 2012).

Firewall optimization is another related area because security filtering may introduce processing overhead. Coscia et al. proposed a two-stage algorithm to optimize firewall rule ordering, showing that rule placement can affect packet classification latency and overall firewall performance (Coscia et al., 2023). This is relevant to WAF design because a practical WAF must balance detection accuracy, response capability, and processing efficiency.

Based on these studies, existing WAF research has progressed from static signatures toward machine learning, deep learning, adaptive configuration, anomaly detection, and active response. However, many approaches still focus mainly on detection or on a single defensive component. This study therefore proposes an integrated framework that combines machine-learning-based detection, regex and attack keyword analysis, anomaly logging, active response, and adaptive attack-pattern library updates to improve WAF protection against both known and emerging threats.

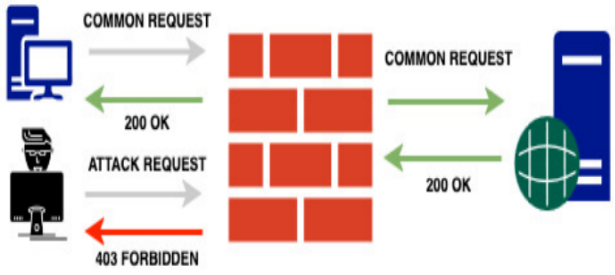


Figure 1: WAF process

## 3. Methodology

### 3.1. Method Used

This research uses a mixed quantitative and qualitative methodology to develop and evaluate a machine-learning-based Web Application Firewall (WAF) with an active response mechanism. The quantitative method is used to measure the performance of machine learning models through accuracy, precision, recall, and F1-score. These metrics help evaluate how effectively the proposed WAF detects malicious web requests and reduces false positives and false negatives (Chicco et al., 2021; Shaheed and Kurdy, 2022).

The qualitative method is used to analyze attack behavior, anomaly handling, and the impact of active response on web application security. This analysis is important because WAF protection does not only depend on detecting known attacks, but also on understanding suspicious request patterns and adapting to new or unknown threats (Sepczuk, 2022; Xuan et al., 2020).

The proposed method integrates machine learning classification, attack keyword analysis, regular expression matching, anomaly logging, and active response. Incoming HTTP requests are first inspected by the WAF before reaching the protected web application. Benign requests are forwarded to the application, while malicious or suspicious requests are blocked, logged, isolated, or stored as anomaly data for future learning. This process supports adaptive WAF behavior because the system can update its detection knowledge when new attack patterns are discovered (Calvo and Beltrán, 2022; Mohammadi Rouzbahani et al., 2020).

As shown in Figure 2, the WAF acts as an intermediate security layer between users and the web application. Common requests are allowed to continue to the server, while attack requests are rejected or processed through the detection and response mechanism. This design extends the traditional WAF concept by combining static filtering with machine learning and adaptive response, making it more suitable for handling modified, obfuscated, and zero-day attack patterns (Applebaum et al., 2021; Sepczuk, 2022; Shahid et al., 2022b).

### 3.2. Evaluation Variables

The evaluation uses the confusion-matrix variables true positive ($TP$), true negative ($TN$), false positive ($FP$), and false negative ($FN$). These variables are used to calculate accuracy, precision, recall, F1-score, and Matthews Correlation Coefficient (MCC), as shown in Equations (2)–(1). Accuracy measures overall correctness, precision measures the reliability of attack predictions, recall measures how many actual attacks are detected, and F1-score balances precision and recall. MCC is included because it considers all four confusion-matrix components and is more informative when binary classification data are imbalanced (Chicco et al., 2021). In a WAF context, recall is particularly important because false negatives allow malicious requests to reach the protected application, while precision is also important because false positives may block legitimate users.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{1}$$

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

$$\text{Precision} = \frac{TP}{TP+FP} \tag{3}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{4}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

These metrics are interpreted together with the confusion matrix rather than in isolation. This is important because the deployed-model test and library test have limited true-negative representation. Therefore, the discussion emphasizes recall, precision, false positives, and false negatives, while MCC is included as a recommended metric for future evaluation with a more balanced test set.

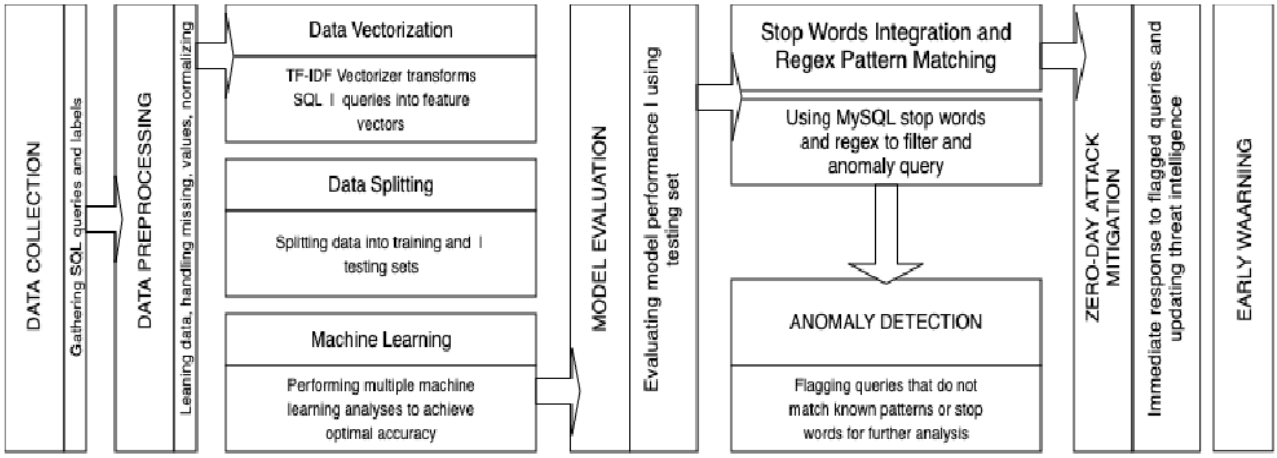


**Figure 2:** Processing step

### 3.3. Attack Keyword-Based Filtering

Attack-keyword filtering is used as an initial screening stage before regex matching and machine-learning classification. The keyword list was constructed from three sources: web-attack categories represented in the collected datasets, common payload structures discussed in prior SQL injection and WAF studies, and attack classes aligned with OWASP-style web application risks such as injection, cross-site scripting, broken access/path manipulation, and request-forgery patterns (Kar et al., 2016; Shaheed and Kurdy, 2022; Riera et al., 2022). The purpose of the list is not to classify attacks by itself, but to highlight request fragments that require deeper inspection.

As shown in Figure 3, incoming traffic is first inspected for attack-related terms. Requests containing suspicious keywords are not automatically blocked only because a keyword is present, since several terms may also appear in legitimate requests. Instead, the keyword result is combined with regex matching, anomaly checking, and machine-learning classification. This reduces unnecessary processing while also reducing the risk of blocking benign requests that contain isolated keywords.

Table 1: Attack keyword categories and intended detection role

| Category | Example indicators | Detection role |
|---|---|---|
| SQL injection | `SELECT`, `UNION`, `OR`, `DROP`, `SLEEP`, `INFORMATION_SCHEMA` | Flags database-oriented payloads and suspicious query structures for regex and ML validation. |
| Cross-site scripting | `script`, `onerror`, `alert`, `javascript`, `document.cookie`, `eval` | Identifies script execution attempts and client-side payload fragments in user-controlled input. |
| Path traversal | `etc/passwd`, `../`, `boot.ini`, `/bin/bash`, `/proc/self/environ`, `system32` | Highlights attempts to access restricted files, operating-system paths, or server-side resources. |
| Request forgery tokens | `csrf_token`, `X-CSRF-Token`, `authenticity_token`, `_csrf`, `_token` | Helps identify suspicious manipulation or absence of expected request-validation tokens. |

Table 1 summarizes representative keyword groups used in the filtering stage. The table is intentionally limited to representative indicators because the operational pattern library is expected to evolve as new confirmed anomalies are discovered. Confirmed malicious patterns are added to the attack-pattern library after validation, which allows the WAF to adapt without depending only on a fixed static rule list (Xuan et al., 2020; Calvo and Beltrán, 2022).

### 3.4. Regex-Based Detection

Regular expressions, commonly known as regex, are sequences of characters that define a search pattern. Regex can be used for string matching and manipulation. In the context of injection detection, regex is utilized to identify patterns that are characteristic of injection attacks. This method complements attack keyword detection because regex can describe suspicious query structures, not only individual words or tokens (Sun and Li. 2012: Kar et al.. 2016).

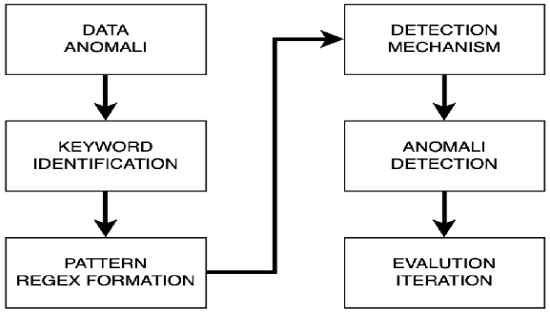


Figure 4: Method keywords-based detection

The keyword-based detection process first checks whether a query contains common attack-related terms. If suspicious keywords are found, the query can be forwarded to the next detection stage for deeper analysis. This process is illustrated in Figure 4.

Regex patterns are predefined rules that match specific sequences of characters. These patterns are designed to identify common injection techniques. Some common patterns used

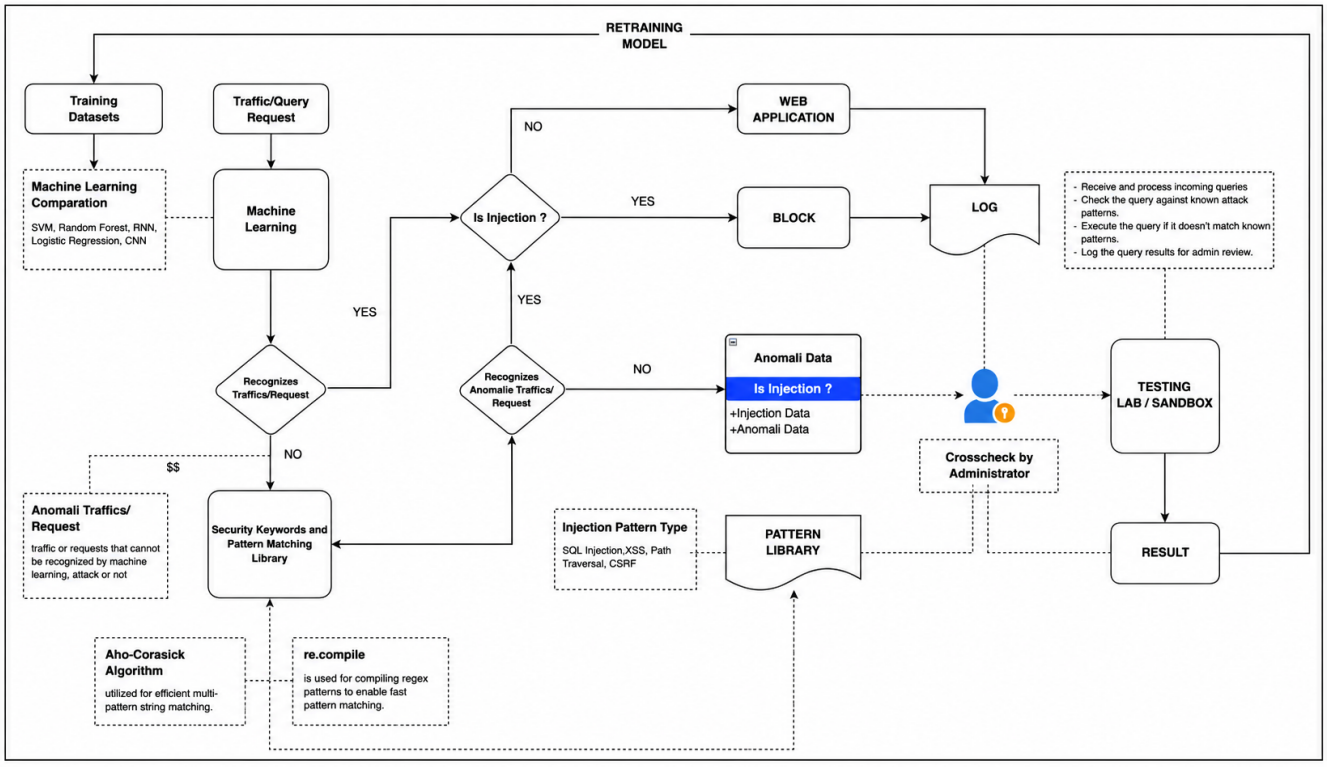


**Figure 3:** Attack keyword-based filtering process

to detect injection include the presence of SQL keywords such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` followed by suspicious constructs. Other examples include `OR 1=1`, comment markers such as `--`, and destructive statements such as `; DROP TABLE`, which are commonly associated with SQL injection attacks. Examples of regex patterns for injection detection are shown in Figure 5.

- `` `r"(?i)(\%27)|(\')|(\-\-)|(\%23)|(#)"` ``: Detects single quotes, comments (`` `--` `` or `` `#` ``), or encoded forms of single quotes.
- `` `r"(?i)(\%27)|(\')|(\-\-)|(\%23)|(#)|((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))"` ``: Detects SQL meta-characters and typical SQL injection patterns like `` `OR 1=1` ``.
- `` `r"(?i)select.+from"` ``: Detects `` `SELECT` `` statements followed by `` `FROM` ``.

Figure 5: Some common regex patterns

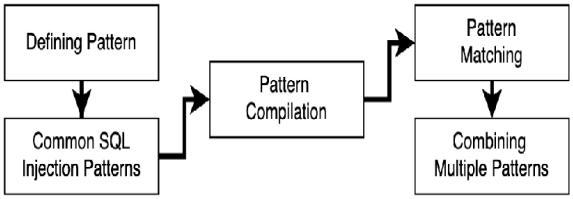


Figure 6: How regex work

Regex patterns are compiled to create regex objects, making the matching process faster and more efficient. Incoming SQL queries are matched against the compiled regex patterns. If a query matches one or more patterns, it is flagged as a potential SQL injection attack. Multiple regex patterns can also be combined to cover a wider range of SQL injection techniques and payload variations. The regex matching process is shown in Figure 6.

In advanced SQL injection detection systems, regex patterns can be combined with attack keywords to improve detection accuracy. The process begins by checking whether a query contains known attack keywords. After that, regex matching is applied to determine whether the query structure follows a suspicious injection pattern. If both keyword and regex indicators are detected, the request can be classified as malicious or forwarded to the machine learning model for additional validation (Shaheed and Kurdy, 2022; Surendhar et al., 2023).

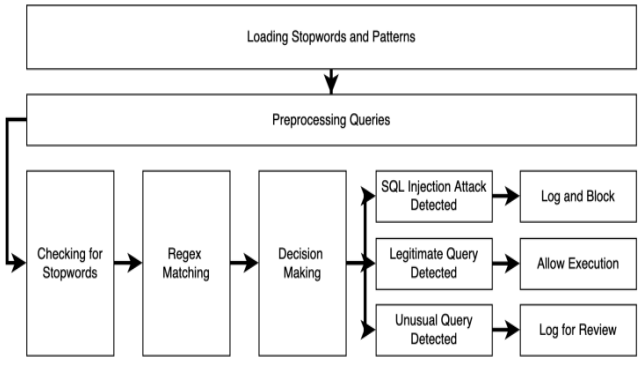


Figure 7: Regex and attack keyword combination detection

Figure 7 describes the combined decision flow used by the proposed detector. First, the request is inspected for attack-related keywords. Second, requests containing suspicious

indicators are matched against compiled regex patterns to determine whether the structure resembles known injection payloads. Third, requests that remain uncertain are forwarded to the machine-learning classifier or stored as anomaly data. This design prevents the WAF from relying on a single detection signal and allows the system to combine fast pattern matching with adaptive classification.

The combination of regex and attack keyword analysis strengthens the proposed WAF because it detects both known attack terms and suspicious query structures. This hybrid approach also reduces false negatives by identifying modified injection payloads that may not be detected by keyword matching alone. When combined with anomaly detection and machine learning, the system can provide a more adaptive defense against both known and emerging injection attacks (Xuan et al., 2020; Mohammadi Rouzbahani et al., 2020).

### 3.5. Data Collecting

The data used in this research was collected from several sources to represent both normal web traffic and malicious web application attacks. The dataset consists of synthetic attack data, historical attack data from Indonesian military web applications, Kaggle datasets, and CSIC web traffic datasets. These data sources were selected to provide a diverse set of request patterns, including legitimate requests and several common attack types. Using multiple sources is important because machine-learning-based WAF models require varied traffic patterns to improve generalization and reduce bias toward a single attack scenario (Dawadi et al., 2023; Riera et al., 2022).

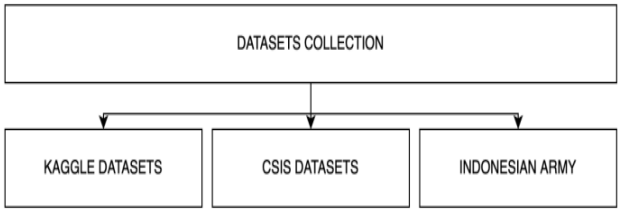


Figure 8: Dataset collection

As shown in Figure 8, the dataset collection process combines several data sources before they are processed for model training and testing. Synthetic data is used to simulate different attack scenarios, while public datasets such as Kaggle and CSIC provide examples of common web request behavior and known attack signatures. Historical attack data is also used to make the dataset more relevant to real-world web application security cases.

The collected dataset contains several types of web application attacks, including SQL injection (SQLi), cross-site scripting (XSS), path traversal, and cross-site request forgery (CSRF). These attack categories are commonly used in web attack classification and WAF research because they represent major threats against web applications and input-handling mechanisms (Riera et al., 2022; Shaheed and Kurdy, 2022).

Figure 9 shows the distribution of attack types in the dataset. SQL injection has the highest number of instances, followed by XSS, path traversal, and CSRF. These attack categories were selected because they represent common input-driven web application threats and can be related to OWASP-style risk areas. SQL injection is directly related to injection risk; XSS represents client-side script injection; path traversal is related to broken access control and unauthorized file access; and CSRF represents request-forgery behavior that can abuse authenticated user actions.

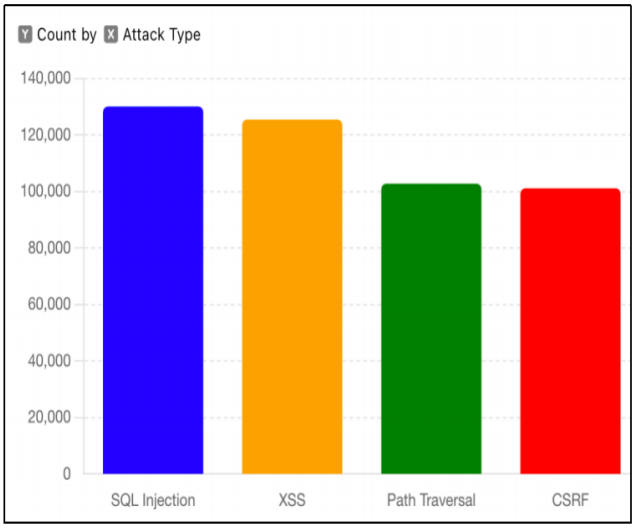


Figure 9: Attack type dataset

Table 2 clarifies why these attack classes are relevant for WAF evaluation. Nevertheless, the dataset does not cover every OWASP Top Ten risk. Therefore, the result should be interpreted as evidence for selected request-level attacks rather than complete coverage of all web application security risks.

Table 2: Relationship between dataset attack types and OWASP-style web risks

| Dataset attack type | Related OWASP-style risk area | Reason for inclusion |
|---|---|---|
| SQL injection | Injection | Represents database-oriented payloads that attempt to manipulate backend queries. |
| Cross-site scripting (XSS) | Injection / client-side script execution | Represents malicious script payloads submitted through user-controlled input. |
| Path traversal | Broken access control / unauthorized file access | Represents attempts to access restricted files or directories through crafted paths. |
| Cross-site request forgery (CSRF) | Request forgery / abuse of authenticated actions | Represents manipulation of trusted user sessions or request-validation mechanisms. |

Before being used for machine learning, the collected data is prepared for the next stage of data pre-processing. This includes cleaning duplicate records, checking missing values, organizing attack labels, and ensuring that the request data can be transformed into numerical features. Proper data collection and preparation are essential because the quality of training data directly affects the detection accuracy and reliability of the WAF model (Dawadi et al., 2023; Shaheed and Kurdy, 2022).

### 3.6. Data Pre-Processing

Data pre-processing is an important stage in preparing raw web request data before it is used for machine learning model training and testing. Raw traffic data often contains duplicate records, missing values, inconsistent formats, unnecessary symbols, and unstructured request payloads. If these problems are not handled properly, the model may learn irrelevant patterns and produce inaccurate detection results. Therefore, pre-processing is required to clean, normalize, and transform the collected data into a structured format suitable for web attack classification (Dawadi et al., 2023; Shaheed and Kurdy, 2022).

As shown in Figure 10, the data pre-processing stage begins with raw data collection, followed by tokenization, feature extraction, and vectorization. Tokenization is used to split web requests or SQL queries into smaller units such as words, symbols, or query tokens. This step helps the system identify important parts of the request, including SQL commands, parameters, operators, and suspicious payload fragments. Token-based representation is useful in SQL injection detection because malicious queries often contain abnormal token structures compared with legitimate queries (Kar et al., 2016).

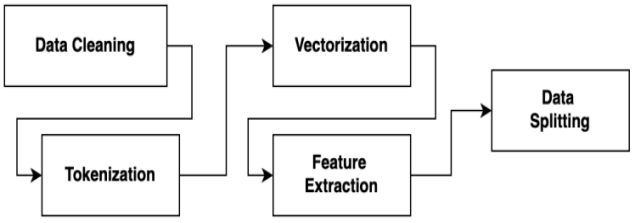


Figure 10: Data processing

After tokenization, feature extraction is performed to identify patterns that may indicate web attacks. These features may include attack keywords, regex-matched patterns, request length, suspicious characters, encoded payloads, and the presence of SQL or script-related terms. Feature engineering is essential because machine learning models cannot directly understand raw text; they require numerical or structured features to perform classification (Shaheed and Kurdy, 2022; Riera et al., 2022).

The next step is vectorization, where textual request data is transformed into numerical values. Techniques such as TF-IDF can be used to convert tokens into feature vectors so that machine learning algorithms can process them. After vectorization, the dataset is divided into training and testing sets. The training set is used to build the model, while the testing set is used to evaluate how well the model detects unseen web requests. This separation is necessary to measure the generalization ability of the proposed WAF model (Dawadi et al., 2023).

In this research, data pre-processing supports the overall WAF detection pipeline by reducing noise, improving feature quality, and making the dataset ready for model comparison. A well-prepared dataset helps improve accuracy, precision, recall, and F1-score because the model can focus on relevant attack indicators rather than irrelevant or inconsistent data.

### 3.7. Data Processing

After data pre-processing, the cleaned and vectorized dataset is processed using several machine learning and deep learning models. This research compares Support Vector Machine (SVM), Random Forest, Logistic Regression, Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). The use of multiple models allows the study to identify which algorithm provides the most suitable performance for malicious web request detection (Dawadi et al., 2023; Shaheed and Kurdy, 2022; Wang et al., 2022).

Table 3: SVM parameters

| Parameter | Description | Value |
|---|---|---|
| `kernel` | Kernel type to be used in the algorithm | `'linear', 'poly', 'rbf', 'sigmoid'` |
| `C` | Regularization parameter | `1.0` |
| `gamma` | Kernel coefficient for 'rbf', 'poly', and 'sigmoid' | `'scale'` |
| `degree` | Degree of the polynomial kernel function ('poly') | `3` |
| `coef0` | Independent term in kernel function; significant in 'poly' and 'sigmoid' | `0.0` |

SVM is used because it is effective for classification problems with high-dimensional feature spaces, including text-based and token-based representations of web requests. In SQL injection detection, SVM has been used with token-based features to distinguish malicious and benign query structures (Kar et al., 2016). The SVM parameters used in this research are shown in Table 3.

Random Forest is used as an ensemble learning model that combines multiple decision trees to improve classification stability. This model is useful for structured feature-based detection because it can handle nonlinear relationships among extracted request features. The Random Forest parameters used in this research are shown in Table 4.

Table 4: Random Forest parameters

| Parameter | Description | Value |
|---|---|---|
| `n_estimators` | Number of trees in the forest | `100` |
| `max_depth` | Maximum depth of the tree | `None` |
| `min_samples_split` | Minimum number of samples required to split an internal node | `2` |
| `min_samples_leaf` | Minimum number of samples required to be at a leaf node | `1` |
| `max_features` | Number of features to consider when looking for the best split | `'auto'` |

Logistic Regression is used as a baseline classification model because it is simple, interpretable, and effective for binary classification tasks. It helps compare whether more complex models provide better detection performance than a linear classifier. The Logistic Regression parameters used in this research are shown in Table 5.

Table 5: Logistic Regression parameters

| Parameter | Description | Value |
|---|---|---|
| C | Inverse of regularization strength | 1.0 |
| solver | Algorithm to use in the optimization problem | 'lbfgs' |
| max_iter | Maximum number of iterations taken for the solvers to converge | 100 |
| penalty | Used to specify the norm of the penalty | 'l2' |
| tol | Tolerance for stopping criteria | 1e-4 |

CNN is evaluated because convolutional layers can learn local patterns from vectorized request data and have been applied in WAF research for web attack detection (Wang et al., 2022; Jemal et al., 2022). The CNN parameters used in this research are shown in Table 6.

Table 6: CNN parameters

| Parameter | Description | Value |
|---|---|---|
| filters | Number of output filters in the convolution | 64 |
| kernel_size | Length of the convolution window | (3, 3) |
| strides | Stride length of the convolution | (1, 1) |
| padding | Padding method (valid or same) | 'same' |
| activation | Activation function to use | 'relu' |
| pool_size | Downscale factor in the pooling operation | (2, 2) |
| dropout | Fraction of the input units to drop for regularization | 0.5 |
| dense_units | Number of units in the dense layer | 128 |
| epochs | Number of epochs to train the model | 10 |
| batch_size | Number of samples per gradient update | 32 |

RNN is evaluated because request payloads and query strings can be treated as sequences of tokens. This makes RNN suitable for learning sequential dependencies in malicious request patterns. The RNN parameters used in this research are shown in Table 7.

Table 7: RNN parameters

| Parameter | Description | Value |
|---|---|---|
| units | Number of units in the RNN layer | 50 |
| dropout | Fraction of the input units to drop for regularization | 0.2 |
| recurrent_dropout | Fraction of the recurrent units to drop for regularization | 0.2 |
| activation | Activation function to use | 'tanh' |
| batch_size | Number of samples per gradient update | 32 |
| epochs | Number of epochs to train the model | 10 |

After all models are trained, their performance is compared using the evaluation metrics defined in the variable operations section. The best-performing model is then selected for integration into the WAF detection workflow. This comparison helps ensure that the final WAF model is selected based on measurable detection performance rather than assumption.

### 3.8. Research Environment and Metrics

The proposed framework was evaluated in a controlled environment consisting of a client, the WAF component, a protected web application, and a testing or analysis component. Incoming requests are inspected by the WAF before reaching the protected application. Legitimate requests are forwarded, while malicious or suspicious requests may be blocked, logged, isolated, or sent for further analysis. This environment allows repeated testing of normal and attack traffic without exposing production systems to harmful payloads (Applebaum et al., 2021; Dawadi et al., 2023).

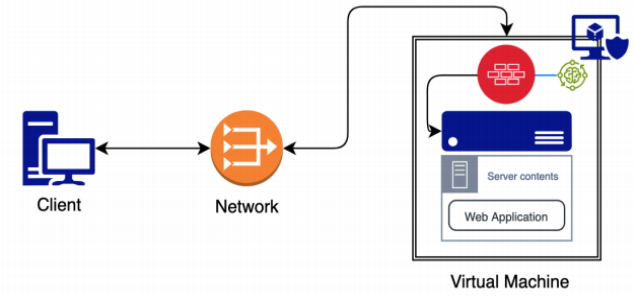


Figure 11: Controlled WAF testing environment

Figure 11 illustrates the controlled testing environment used to evaluate the proposed framework. Incoming requests are generated from the client side and first inspected by the WAF component before reaching the protected web application. Requests classified as legitimate are forwarded to the application, while malicious or suspicious requests are blocked, logged, or sent to the sandbox analysis component. Confirmed malicious anomaly patterns can then be added to the attack-pattern library and used as input for future model updates. This environment connects the detection model, active-response mechanism, logging process, sandbox validation, and adaptive pattern-library update in a single evaluation workflow.

Model performance is interpreted using accuracy, precision, recall, and F1-score. In the result section, these metrics are discussed together with the confusion matrix because WAF evaluation must consider the operational cost of both false negatives and false positives. False negatives may permit attacks to reach the web application, while false positives may block legitimate users and reduce service availability.

## 4. Results

### 4.1. Machine Learning Model Training and Evaluation

The first experiment compared five candidate models using the same pre-processed and vectorized request data: SVM, RNN, Random Forest, Logistic Regression, and CNN. The purpose of this comparison was to select the most appropriate classifier for integration into the proposed WAF workflow. Because several models produced very close scores, the comparison is interpreted as an indicator of practical suitability rather than as proof that one model is statistically superior in all conditions.

Table 8 shows that Random Forest obtained the highest accuracy, followed very closely by CNN and RNN. However, the difference between Random Forest and CNN is small: 0.9954 compared with 0.9951. Therefore, the result should be interpreted carefully. It indicates that Random Forest is a strong candidate for the proposed feature-based WAF pipeline, but it does not prove that Random Forest is universally superior to CNN or RNN under all traffic conditions.

Table 8: Model comparison result on the evaluation dataset

| Model | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Support Vector Machine (SVM) | 0.9859 | 0.985 | 0.985 | 0.985 |
| Recurrent Neural Network (RNN) | 0.9941 | 0.995 | 0.995 | 0.995 |
| Random Forest | 0.9954 | 0.995 | 0.995 | 0.995 |
| Logistic Regression | 0.9511 | 0.960 | 0.940 | 0.945 |
| Convolutional Neural Network (CNN) | 0.9951 | 0.995 | 0.995 | 0.995 |

Logistic Regression produced the weakest result among the evaluated models. This suggests that a simple linear classifier may be less suitable for representing complex request patterns, especially when malicious payloads contain obfuscation, mixed token order, or encoded characters. SVM produced strong performance but remained below the tree-based and neural-network models in this experiment.

Random Forest was selected for deployment because it combines high classification performance with practical advantages for WAF integration. It can process structured feature vectors efficiently, is easier to deploy than deep learning models, and is less computationally demanding than CNN or RNN inference in many operational settings. Nevertheless, the small difference between the top three models indicates that repeated experiments, cross-validation, and production-scale testing are needed before making a stronger claim about model superiority.

## 4.2. Model Deployment

The selected Random Forest model was deployed together with the TF-IDF vectorizer used during training. This is necessary because incoming HTTP requests are textual, while the model expects numerical feature vectors. At runtime, each request is transformed by the saved vectorizer and then classified by the trained model.

```
Random Forest Model:
RandomForestClassifier()
Number of trees in the forest: 100
Model parameters: {
                  'bootstrap': True,
                  'ccp_alpha': 0.0,
                  'class_weight': None,
                  'criterion': 'gini',
                  'max_depth': None,
                  'max_features': 'sqrt',
                  'max_leaf_nodes': None,
                  'max_samples': None,
                  'min_impurity_decrease': 0.0,
                  'min_samples_leaf': 1,
                  'min_samples_split': 2,
                  'min_weight_fraction_leaf': 0.0,
                  'n_estimators': 100,
                  'n_jobs': None,
                  'oob_score': False,
                  'random_state': None,
                  'verbose': 0, 'warm_start': False}
Feature importances: [3.24259883e-05 1.14984321e-08 4.69590909e-06 ...
9.72116376e-12
 3.08637039e-05 0.00000000e+00]

TfidfVectorizer:
TfidfVectorizer()
Feature names: ['00' '000001' '000003' ... 'zzmn' 'ä¼' '트리거']
Vocabulary size: 24733
```

Figure 12: Saved model and vectorizer used in deployment

As shown in Figure 12, the deployed detector consists of two linked components: the trained classifier and the vectorizer vocabulary. If the vectorizer used during deployment differs from the one used during training, the feature representation will change and the prediction result may become unreliable. For this reason, both components must be serialized and loaded together, for example as `rf_model.pkl` and a corresponding vectorizer file.

When a request is classified as malicious, the WAF can block the request, record the event, and trigger the active-response process. If a request is suspicious but does not match existing patterns, it is stored as anomaly data for sandbox analysis and possible pattern-library updates.

## 4.3. Comparison with Prior Work and Industrial WAF Practice

To clarify the significance of the result, the proposed framework is compared conceptually with prior WAF studies and common industrial WAF practice. The comparison focuses on detection approach, adaptive capability, and active response support rather than only on raw accuracy, because different studies use different datasets, attack distributions, and evaluation settings.

Table 9: Comparison with prior work and industrial WAF practice

| Approach | Main Technique | Main Limitation or Difference |
|---|---|---|
| Signature/rule-based WAF practice, such as ModSecurity with OWASP CRS | Static rules and signatures for known web attacks | Effective for known payloads, but requires frequent rule maintenance and may miss obfuscated or zero-day variants. Adaptive updates are mostly manual or rule-driven. |
| Cloud or managed industrial WAF practice, such as Cloudflare WAF or AWS WAF | Managed rules, reputation data, rate limiting, and provider threat intelligence | Provides strong operational coverage, but the model logic, rule tuning, and update process are usually not transparent to researchers. |
| Shaheed and Kurdy (2022) | Feature engineering and machine-learning classification | Demonstrates ML-based WAF detection, but focuses mainly on classification rather than active response and attack-pattern library feedback. |
| Dawadi et al. (2023) | Deep-learning-enabled WAF for web attack detection | Shows strong deep-learning potential, but deployment, anomaly isolation, and adaptive active response are not the central contribution. |
| Calvo and Beltrán (2022) | Adaptive WAF configuration using context and risk indicators | Focuses on adaptive WAF configuration, while the present work combines ML classification, keyword filtering, regex detection, and active response. |
| Xuan et al. (2020) | Dynamic web profiles and anomaly request detection | Supports anomaly-based updates, but does not provide the same hybrid comparison of SVM, Random Forest, Logistic Regression, CNN, and RNN. |
| Proposed framework | Random Forest classifier, keyword and regex filtering, anomaly logging, active response, and attack-pattern library update | Provides an integrated adaptive WAF prototype, but still requires broader benign traffic, repeated experiments, direct baseline comparison, and production-scale latency testing. |

Table 9 shows that the main contribution of this study is not only the use of machine learning, but the integration of classification, keyword and regex filtering, anomaly handling,

and active response in one WAF workflow. Compared with traditional rule-based WAF operation, the proposed approach is designed to learn from newly confirmed patterns. Compared with several ML-based studies, the proposed framework adds an explicit attack-pattern library and response loop.

This comparison should be interpreted as a design-level comparison rather than a direct performance benchmark. Direct numerical comparison with industrial WAF products is difficult because managed WAF services use proprietary rule sets, reputation systems, traffic intelligence, and tuning mechanisms that are not fully reproducible in an academic experiment. Therefore, this study does not claim that the proposed framework outperforms industrial WAFs. Instead, the contribution lies in demonstrating an integrated and transparent workflow that combines machine-learning classification, keyword and regex filtering, anomaly logging, active response, and pattern-library updating. A stronger performance claim would require testing the proposed framework against baseline configurations, such as ModSecurity with OWASP Core Rule Set, under the same traffic, attack distribution, OWASP Top Ten-inspired scenarios, and operational performance metrics.

## 4.4. Model Testing and Evaluation

The deployed model was tested using unseen request samples, including injection attack samples related to incidents on Indonesian military websites. This test is important because the model must generalize beyond the data used during training. The testing result is summarized in Table 10 and Table 11.

Table 10: Confusion matrix of the deployed model test

| | Predicted benign | Predicted attack |
|---|---|---|
| **Actual benign** | 0 | 7 |
| **Actual attack** | 0 | 40 |

Table 11: Performance metrics of the deployed model test

| Metric | Value |
|---|---|
| Accuracy | 85.1% |
| Precision | 85.1% |
| Recall | 100% |
| F1-score | 91.9% |

The result shows that the model detected all attack samples in the test set, producing 100% recall and zero false negatives. This is valuable for a WAF because missed attacks are high-risk events. However, the result also shows seven false positives and no true negatives in this test sample. As a result, the 85.1% accuracy should not be interpreted as a guarantee of safe production operation. It indicates that the model was aggressive in classifying suspicious requests as attacks.

From an operational perspective, this behavior has two implications. First, the model prioritizes attack coverage, which is useful when the cost of missing an attack is high. Second, the false-positive rate may reduce service availability by blocking legitimate traffic. Therefore, the deployment result supports the feasibility of the proposed approach, but it also shows that additional benign traffic, broader testing, and threshold or feature refinement are required before production use.

These findings also explain why the adaptive component is necessary. False positives and uncertain samples should be reviewed through logging, sandbox testing, and administrator validation before being added permanently to the pattern library. This prevents the system from learning incorrect patterns while still allowing it to improve over time.

## 4.5. Query Injection Pattern Creation

Query injection pattern creation is used to support the active response and adaptive learning capability of the proposed WAF. After suspicious requests are detected, the system analyzes the query structure to identify repeated or modified attack patterns. These patterns are then stored in a pattern library so they can be used for future detection, blocking, and model retraining. This process is important because attackers often modify payloads to bypass static signatures, so the WAF must be able to learn from new variations of malicious requests (Sepczuk, 2022; Calvo and Beltrán, 2022).

The pattern creation process compares existing injection patterns with new patterns discovered during detection and anomaly analysis. Existing patterns represent known attack forms that have already been stored in the detection library, while new patterns represent modified payloads or newly observed query structures. The comparison helps the WAF identify whether an incoming payload is a known attack or a new variation that should be added to the library (Xuan et al., 2020).

Table 12 shows examples of existing and newly generated injection patterns. The new patterns are created by modifying parts of known SQL injection payloads, such as replacing `SELECT` with `FROM`, changing keyword order, or preserving obfuscated forms such as mixed capitalization and encoded characters. These variations show how attackers can create new payload forms while keeping the same malicious intent.

Table 12: Injection pattern result

| Existing Patterns | New Patterns |
|---|---|
| `+union+distinct+select+` | `+union+distinct+FROM+select+` |
| `+union+distinctROW+select+` | `+union+distinctROW+FROM+select+` |
| `///!12345UNIONSELECT///` | `///!12345UNIONFROM///` |
| `///!50000UNIONSELECT///` | `///!50000UNIONFROM///` |
| `+/!50000UnIoN/!50000SeLeCtaLl/+` | `+/!50000UnIoN/!50000FROMaLl/+` |
| `+/!u%6eion/+/!se%6cect/+` | `+/!u%6eion/+/!FROM%6cect/+` |
| `//uniUNIONon//aALLI///selSELECTect//` | `//uniUNIONon//aALLI///selFROMect//` |
| `1%')and(0)union(select(1),version(),3,4,5,6)%23%23%23` | `1%')and(0)union(FROM(1),version(),3,4,5,6)%23%23%23` |
| `/!50000%55nIoN/+/!50000%53eLeCt/` | `/!50000%55nIoN/+/!50000%53eLeFROM/` |

The generated patterns are useful for future attack prediction because they expand the WAF's detection knowledge beyond the original payload examples. When a new pattern is confirmed as malicious, it can be added to the pattern library and used by the regex, keyword, and machine learning modules. This allows the WAF to update its detection

**Table 13**
Confusion matrix of the attack-pattern library test

| | Predicted benign | Predicted attack |
|---|---|---|
| **Actual benign** | 0 | 0 |
| **Actual attack** | 19 | 251 |

**Table 14**
Performance metrics of the attack-pattern library test

| Metric | Value |
|---|---|
| Accuracy | 93% |
| Precision | 100% |
| Recall | 93% |
| F1-score | 96% |

capability without relying only on manually written static rules.

The pattern library also supports administrator review and sandbox testing. Suspicious anomaly data can be cross-checked before being added to the library, reducing the risk of adding incorrect patterns. This process strengthens the active response mechanism because the WAF can block confirmed malicious patterns, log suspicious variations, and improve future detection through continuous updates (Shahid et al., 2022b; Xuan et al., 2020).

Overall, query injection pattern creation helps transform detected anomalies into reusable security knowledge. By continuously updating the pattern library, the proposed WAF becomes more adaptive against emerging injection techniques and more resilient against payload obfuscation.

## 4.6. Library Testing

Library testing was conducted to evaluate the effectiveness of the attack-pattern library in detecting injection patterns. The library contains known malicious patterns generated from attack keywords, regex rules, and confirmed anomaly data. This stage is important because the proposed WAF does not rely only on the machine learning model; it also uses a pattern library to support immediate detection and active response against known or repeated attack payloads (Sun and Li, 2012; Kar et al., 2016).

The testing process compares incoming query patterns with the existing pattern library. If a query matches a known malicious pattern, the WAF can immediately classify it as an attack and trigger a response. If the query does not fully match known patterns but appears suspicious, it can be treated as anomaly data and sent for further analysis or sandbox testing. This process supports adaptive learning because confirmed new patterns can be added to the library for future detection (Xuan et al., 2020; Calvo and Beltrán, 2022).

The attack-pattern library was also evaluated separately because the proposed WAF does not rely only on the machine-learning model. The library is intended to provide fast detection for known or repeated malicious patterns. The result is summarized in Table 13 and Table 14. The tabular presentation is used because it reports the exact tested values and avoids ambiguity between graphical and numerical results.

The library test shows a different trade-off from the deployed-model test. The pattern library produced no false positives in the tested sample, which means that every request flagged by the library was malicious. However, it missed 19 attack samples, showing that the library alone is not sufficient for complete protection. This supports the hybrid design of the proposed WAF: the pattern library provides precise detection for known attacks, while anomaly detection and machine learning are needed to identify modified or previously unseen payloads.

Overall, the library result supports the adaptive WAF concept, but it also confirms the need for continuous pattern updates. Newly confirmed malicious anomalies should be added to the library only after validation, so that recall can improve without introducing unnecessary false positives.

## 4.7. Adaptive Concept

The adaptive concept is designed to ensure that the proposed WAF can improve its detection capability over time. Traditional WAF systems often depend on static signatures and manually configured rules, which can become ineffective when attackers modify payloads or create new attack patterns. Therefore, the proposed framework integrates machine learning, known pattern detection, anomaly analysis, sandbox testing, and model updating to support continuous adaptation (Applebaum et al., 2021; Sepczuk, 2022; Calvo and Beltrán, 2022).

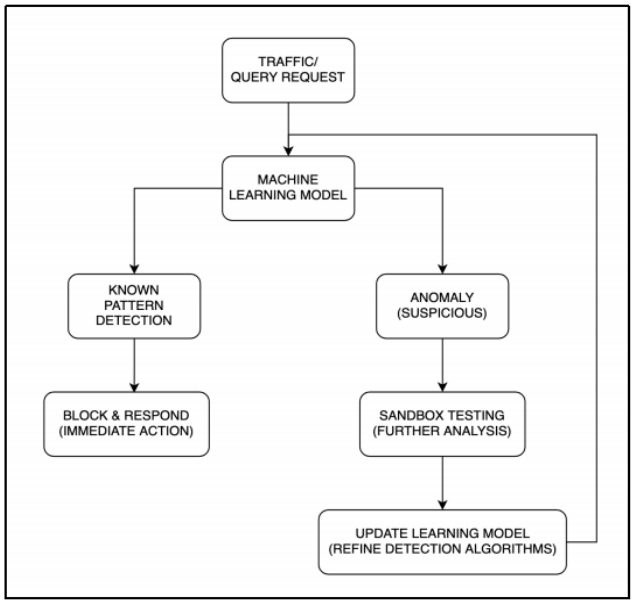


Figure 13: Adaptive concept

As shown in Figure 13, incoming traffic or query requests are first processed by the machine learning model. If the request matches a known malicious pattern, the system performs immediate action by blocking and responding to the request. This path allows the WAF to handle known attacks quickly without requiring additional manual analysis.

If the request does not match a known pattern but appears suspicious, it is treated as anomaly data. The suspicious

request is then forwarded to sandbox testing for further analysis. Sandbox testing helps determine whether the request is benign, malicious, or a new variation of an existing attack. This step is important because anomaly data may contain new payload structures that are not yet included in the current pattern library (Xuan et al., 2020; Mohammadi Rouzbahani et al., 2020).

After the suspicious request is analyzed, confirmed malicious patterns can be used to update the learning model and refine the detection algorithm. This creates a feedback loop in which new attack knowledge is added to the WAF system. As a result, the WAF becomes more resilient against emerging threats because it does not depend only on the original training dataset or static detection rules.

The adaptive concept also supports active response. Known malicious traffic can be blocked immediately, while unknown suspicious traffic can be isolated and reviewed before it is added to the detection library. This combination reduces the risk of allowing new attacks to reach the protected web application and also reduces the risk of incorrectly blocking legitimate traffic. By combining detection, response, sandbox analysis, and model refinement, the proposed WAF can support more adaptive handling of modified and zero-day attack patterns under the evaluated framework (Shahid et al., 2022b; Calvo and Beltrán, 2022).

### 4.8. Addressing Problem Statement

The proposed WAF framework is designed to address several weaknesses commonly found in traditional WAF implementations. These weaknesses include dependency on static rules, false positives and false negatives, architectural complexity, performance impact, and lack of application awareness. Each problem is addressed using a specific method in the proposed system, including machine learning, anomaly detection, regex and keyword filtering, optimized pattern matching, and adaptive model updating (Applebaum et al., 2021; Calvo and Beltrán, 2022; Shaheed and Kurdy, 2022).

Table 15 shows how the proposed approach addresses the main problem statements of the research. Static rule dependency is handled through machine learning models that can learn from updated datasets and support adaptation to new threats. This is important because static signatures alone may fail when attackers use obfuscated or modified payloads (Sepczuk, 2022; Xuan et al., 2020).

False positives and false negatives are addressed using anomaly detection, stopword or attack-keyword filtering, and regex-based pattern recognition. These methods help the system distinguish between legitimate and malicious query structures. By combining signature-like filtering with machine learning, the WAF can reduce missed attacks and avoid blocking legitimate traffic unnecessarily (Kar et al., 2016; Sun and Li, 2012).

System complexity is addressed through a modular and scalable WAF architecture. The proposed framework separates data pre-processing, feature extraction, machine learning classification, pattern matching, anomaly logging, and active response into manageable components. This modular design supports future updates, model replacement, and pattern library expansion.

Table 15: Addressing problem statement

| Problem Statement | Method | Solution |
|---|---|---|
| Static Rules | Machine Learning (SVM, Random Forest) | Dynamic adaptation to new threats by learning from updated datasets and threat intelligence feeds. |
| False Positives and Negatives | Anomaly Detection, Stopwords, and Regex | Improved accuracy in identifying malicious queries, reducing both false positives and false negatives through continuous learning and pattern recognition. |
| Complexity | Modular and Scalable WAF Architecture | Simplified integration and management of the WAF system, allowing easy updates and scalability. |
| Performance Impact | Efficient Model Training and Execution, Compiled Patterns, and Optimized Searching | Optimized machine learning models ensure minimal performance degradation while maintaining high security. Pre-compiling regex patterns and using efficient searching algorithms such as Aho-Corasick enhance performance during pattern matching. |
| Lack of Application Awareness | Context-Aware Anomaly Detection, Stopwords, and Regex | Enhanced understanding of application-specific queries through stopwords and contextual features, leading to more accurate detection. |

Performance impact is addressed through efficient model training, compiled regex patterns, and optimized searching. Pre-compiled regex patterns reduce matching overhead, while efficient multiple-pattern matching algorithms such as Aho-Corasick can improve performance when many patterns must be checked at the same time (Sun and Li, 2012; Coscia et al., 2023).

Lack of application awareness is addressed using context-aware anomaly detection and application-specific keyword analysis. By learning normal request structures and comparing them with suspicious query behavior, the WAF can better understand whether a request is normal for the protected application or potentially malicious. This supports more accurate detection and continuous improvement of the WAF system (Calvo and Beltrán, 2022; Mohammadi Rouzbahani et al., 2020).

## 5. Limitations

This study has several limitations. First, the deployed-model test used a limited test sample and produced no true negatives, which restricts the interpretation of accuracy and precision. Although the recall reached 100%, the presence of seven false positives shows that the model may be too aggressive in classifying suspicious requests as attacks.

Second, the model comparison did not include repeated runs, cross-validation, confidence intervals, or statistical significance testing. Therefore, the small performance difference between Random Forest, CNN, and RNN should be interpreted cautiously.

Third, the comparison with industrial WAFs is conceptual rather than experimental. Industrial WAF products and managed WAF services use proprietary rules, traffic reputation,

threat intelligence, and operational tuning that are not directly reproduced in this study.

Fourth, the current evaluation focuses on selected request-level attacks, including SQL injection, XSS, path traversal, and CSRF-related patterns. It does not fully cover all OWASP Top Ten categories. Future work should expand the dataset and evaluate the framework across broader attack scenarios.

## 6. Conclusion

This research proposed a hybrid Web Application Firewall framework that integrates machine-learning classification, attack-keyword filtering, regex-based pattern matching, anomaly handling, active response, and an adaptive attack-pattern library. The framework was designed to address limitations of static rule-based WAF approaches, especially when facing modified, obfuscated, or previously unseen request patterns.

The model comparison showed that Random Forest achieved the highest accuracy among the evaluated models, with 0.9954, followed closely by CNN with 0.9951 and RNN with 0.9941. However, the differences among the top-performing models were small. Therefore, Random Forest should be interpreted as the most practical candidate in this experiment, not as a universally superior model. Its selection is supported by both its classification result and its relative simplicity for feature-based deployment.

The deployed-model test showed 85.1% accuracy, 85.1% precision, 100% recall, and 91.9% F1-score. This result is important because no attack samples were missed in the test set, but it also reveals a limitation: seven benign or negative samples were classified as attacks. For a real WAF deployment, these false positives may affect service availability. Therefore, the result supports feasibility but does not yet guarantee production-level security.

The attack-pattern library achieved 93% accuracy, 100% precision, 93% recall, and 96% F1-score. This indicates that the library can provide precise detection for known malicious patterns, but the missed samples show that static pattern storage alone is not sufficient. The library must be continuously updated using validated anomaly data.

Overall, the study supports the value of combining machine learning, keyword and regex filtering, pattern-library detection, and active response in one adaptive WAF workflow. The main contribution is the integrated design and feedback loop rather than a claim of complete superiority over existing WAFs. Future work should compare the framework directly with baseline rule-based WAF configurations and industrial WAF settings, expand benign and malicious datasets, measure latency and throughput, and evaluate coverage against OWASP Top Ten-inspired attack scenarios.

## 7. Recommendation

Several improvements are recommended before the proposed framework can be considered ready for production-oriented evaluation. First, the dataset should be expanded with more diverse benign traffic. The deployed-model test produced false positives, so additional normal request samples are needed to help the model distinguish suspicious but legitimate requests from actual attacks.

Second, future experiments should include direct baseline comparisons. The proposed framework should be tested against a rule-based WAF configuration, such as ModSecurity with OWASP Core Rule Set, and compared with results from related ML-based WAF studies where comparable datasets are available. This would make the claim of improvement more measurable.

Third, the attack coverage should be mapped more systematically to OWASP Top Ten-inspired scenarios. The current dataset includes SQL injection, XSS, path traversal, and CSRF-related patterns, but it does not represent all major web application risks. Broader evaluation would clarify which threat categories are actually covered by the proposed framework.

Fourth, the active-response mechanism should be evaluated in a production-like environment. Future testing should measure not only accuracy, precision, recall, and F1-score, but also latency, throughput, false-positive handling time, logging overhead, and the operational impact of blocking or isolating suspicious traffic.

Finally, the adaptive learning process should include careful validation before new patterns are added to the library or used for retraining. Sandbox testing and administrator review remain important to prevent the WAF from learning incorrect patterns and increasing false positives.

## 8. Authors Contribution Statement

Agung Prasetiawan contributed to all stages of this research. The author was responsible for identifying the research problem, reviewing related literature, designing the proposed Web Application Firewall framework, preparing the dataset, conducting data pre-processing, implementing the machine learning models, evaluating model performance, analyzing the results, and preparing the manuscript.

The author also designed the active response and adaptive learning concept, including attack keyword filtering, regex-based detection, anomaly handling, attack-pattern library testing, and model deployment strategy. The author read and approved the final version of the manuscript.

## References

Applebaum, S., Gaber, T., and Ahmed, A. (2021). Signature-based and machine-learning-based web application firewalls: A short survey. *Procedia Computer Science*, 189:359–367.

Calvo, M. and Beltrán, M. (2022). An adaptive web application firewall. In *Proceedings of the 19th International Conference on Security and Cryptography*, pages 96–107. SCITEPRESS – Science and Technology Publications.

Chicco, D., Warrens, M. J., and Jurman, G. (2021). The matthews correlation coefficient (mcc) is more informative than cohen's kappa and brier score in binary classification assessment. *IEEE Access*, 9:78368–78381.

Coscia, A., Dentamaro, V., Galantucci, S., Maci, A., and Pirlo, G. (2023). An innovative two-stage algorithm to optimize firewall rule ordering. *Computers & Security*, 134:103423.

Dawadi, B., Adhikari, B., and Srivastava, D. (2023). Deep learning technique-enabled web application firewall for the detection of web attacks. *Sensors*, 23(4):2073.

Jemal, I., Haddar, M. A., Cheikhrouhou, O., and Mahfoudhi, A. (2022). Swaf: A smart web application firewall based on convolutional neural network. In *2022 15th International Conference on Security of Information and Networks (SIN)*, pages 1–6. IEEE.

Kar, D., Panigrahi, S., and Sundararajan, S. (2016). Sqligot: Detecting sql injection attacks using graph of tokens and svm. *Computers & Security*, 60:206–225.

Manaseer, S. and Al Hwaitat, A. K. (2018). Centralized web application firewall security system. *Modern Applied Science*, 12(10):164.

Mohammadi Rouzbahani, H., Karimipour, H., Rahimnejad, A., Dehghantanha, A., and Srivastava, G. (2020). Anomaly detection in cyber-physical systems using machine learning. In *Handbook of Big Data Privacy*, pages 219–235. Springer International Publishing.

Riera, T. S., Higuera, J.-R. B., Higuera, J. B., Herraiz, J.-J. M., and Montalvo, J.-A. S. (2022). A new multi-label dataset for web attacks capec classification using machine learning techniques. *Computers & Security*, 120:102788.

Sepczuk, M. (2022). Dynamic web application firewall detection supported by cyber mimic defense approach. *SSRN Electronic Journal*.

Shaheed, A. and Kurdy, M. H. D. B. (2022). Web application firewall using machine learning and features engineering. *Security and Communication Networks*, 2022:1–14.

Shahid, W. B., Aslam, B., Abbas, H., Afzal, H., and Khalid, S. B. (2022a). A deep learning assisted personalized deception system for countering web application attacks. *Journal of Information Security and Applications*, 67:103169.

Shahid, W. B., Aslam, B., Abbas, H., Khalid, S. B., and Afzal, H. (2022b). An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. *Journal of Network and Computer Applications*, 198:103270.

Sun, L. and Li, L. (2012). Improve aho-corasick algorithm for multiple patterns matching memory efficiency optimization. *Journal of Convergence Information Technology*, 7(19):162–168.

Surendhar, K., Pandey, B. K., Geetha, G., and Gohel, H. (2023). Detection of payload injection in firewall using machine learning. In *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 186–190. IEEE.

Tiwari, C., Pillai, S., Obaid, A. J., Saear, A. R., and Sabri, A. K. (2023). Integration of artificial intelligence/machine learning in developing and defending web applications. In *AIP Conference Proceedings*, volume 2990, page 060038. AIP Publishing.

Tundis, A., Ruppert, S., and Mühlhäuser, M. (2022). A feature-driven method for automating the assessment of osint cyber threat sources. *Computers & Security*, 113:102576.

Wang, S., Liu, R., Guo, X., and Wei, G. (2022). Design of web application firewall system through convolutional neural network and deep learning. In *2022 International Conference on Computers, Information Processing and Advanced Education (CIPAE)*, pages 454–457. IEEE.

Xuan, C. D., Nguyen, N., and Dinh, H. N. (2020). An adaptive anomaly request detection framework based on dynamic web application profiles. *International Journal of Electrical and Computer Engineering (IJECE)*, 10(5):5335–5346.