Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

Deployment-Aware Evaluation of a Machine Learning-Based Web Application Firewall for Academic E-Voting Systems

Muhammad Zainuddin[1], Achmad Zakki Falani[2*], Hariadi Yutanto[3]
[1,3]Informatics, Universitas Hayam Wuruk Perbanas, Indonesia, [2]Information Systems, Universitas Narotama, Indonesia
[1]202202011008@mhs.hayamwurukperbanas.ac.id, [2*]achmad.zakki@narotama.ac.id, [3]antok@perbanas.ac.id

open access

***Corresponding Author**

Article History:
Submitted: 04-10-2026
Accepted: 13-10-2026
Published: 01-11-2026

Keywords:
Cross-Site Scripting; E-Voting; Machine Learning; Random Forest; SQL Injection; Web Application Firewall.

Brilliance: Research of Artificial Intelligence is licensed under a Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0).

ABSTRACT

Web applications in higher education, including e-voting systems for student-organization elections, are frequent targets of SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks. Machine Learning (ML) based Web Application Firewall (WAF) studies usually select models by offline accuracy on curated datasets, which says little about how a model behaves once it filters live requests. This study aims to build an ML-based WAF embedded as middleware in an academic e-voting application and to test whether offline accuracy is a sufficient criterion for selecting a deployable model. HTTP request payloads were vectorized with character-level TF-IDF n-grams and Chi-Square feature selection, and Decision Tree (DT) and Random Forest classifiers were integrated into a Flask-based WAF that performs allow, block, and log actions. The models were assessed with a three-layer deployment-aware protocol: offline hold-out evaluation, black-box testing of the integrated middleware, and an independent stress test of 500 requests containing obfuscated attacks and look-alike benign inputs. Offline, the two models differed by less than 0.3 percentage points, with Random Forest reaching 99.54% accuracy for SQLi and 100% for XSS. Under the stress test, however, DT missed four obfuscated SQLi payloads, whereas Random Forest blocked all 400 attacks with 99.2% overall accuracy and a mean inference latency of 5.5 ms. Offline accuracy therefore hid a security-relevant difference that only deployment-aware testing revealed. The protocol and the security-first selection criterion offer a practical basis for evaluating ML-based WAFs in institutional web applications.

INTRODUCTION

The accelerating adoption of web-based applications has made them indispensable across business, government, and education, yet it has also widened the attack surface available to adversaries. Among the most persistent threats are SQL Injection (SQLi) and Cross-Site Scripting (XSS), both of which remain among the critical web risks catalogued by the Open Web Application Security Project (OWASP) (Mani & Shenoy, 2025; Nair, 2024). In higher education, web applications such as academic information systems, learning management systems, and electronic voting (e-voting) platforms store personal data and mediate processes whose integrity must be guaranteed. This makes them attractive targets and raises the cost of a successful compromise (Kepuska & Tomašević, 2024).

A Web Application Firewall (WAF) is the conventional countermeasure for these threats. Traditional WAFs operate at the application layer and rely on predefined signatures or rules. They are effective against known patterns but generalize poorly to obfuscated or previously unseen payloads (Shaheed & Kurdy, 2022). To address this rigidity, a growing body of research integrates machine learning (ML) into WAFs so that malicious traffic is identified from learned statistical patterns rather than fixed rules (Asan Nainar & Sibikarthik, 2025) ; (Dawadi et al., 2023). Most of these studies report high detection accuracy on benchmark or curated datasets, as reviewed in the next section.

Despite this progress, three gaps remain underexplored. First, many ML-WAF studies stop at offline model evaluation (Alhamyani & Alshammari, 2024; Younas et al., 2024). A classifier is trained and its accuracy, precision, recall, and F1-score are reported, but the model is seldom embedded into a live application and exercised end-to-end. Its deployment behavior, including inference latency and request-handling logic, therefore remains largely unmeasured. Second, very high accuracy on curated payload corpora is often presented as the headline contribution. Near-perfect scores obtained from high-dimensional lexical features may reflect memorization of distinctive substrings rather than genuine generalization, a concern reinforced by evidence that ML-based firewalls can be evaded by adversarially crafted payloads (Floris, 2025). Third, little work situates ML-WAF deployment within an academic e-voting system, where both data confidentiality and process integrity are at stake.

This study addresses these gaps with a deployment-aware evaluation of an ML-based WAF embedded in the request-handling path of an e-voting application used for student-organization elections. The novelty lies not in the algorithm, since character-level term frequency–inverse document frequency (TF-IDF) with Random Forest is established, but in how the model is evaluated and selected for deployment. Three research questions guide the study.

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

RQ1: Can a lightweight ML classifier operate as in-line middleware in an academic e-voting application with acceptable latency? RQ2: Does offline hold-out accuracy rank Decision Tree (DT) and Random Forest (RF) classifiers in the same order as their behavior on harder, obfuscated traffic? RQ3: Which classifier should be deployed when attack containment is prioritized over occasional false positives? The contributions are threefold. First, we propose a three-layer deployment-aware evaluation protocol that combines offline classification assessment, black-box validation of the integrated middleware, and an independent robustness stress test with obfuscated attacks and hard benign inputs. Second, we provide empirical evidence that offline accuracy can conceal operational differences: two classifiers that differ by less than 0.3 percentage points offline diverge clearly in missed attacks under obfuscation. Third, we present a working ML-WAF architecture for an academic e-voting system, together with a security-first model selection criterion based on attack-class recall and inference latency rather than aggregate accuracy.

LITERATURE REVIEW

Research on ML-based detection of web attacks has moved from classical classifiers toward deep and feature-fusion models. Shaheed and Kurdy (2022) combined feature engineering with classical ML classifiers in a WAF and reported 99.6% accuracy on a benchmark dataset and 98.8% on traffic from a real server (Shaheed & Kurdy, 2022). Dawadi et al. (2023) proposed a layered deep-learning WAF based on Long Short-Term Memory (LSTM) networks for DDoS and for XSS and SQLi attacks, reporting 89.34% accuracy for the XSS and SQLi layer in a live evaluation (Dawadi et al., 2023). Tadhani et al. (2024) used a hybrid convolutional neural network and LSTM model and reported 99.84% accuracy on payload data and 99.23% on a testbed (Tadhani et al., 2024). Bakır (2025) fused multiple feature representations with ML techniques to detect XSS and SQLi jointly (Bakır, 2025).

Attack-specific studies show a similar pattern. For XSS, Alhamyani and Alshammari (2024) applied ML-driven classifiers (Alhamyani & Alshammari, 2024), Younas et al. (2024) proposed an efficient artificial intelligence approach for early detection (Younas et al., 2024), Qin et al. (2024) combined Random Forest with multi-channel feature extraction (Qin et al., 2024) , and Liu et al. (2023) proposed MFXSS, a multi-feature model for detecting XSS vulnerabilities in JavaScript (Liu et al., 2023) . For SQLi, Rosca et al. (2025) compared several ML models (Rosca et al., 2025) , Casmiry et al. (2025) combined Chi-Square feature selection with classical classifiers (Casmiry et al., 2025), Sun et al. (2023) implemented deep-learning-based detection (Sun et al., 2023), and Paul et al. (2024) addressed detection together with prioritization and prevention (Paul et al., 2024). Other studies moved from payload strings to traffic data: Crespo-Martínez et al. (2023) detected SQLi in network flow data (Crespo-Martínez et al., 2023), and Irfa'issurur and Josaphat (2025) detected brute-force, XSS, and SQLi attacks with ML (Irfa'issurur & Josaphat, 2025). Ramadhan and Ashari (2024) compared Random Forest and Decision Tree for network anomaly detection and found Decision Tree slightly more accurate and considerably faster on binary data, with Random Forest only marginally better on multi-class data. Decision Tree is therefore a credible and efficient candidate, so the present study compares the two classifiers under deployment conditions rather than assuming that the ensemble is superior (Ramadhan & Ashari, 2024).

Reviews confirm the breadth of this literature. Mani and Shenoy (2025) reviewed ML models in web applications, and Rahayu et al. (2025) systematically reviewed ML-based SQLi detection (Rahayu et al., 2025); (Mani & Shenoy, 2025). At the same time, robustness has become a central concern. Floris et al. (2025) showed that ML-based SQLi detectors can be evaded by adversarially crafted payloads and proposed robust learning to counter them, while Babaey and Ravindran (2025) proposed a generative artificial intelligence framework for securing WAFs against SQLi attacks (Babaey & Ravindran, 2024).

Taken together, many of these studies evaluate detection offline or on testbeds, and those that report real-server or live results tend to show lower accuracy than on benchmarks (Dawadi et al., 2023; Shaheed & Kurdy, 2022). Robustness studies such as Floris et al. (2025) examine adversarial evasion but do not address integration into an operational application (Floris, 2025). To the best of our knowledge, none of the reviewed studies combines in-line integration in an institutional application, independent obfuscation testing, and latency measurement in a single evaluation, or examines whether offline accuracy preserves the ranking of candidate models under such conditions. The present study targets this gap.

METHOD

The system was developed using the prototype methodology, which supports incremental construction, evaluation, and refinement of a security artifact through iterative feedback; a comparable iterative prototyping approach has been used to build and evaluate an SQL injection detection and mitigation tool for web applications (Rodriguez Gavilanes et al., 2025).

Deployment-Aware Evaluation Protocol

The evaluation follows a three-layer protocol designed to answer RQ1–RQ3. Layer 1, offline assessment, measures classification performance on a stratified hold-out split. Layer 2, integrated validation, tests the deployed middleware end-to-end with black-box requests and records the response time. Layer 3, a robustness stress test, passes an independent and harder request set containing obfuscated attacks and look-alike benign inputs through the WAF

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

decision function and records the action and latency for each request. A model is then selected by a security-first criterion: the highest attack-class recall in Layer 3, subject to an inference latency acceptable for interactive use, with the number of false positives as a tie-breaker. The following subsections describe the system, the data, the models, the metrics, and each layer.

System Architecture

The proposed system consists of five principal components: the client, the WAF API, the preprocessing module, the ML classification model, and the firewall decision layer. When the client issues an HTTP request to the e-voting application, the request is first intercepted by the WAF API. The preprocessing module extracts the relevant request content and transforms it into the numerical representation expected by the model. The classifier then labels the request as benign or malicious, and the firewall decision layer maps this label to one of three actions: allow (forward the request to the application server), block (return a generic block page), or log (store the event in a database). All decisions are recorded for monitoring and auditing. The component flow is shown in Figure 1.



Figure 1. Component flow of the proposed ML-based WAF

Figure 1 shows that the WAF is arranged as a sequential pipeline placed in front of the e-voting application rather than as a separate monitoring tool. The first stage, input data, is the HTTP request content captured by the WAF API from the client. Preprocessing converts this content into a character-level TF-IDF vector, classification assigns a benign or malicious label, and the firewall decision stage turns that label into an allow or block action. The final stage, log recording, stores each decision so that blocked requests can be audited after an election. Because every request must pass through all five stages before it reaches the application server, the practical value of the design depends on the latency of the whole chain and not only on the accuracy of the classifier; for this reason, latency is measured in Layers 2 and 3.

Dataset and Preprocessing

The dataset comprises HTTP request payloads labeled as benign or malicious. Malicious samples are SQLi and XSS payloads compiled from publicly circulated payload collections, while benign samples are legitimate request strings. Two independent detection tasks were defined, one for SQLi and one for XSS. A separate corpus was prepared for each task; the two corpora share the same benign class, while the malicious class is task-specific. Table 1 summarizes the class composition. Both corpora are imbalanced toward the benign class, at about 4.6:1 for SQLi and 8.6:1 for XSS, which is why stratified sampling and macro-averaged metrics are used.

Table 1. Class Composition of the SQLi and XSS Detection Corpora

Class	SQLi task	XSS task	Source
Benign	15,260	15,260	Legitimate request strings
Malicious (SQLi)	3,290	–	Public payload lists
Malicious (XSS)	–	1,770	Public payload lists
Total	**18,550**	**17,030**	

For the final confusion matrices, each corpus was split 80:20 with stratification. The held-out partition therefore contains 3,710 samples for SQLi (3,052 benign and 658 SQLi) and 3,406 samples for XSS (3,052 benign and 354 XSS). Because the corpus is assembled from curated payload lists rather than captured production traffic, its classes are highly separable by construction; this point is revisited in the Discussion.

Each request string was converted into a numerical vector using character-level TF-IDF n-grams. This representation suits web-attack payloads because it captures character distributions and short character sequences, such as quotes, comment markers, and script tags, without application-specific parsing. Chi-Square feature selection was then applied to reduce dimensionality and retain the most discriminative features, following its reported use for SQLi detection with classical classifiers (Casmiry et al., 2025).

Machine Learning Models

Two supervised classifiers were evaluated: DT and RF. DT was selected for its interpretability and fast inference, which are desirable for in-line filtering, whereas RF, an ensemble of decision trees, was selected because averaging many trees reduces the variance of a single tree, which is expected to help with noisy and varied payloads. Both models

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence
E-ISSN : 2807-9035
Volume 6, Number 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

were trained separately for SQLi and XSS detection. Each model was trained on the stratified 80% training partition and evaluated on the 20% hold-out partition (Layer 1), while the independent stress test (Layer 3) serves as the out-of-distribution check. Figure 2 summarizes the pipeline from dataset preparation to evaluation.



Figure 2. Data and machine-learning model design pipeline

Figure 2 is read from left to right as six sequential stages. Dataset preparation assembles and labels the SQLi and XSS corpora summarized in Table 1. Feature extraction converts each request string into character-level TF-IDF n-grams, and feature selection applies the Chi-Square (χ^2) test to retain the most discriminative n-grams. Model training fits the DT and RF classifiers on the 80% training partition using the hyperparameters in Table 2, model evaluation measures them on the 20% hold-out partition with the metrics defined in (1)–(4), and the final stage interprets the results to select the model to be deployed. The pipeline was executed separately for the SQLi and XSS tasks, so each task has its own vectorizer, feature selector, and classifier.

The hyperparameters, listed in Table 2, were selected from preliminary experiments and configurations commonly adopted in web-attack detection studies. RF used 100 trees to balance classification performance and inference efficiency, while both models retained unrestricted tree depth to preserve discriminative capability in the high-dimensional TF-IDF feature space.

Table 2. Hyperparameter Configuration of the Evaluated Models

Parameter	Decision Tree	Random Forest
Criterion	Gini	Gini
Max depth	None	None
Min samples split	2	2
Min samples leaf	1	1
Number of trees	–	100
Bootstrap	–	True
Random state	42	42

Evaluation Metrics

Classification performance was measured with accuracy, precision, recall, and F1-score, supported by the confusion matrix. The metrics are defined in (1)–(4), where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives, respectively.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

$$F1-Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (4)$$

Because the corpora are imbalanced, precision, recall, and F1-score in Table 3 are macro-averaged, that is, computed separately for the benign and malicious classes and then averaged. In a security context, the recall of the attack class matters most, because a false negative is an undetected attack reaching the application server, whereas precision governs how often legitimate users are wrongly blocked. Attack-class recall is therefore reported separately in Table 4.

Layer 2: Integrated Middleware Validation

The best-performing model was serialized and integrated into the WAF through a Flask-based API. At runtime, the middleware receives each HTTP request, applies the preprocessing pipeline, classifies the request, and enforces the corresponding allow, block, or log action, and the prediction and outcome are written to a log database. System verification used black-box testing, which evaluates behavior from inputs and outputs without reference to internal structure and is well suited to the security testing of web applications (Aydos et al., 2022; Santi et al., 2022). Two

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

Postman test suites, one for SQLi and one for XSS, each containing five requests, were dispatched to confirm that malicious requests were intercepted before reaching the application server while benign requests were served normally.

Layer 3: Robustness Stress Test

Because the Postman suites contained only a few requests, the firewall logic was also evaluated on a larger and deliberately harder set using a reconstructed pipeline that mirrors the deployed configuration (character-level TF-IDF n-grams, Chi-Square selection, and DT and RF classifiers). The set comprised 500 requests: 200 SQLi (120 plain and 80 obfuscated through case randomization, inline comments, and URL encoding), 200 XSS (120 plain and 80 obfuscated), and 100 benign requests, including 10 hard negatives that superficially resemble attacks, such as natural-language text containing the words *select*, *union*, or *alert*. Attack requests were adapted from the same public payload families used for training and then hardened, and none were drawn from the training partition. Each request was passed through the WAF decision function, and the resulting action and per-request latency were recorded. This test is a complementary robustness check rather than a substitute for the deployed system.

RESULT

System Implementation

The ML-based WAF was implemented and integrated with the academic e-voting application. The realized system comprises the preprocessing module, the classification model, and the firewall decision layer, and it follows the intended workflow. An incoming HTTP request is preprocessed, classified, and resolved to an allow, block, or log action, and every event is stored in the log database for monitoring. The request-handling workflow is shown in Figure 3.

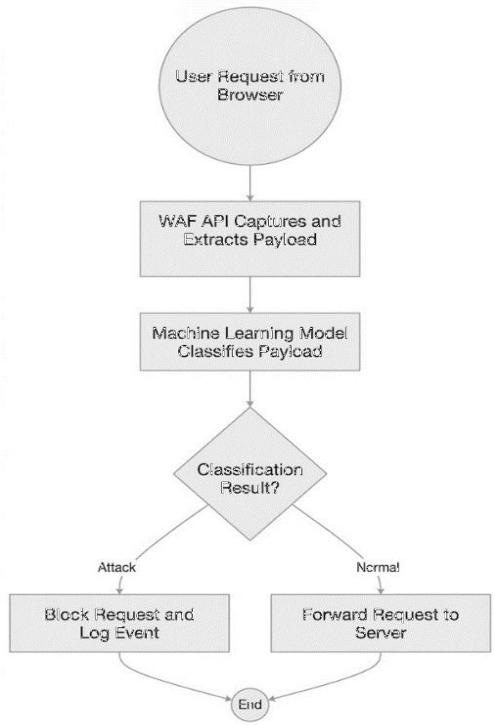


Figure 3. Request-handling workflow of the integrated WAF

Figure 3 traces the run-time path of a single request in the deployed system. A request sent from the voter's browser is first captured by the WAF API, which extracts the payload and passes it to the machine learning model. At the decision node, the flow branches on the classification result: a request labeled as an attack is blocked and the event is logged, whereas a request labeled as normal is forwarded to the e-voting server, after which both branches terminate. The figure shows that classification takes place before the request reaches the application, so a detected attack never reaches the voting database. It also shows that the decision is made per request without human intervention; consequently, a false positive directly blocks a legitimate voter and a false negative directly exposes the server, which is why both error types are analysed in the following subsections.

Layer 1: Offline Detection Performance

Table 3 reports the hold-out performance of both classifiers for each task. For SQLi, DT achieved 99.25% accuracy and RF 99.54%. For XSS, DT achieved 99.94% and RF 100%. RF was equal to or better than DT on every

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

metric for both tasks. On this layer alone, the two classifiers appear nearly interchangeable: the accuracy gap is 0.29 percentage points for SQLi and 0.06 for XSS.

Table 3. Detection Performance on the Hold-Out Split (Macro-Averaged)

Task	Model	Accuracy	Precision	Recall	F1-Score
SQLi	Decision Tree	0.9925	0.9923	0.9817	0.9869
SQLi	Random Forest	0.9954	0.9942	0.9901	0.9921
XSS	Decision Tree	0.9994	0.9997	0.9972	0.9984
XSS	Random Forest	1.0000	1.0000	1.0000	1.0000

Three observations follow from Table 3. First, all macro-averaged scores exceed 0.98, indicating that both classifiers separate benign and malicious payloads well when the test data come from the same distribution as the training data. Second, the SQLi task is harder than the XSS task for both models: the lowest values in the table are the SQLi recall scores (0.9817 for DT and 0.9901 for RF), whereas the XSS scores are at or near 1.0. A plausible reason is that SQLi payloads share many characters and keywords with ordinary input, such as quotes, equals signs, and words like select, while XSS payloads usually contain distinctive markers such as script tags and event handlers. Third, the advantage of RF over DT on SQLi is larger in recall (0.0084) than in precision (0.0019), which indicates that RF gains mainly by detecting more attacks rather than by blocking fewer benign requests. Because recall in Table 3 is macro-averaged, it does not show how many attacks were actually missed; this is examined next through the confusion matrices.

Confusion Matrix Analysis

Table 4 reports the confusion matrices on the hold-out split. For SQLi, RF reduced false negatives from 23 to 12 while both models produced the same five false positives, which raised attack-class recall from 96.50% to 98.18%. For XSS, RF produced no misclassifications. The attack-class recall for SQLi is noticeably lower than the macro-averaged recall in Table 3. This shows why the per-class view matters for security: about 1.8% of the SQLi payloads in the hold-out set would still pass the RF-based firewall.

Table 4. Confusion Matrices and Attack-Class Recall on the Hold-Out Split

Task	Model	TN	FP	FN	TP	Attack recall
SQLi	Decision Tree	3,047	5	23	635	0.9650
SQLi	Random Forest	3,047	5	12	646	0.9818
XSS	Decision Tree	3,052	0	2	352	0.9944
XSS	Random Forest	3,052	0	0	354	1.0000

Table 4 makes the security consequence of these scores concrete. In the SQLi task, both models wrongly blocked the same five benign requests (FP = 5), so they are equally strict toward legitimate users; the entire difference lies in the false-negative column, where DT allowed 23 SQLi payloads through and RF allowed 12. RF therefore nearly halved the number of missed SQLi attacks without increasing false alarms. In the XSS task, DT missed two payloads and RF missed none, and neither model blocked any benign request. The table also shows how macro-averaging masks the weaker class: for DT on SQLi, the benign class reaches a recall of 0.9984 (3,047 of 3,052), which raises the macro-averaged recall to 0.9817 even though only 0.9650 of the attacks were detected. For this reason, attack-class recall rather than macro-averaged recall is used as the primary model selection criterion in this study.

Layer 2: Integrated Middleware Validation

Both Postman suites passed all assertions, with a 100% pass rate and a mean response time of about 24 ms per request (Figure 4). This response time covers the full HTTP round trip through the Flask middleware on localhost, so it is larger than the model-only latency reported in the next subsection. Malicious requests were intercepted before they reached the application server and received a generic block page that withholds technical detail, while benign requests were served without disruption. Representative SQLi payloads, namely the OR-based authentication bypass *' OR '1'='1*, the UNION-based extraction *' UNION SELECT username, password FROM users*--, and the destructive statement *'; DROP TABLE users;*--, were blocked, whereas a numeric identifier (*12345*) and a valid email address (*user@example.com*) were allowed.

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

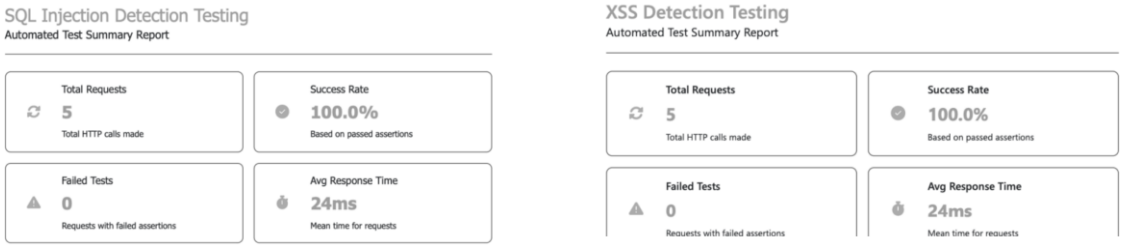


Figure 4. Automated black-box test summaries in Postman: (a) SQL Injection suite, (b) XSS suite

Figure 4 summarizes the automated Postman runs for the two test suites, with panel (a) showing the SQLi suite and panel (b) the XSS suite. Each panel reports four indicators: the total number of HTTP calls (5), the success rate based on passed assertions (100%), the number of failed tests (0), and the average response time (24 ms). A passed assertion means that the WAF returned the expected outcome for the request, namely a block page for a malicious payload and a normal application response for a benign input. The identical results of the two suites indicate that the integrated middleware applies the same decision logic to both attack types and that the response time is not noticeably affected by the type of attack being classified. Because each suite contains only five requests, Figure 4 should be interpreted as functional verification that the middleware is correctly integrated into the application rather than as a statistical estimate of detection performance, which is provided by Layers 1 and 3.

Layer 3: Robustness Stress Test

On the harder 500-request set, the scores remained high but were no longer perfect (Figure 5, Figure 6, and Table 5). RF correctly handled 496 of 500 requests (99.2%). It blocked all 400 attacks but wrongly blocked four benign inputs. DT correctly handled 494 requests (98.8%), with two false positives and four obfuscated SQLi payloads that evaded detection. Mean per-request latency was about 0.7 ms for DT and 5.5 ms for RF, with a 95th percentile of 6.7 ms for RF. The near-tie observed in Layer 1 therefore turned into a clear difference in missed attacks, four for DT versus none for RF.

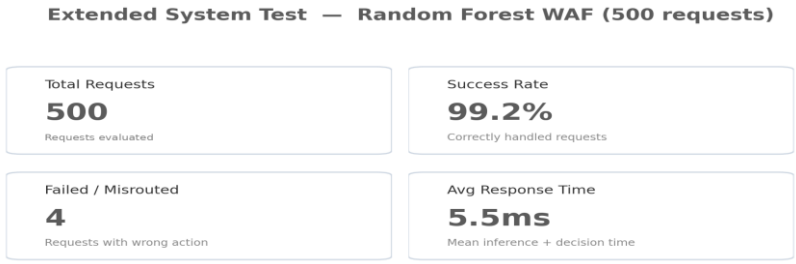


Figure 5. Extended 500-request test summary for the Random Forest WAF

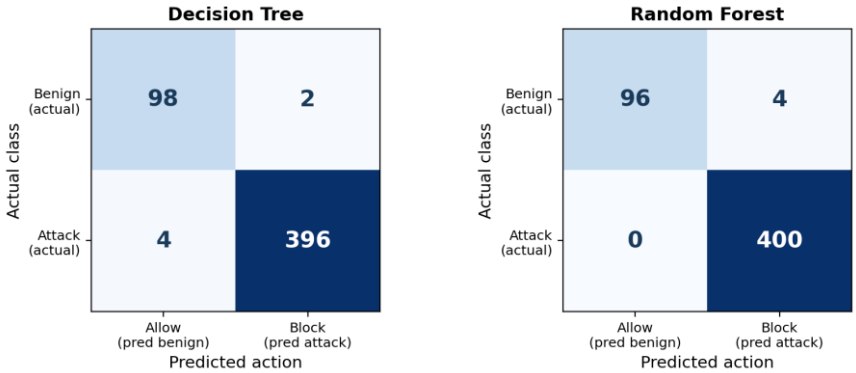


Figure 6. Confusion matrices of the extended 500-request test for Decision Tree and Random Forest

Table 5. Per-Class Outcomes of the Extended 500-Request Test

Model	SQLi blocked	XSS blocked	Benign allowed	Accuracy	Mean latency
Decision Tree	196 / 200	200 / 200	98 / 100	0.9880	0.7 ms
Random Forest	200 / 200	200 / 200	96 / 100	0.9920	5.5 ms

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Brilliance
Research of Artificial Intelligence

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

Table 5 breaks the stress-test result down by request class and shows where the errors of each model occur. Both models blocked all 200 XSS requests, including the 80 obfuscated ones, so the XSS task did not distinguish them. The difference lies entirely in the SQLi and benign columns and runs in opposite directions: DT missed four SQLi payloads (196/200) but wrongly blocked only two benign requests (98/100), whereas RF blocked every SQLi payload (200/200) but wrongly blocked four benign requests (96/100). The two models thus trade attack containment against user convenience, and the accuracy column (0.9880 versus 0.9920) understates this difference because it counts a missed attack and a wrongly blocked voter as equal errors. The latency column shows the cost of RF: its mean of 5.5 ms is roughly eight times that of DT (0.7 ms), since each request is evaluated by 100 trees instead of one, but both values remain small relative to the 24 ms HTTP round trip measured in Layer 2. Under the security-first criterion defined in the Method section, RF is preferred because it is the only model that missed no attack while keeping latency acceptable for interactive use.

DISCUSSION

Interpreting Near-Perfect Accuracy

The hold-out accuracy, in particular the 100% achieved by RF on XSS, should not be read as field performance. Very high accuracy on high-dimensional character features is consistent with the model memorizing distinctive substrings of the training payloads, especially when the data come from curated payload lists with limited diversity. Our corpus shares this characteristic: the malicious samples are explicit attack strings rather than obfuscated or adversarially modified traffic, so the classes are highly separable by construction. The scores in Tables 3 and 4 are therefore best read as an upper bound under near-ideal conditions. Repeated splitting of the same corpus would not remove this bias, because every partition is drawn from the same curated distribution; this is why the independent Layer 3 test is needed. Studies that evaluated WAFs on real-server or live traffic reported lower accuracy than on benchmarks, for example 98.8% versus 99.6% (Shaheed & Kurdy, 2022) and 89.34% in a live layered setting (Dawadi et al., 2023). ML-based firewalls also remain vulnerable to deliberately crafted evasive payloads (Floris, 2025).

Offline Accuracy Versus Operational Ranking

The central finding concerns RQ2. Offline accuracy suggested that DT and RF were nearly interchangeable, yet the stress test separated them on the metric that matters most for security, namely missed attacks. Model selection based on Layer 1 alone could therefore have chosen DT for its faster inference and accepted a weakness that only appears under obfuscation. This result also differs from Ramadhan and Ashari (2024) (Ramadhan & Ashari, 2024), who found DT slightly more accurate on binary network anomaly data, which suggests that the ranking of these classifiers depends on the threat setting and, in particular, on whether obfuscated inputs are part of the evaluation.

The stress test also exposes a trade-off that the hold-out scores hide. RF favored security by letting no attack through but was slightly more aggressive toward ambiguous benign traffic, whereas DT was gentler on benign users but missed some evasive attacks. A plausible explanation lies in the feature space. Character-level TF-IDF produces a high-dimensional sparse representation in which a single attack signature may appear in many lexical variations. A single decision tree tends to build highly specific boundaries that are sensitive to such variations, which makes it more vulnerable to obfuscation. RF aggregates many trees trained on different bootstrap samples and feature subsets, which reduces variance and improves robustness against lexical manipulation such as URL encoding, mixed-case injection, and inline comments.

In operational environments, preventing attack bypasses is generally more important than avoiding occasional false positives, because a single undetected attack may compromise the integrity of an election. The latency cost of this choice is small: a mean of 5.5 ms per request for RF is negligible compared with the 24 ms HTTP round trip and is acceptable for interactive use. These results support selecting RF for deployment, while confirming that obfuscation and look-alike benign inputs introduce genuine errors that offline accuracy does not reveal.

Comparison with Previous Studies

Table 6 compares the deployed RF-based WAF with representative recent ML- and DL-based WAF studies. The comparison is indicative rather than controlled, because the cited works use different datasets, attack scopes, and evaluation protocols; it positions the present results within the literature and does not claim a like-for-like ranking.

Table 6. Indicative Comparison with Recent ML- and DL-Based WAF Studies

Study	Method	Reported accuracy	Deployment
Shaheed & Kurdy (2022)	Feature engineering + classical ML	99.6% (benchmark); 98.8% (real server)	Offline + reverse-proxy concept
Dawadi et al. (2023)	Layered LSTM (DDoS; XSS+SQLi)	89.34% (XSS+SQLi layer)	Live evaluation
Tadhani et al. (2024)	Hybrid CNN-LSTM	99.84% (payload); 99.23% (testbed)	Offline
This study	Char-level TF-IDF + Chi-Square + RF	99.54% (SQLi), 100% (XSS) hold-out; 99.2% extended test	In-line middleware; three-layer protocol

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

The proposed model attains accuracy comparable to recent deep-learning approaches while using a much lighter classical pipeline. Unlike offline studies such as Bakır (2025) and Tadhani et al. (2024), it is exercised as an in-line middleware component inside a running application. Its hold-out accuracy, however, was obtained on a curated and highly separable corpus and should be interpreted as discussed above (Bakır, 2025; Tadhani et al., 2024) .

Research Implications

The findings indicate that classifier performance alone is insufficient for assessing practical security effectiveness, because deployment introduces additional constraints, including inference latency, request interception logic, logging overhead, and the trade-off between false positives and attack containment. A model with marginally higher offline accuracy does not necessarily provide better operational security if it introduces unacceptable latency or fails on obfuscated payloads. Deployment-aware evaluation should therefore be treated as an additional dimension of ML-WAF research. The three-layer protocol proposed here, consisting of offline classification assessment, integrated middleware validation, and robustness stress testing, together with the security-first selection criterion, offers future ML-WAF studies a reusable way to bridge the gap between laboratory performance and real-world deployment.

Limitations

Several limitations threaten the validity of these results. First, the training corpus is curated and highly separable, so the hold-out scores overstate the expected field performance. Second, the end-to-end tests ran on localhost with only five requests per Postman suite, and load, concurrency, and production traffic were not evaluated. Third, the extended robustness test used a reconstructed pipeline and a 500-request set built by the authors, and its obfuscation covered only case randomization, inline comments, and URL encoding rather than adaptive adversarial attacks. Fourth, Layer 1 used a single stratified hold-out split rather than repeated cross-validation, and only two attack classes and two classifiers were compared.

CONCLUSION

This study evaluated an ML-based WAF integrated end-to-end into an academic e-voting application using a three-layer deployment-aware protocol. For RQ1, the integrated WAF enforced allow, block, and log actions correctly under black-box testing, with a full HTTP round trip of about 24 ms and a model inference latency of 5.5 ms for RF, which is acceptable for interactive use. For RQ2, offline accuracy did not reveal the operational difference between the classifiers: DT and RF differed by less than 0.3 percentage points on the hold-out split, yet under obfuscated traffic DT missed four SQLi attacks while RF missed none. For RQ3, RF is the preferable deployment choice under a security-first criterion, accepting four false positives in 500 requests in exchange for zero missed attacks. The contribution is therefore methodological as well as practical: deployment-aware testing changes the evidence on which an ML-WAF model is selected, and the proposed protocol can be reused for other institutional web applications. Because the hold-out accuracy was obtained on a curated corpus in a localhost setting, it should be regarded as an upper bound. Future work should evaluate the system on external and benchmark datasets (Crespo-Martínez et al., 2023; Erskine, 2025), include adaptive adversarial payloads, extend coverage to attacks such as command injection and directory traversal, and conduct load testing under realistic production traffic.

ACKNOWLEDGMENT

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors. The processed dataset and source code supporting the findings are available from the corresponding author upon reasonable request.

REFERENCES

Alhamyani, R., & Alshammari, M. (2024). Machine Learning-Driven Detection of Cross-Site Scripting Attacks. *Information*, *15*(7), 420. https://doi.org/10.3390/info15070420

Asan Nainar, M. M., & Sibikarthik, B. K. (2025). ML-Enhanced Behavioral and Anomaly Detection Web Application Firewall. *International Journal of Innovative Research in Technology (IJIRT)*, *11*(12). https://ijirt.org/publishedpaper/IJIRT178127_PAPER.pdf

Aydos, M., Aldan, Ç., Coşkun, E., & Soydan, A. (2022). Security testing of web applications: A systematic mapping of the literature. *Journal of King Saud University - Computer and Information Sciences*, *34*(9), 6775–6792. https://doi.org/10.1016/j.jksuci.2021.09.018

Babaey, V., & Ravindran, A. (2024). GenSQLi: A Generative Artificial Intelligence Framework for Automatically Securing Web Application Firewalls Against Structured Query Language Injection Attacks. *Future Internet*, *17*(1), 8. https://doi.org/10.3390/fi17010008

Bakır, R. (2025). UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques. *Arabian Journal for Science and Engineering*, *50*(19), 15591–15604.

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

E-ISSN : 2807-9035
Volume 6, **Number** 4, November 2026
https://doi.org/10.47709/brilliance.v6i4.9731

https://doi.org/10.1007/s13369-024-09916-4

Casmiry, E., Mduma, N., & Sinde, R. (2025). Enhanced SQL injection detection using chi-square feature selection and machine learning classifiers. *Frontiers in Big Data*, *8*, 1686479. https://doi.org/10.3389/fdata.2025.1686479

Crespo-Martínez, I. S., Campazas-Vega, A., Guerrero-Higueras, Á. M., Riego-DelCastillo, V., Álvarez-Aparicio, C., & Fernández-Llamas, C. (2023). SQL injection attack detection in network flow data. *Computers & Security*, *127*, 103093. https://doi.org/10.1016/j.cose.2023.103093

Dawadi, B., Adhikari, B., & Srivastava, D. (2023). Deep Learning Technique-Enabled Web Application Firewall for the Detection of Web Attacks. *Sensors*, *23*(4), 2073. https://doi.org/10.3390/s23042073

Erskine, S. K. (2025). Real-time large-scale intrusion detection and prevention system (IDPS) CICIoT dataset traffic assessment based on deep learning. *Applied System Innovation*, *8*(2), 52. https://doi.org/10.3390/asi8020052

Floris, G. (2025). ModSec-AdvLearn: Countering adversarial SQL injections with robust machine learning. *IEEE Transactions on Information Forensics and Security*, *20*, 6693–6705. https://doi.org/10.1109/TIFS.2025.3583234

Irfa'issurur, M., & Josaphat, B. P. (2025). Machine Learning for Cybersecurity: Web Attack Detection (Brute Force, XSS, SQL Injection). *InPrime: Indonesian Journal of Pure and Applied Mathematics*, *7*(1), 1–15. https://doi.org/10.15408/inprime.v7i1.41025

Kepuska, K., & Tomašević, M. (2024). A lightweight framework for cyber risk management in Western Balkan higher education institutions. *PeerJ Computer Science*, *10*, e1958. https://doi.org/10.7717/peerj-cs.1958

Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2023). MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Computers & Security*, *124*, 103015. https://doi.org/10.1016/j.cose.2022.103015

Mani, K., & Shenoy, A. K. B. (2025). Machine learning models in web applications: A comprehensive review. *ICT Express*, *11*(6), 1110–1119. https://doi.org/10.1016/j.icte.2025.09.001

Nair, S. S. (2024). Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defense. *Journal of Computer Science and Technology Studies*, *6*(1), 76–93. https://doi.org/10.32996/jcsts.2024.6.1.9

Paul, A., Sharma, V., & Olukoya, O. (2024). SQL injection attack: Detection, prioritization & prevention. *Journal of Information Security and Applications*, *85*, 103871. https://doi.org/10.1016/j.jisa.2024.103871

Qin, Q., Li, Y., Mi, Y., Shen, J., Wu, K., & Wang, Z. (2024). Detecting XSS with Random Forest and multi-channel feature extraction. *Computers, Materials & Continua*, *80*(1), 843–874. https://doi.org/10.32604/cmc.2024.051769

Rahayu, A., Yulyanti, E., & Ghalib, M. (2025). Systematic Literature Review: SQL Injection Detection Vulnerability Using Machine Learning. *JURNAL MEDIA INFOTAMA*, *21*(1), 15–20. https://doi.org/10.37676/jmi.v21i1.6702

Ramadhan, R. F., & Ashari, W. M. (2024). Performance comparison of Random Forest and Decision Tree algorithms for anomaly detection in networks. *Journal of Applied Informatics and Computing (JAIC)*, *8*(2), 367–375.

Rodriguez Gavilanes, N. M., Loor Mendoza, B. D., & Llerena Guevara, L. A. (2025). Evaluación y mitigación de inyecciones SQL en aplicaciones web: Desarrollo de un prototipo. *Latin-American Journal of Computing*, *12*(2), 13–25. https://doi.org/10.33333/lajc.vol12n2.01

Rosca, C.-M., Stancu, A., & Popescu, C. (2025). Machine Learning Models for SQL Injection Detection. *Electronics*, *14*(17), 3420. https://doi.org/10.3390/electronics14173420

Santi, P. A. D. A., Afwani, R., Albar, M. A., Anjarwani, S. E., & Mardiansyah, A. Z. (2022). *Black box testing with equivalence partitioning and boundary value analysis methods*. 207–219. https://doi.org/10.2991/978-94-6463-084-8_19

Shaheed, A., & Kurdy, M. H. D. B. (2022). Web Application Firewall Using Machine Learning and Features Engineering. *Security and Communication Networks*, *2022*, 1–14. https://doi.org/10.1155/2022/5280158

Sun, H., Du, Y., & Li, Q. (2023). Deep learning-based detection technology for SQL injection research and implementation. *Applied Sciences*, *13*(16), 9466. https://doi.org/10.3390/app13169466

Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S., & El-Shafai, W. (2024). Securing web applications against XSS and SQLi attacks using a novel deep learning approach. *Scientific Reports*, *14*(1), 1803. https://doi.org/10.1038/s41598-023-48845-4

Younas, F., Raza, A., Thalji, N., Abualigah, L., Zitar, R. A., & Jia, H. (2024). An efficient artificial intelligence approach for early detection of cross-site scripting attacks. *Decision Analytics Journal*, *11*, 100466. https://doi.org/10.1016/j.dajour.2024.100466

This is an Creative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.