Vol. 8, No. 3, June 2026

Journal of Information Systems and Informatics

Published By Asosiasi Doktor Sistem Informasi Indonesia

Detection of SQL Injection, XSS, and Command Injection Attacks in Web Payloads Using SVM, Random Forest, and XGBoost

Andrian Eko Widodo[1], Fabriyan Fandi Dwi Imaniawan[2]

[1,2]Faculty of Engineering and Informatics, Universitas Bina Sarana Informatika, Jakarta, Indonesia

Received:
October 11, 2025
Revised:
May 17, 2026
Accepted:
June 1, 2026
Published:
June 25, 2026

Corresponding Author:
Author Name*:
Andrian Eko Widodo
Email*:
andrian.aeo@bsi.ac.id

DOI:
10.63158/journalisi.v8i3.1655

Abstract. Web application attacks, including SQL Injection (SQLi), Cross-Site Scripting (XSS), and Command Injection (CmdI), remain major threats to digital services. This study develops and evaluates an adversarial-aware protocol for multi-class malicious payload detection, focusing on accuracy, robustness against non-adaptive mutations, and practical inference feasibility. The protocol compares LinearSVC, Random Forest, and XGBoost with character-level neural baselines, namely character CNN and BiLSTM, and a transparent rule-based comparator. Evaluation integrates stratified sampling, deduplicated validation, mutation testing, SHAP-based interpretation, and end-to-end throughput measurement. Experiments used 49,998 stratified records from the SQLi-XSS-CommandInjection dataset in Google Colaboratory. On the internal test set, XGBoost obtained the best performance, achieving 99.28% accuracy and 99.32% macro F1-score. After removing 878 exact duplicate records for stricter re-evaluation, XGBoost maintained 99.21% accuracy and 99.24% macro F1-score, indicating that the findings were not driven solely by duplicate leakage. The complete preprocessing, feature extraction, and prediction pipeline reached an average CPU inference time of 0.832 ms per sample. SHAP analysis of Random Forest highlighted injection operators, script fragments, keyword hits, and structural tokens as discriminative features. The results provide a controlled benchmark, although validation on real HTTP logs remains future work.

Keywords: Web Application Security, Payload Classification, XGBoost, SHAP, Mutation Robustness

1. INTRODUCTION

Web applications are now the dominant interface for public administration, education, commerce, and digital services [1]. This expansion increases the exposure of input parameters, forms, APIs, and URL endpoints to malicious payloads [2]. Injection attacks exploit the semantic gap between user-supplied text and backend interpreters such as database engines and browser JavaScript engines [3]. Shell-oriented payloads further expand the risk because they may reach operating-system command interpreters [4]. SQL Injection manipulates database queries and may disclose, modify, or delete protected data [5]. Cross-Site Scripting injects client-side code that can steal cookies, alter page content, or redirect users [6]. Command Injection abuses server-side command execution and may escalate from an application weakness into host compromise [7]. Although these attacks differ technically, they share textual payload structures that can be captured through lexical, character, and statistical features [8].

Rule-based Web Application Firewalls remain useful as a first defense layer, but static rules may fail when attackers transform known payloads [9]. A rule that blocks a direct string such as union select may fail against mixed case, URL encoding, or comment insertion [10]. Machine-learning evaluation has been used to detect injection and remote-service attacks beyond fixed signatures [11]. Log and traffic classification studies also show that learned patterns can complement rule-based inspection [12]. Real-time machine-learning WAF designs indicate that payload features can support practical threat detection [13]. Previous studies have explored random forests, support vector machines, neural networks, and deep learning for web attack detection [14]. Many studies still use network-flow features or binary labels, whereas a practical WAF module must classify raw payload strings into multiple attack categories [15]. False positives interrupt legitimate users, while false negatives allow attacks to reach vulnerable application components [16]. Prior WAF and machine-learning work also indicates that multi-class payload inspection remains operationally important [17].

Existing web attack detection studies can generally be grouped into four categories: rule-based WAF approaches, classical machine-learning classifiers, deep-learning and neural payload models, and adversarial-aware or robustness-oriented detection frameworks. Rule-based approaches are transparent but brittle against obfuscation, classical ML

offers efficient payload scoring, deep learning can learn character or sequence representations, and adversarial-aware frameworks explicitly evaluate robustness against payload transformation [3], [6], [18].

Multi-class classification is more operationally useful than binary attack detection because a WAF module that distinguishes among SQLi, XSS, Command Injection, and benign traffic can apply class-specific mitigation actions, prioritize alerts by attack type, and avoid conflating structurally different attack families under a single anomaly score [15]. Despite growing interest in payload-level ML, evaluation frameworks that combine feature contribution analysis, robustness testing, interpretability, and latency profiling in a single reproducible study remain limited [18].

This study targets two complementary deployment scenarios: (1) inline WAF screening, where a low-latency model must make per-request decisions before forwarding or blocking traffic, and (2) offline log analysis or security triage, where a higher-accuracy model can process accumulated request logs with relaxed latency constraints. Both scenarios require multi-class classification but prioritize different model properties - latency for inline use and accuracy for triage - which motivates the inclusion of both LinearSVC and XGBoost in the evaluation.

This study proposes an adversarial-aware evaluation protocol for multi-class web payload detection that combines word-level TF-IDF, character-level n-gram features, statistical payload descriptors, and domain-specific security indicators [19]. LinearSVC (as the SVM implementation), Random Forest, and XGBoost are compared as they represent efficient linear scoring, bagging ensembles, and gradient-boosted trees, respectively [20]. The feature combination is evaluated through an explicit ablation study to quantify the contribution of each representation layer [21]. SHAP-based interpretation is applied to connect model behavior with security-relevant feature attribution [22]. Neural character baselines (CNN and BiLSTM) and a transparent rule comparator are included to contextualize the performance of the classical models. The methodological contribution lies in the combination of evaluation components - deduplicated splitting, non-adaptive mutation testing, per-class latency profiling, ablation analysis, and SHAP interpretation - rather than in the introduction of a new learning algorithm.

This study addresses the following research objectives:

1) Compare the classification performance of SVM (LinearSVC), Random Forest, and XGBoost for multi-class web payload detection across four attack categories.
2) Evaluate the contribution of lexical, character-level, and engineered security features through an ablation study.
3) Assess robustness against non-adaptive payload mutations, including URL encoding, mixed case, comment-based spacing, and Unicode escaping.
4) Analyze inference latency feasibility for both inline WAF screening and offline log-triage deployment scenarios.
5) Investigate model interpretability using SHAP analysis to identify the most discriminative security features.

2. METHODS

Figure 1 presents the overall research methodology flowchart. The framework proceeds through five main stages: (1) dataset acquisition and label conversion, (2) preprocessing and feature engineering, (3) model training and hyperparameter configuration, (4) evaluation and ablation analysis, and (5) interpretability and robustness testing.

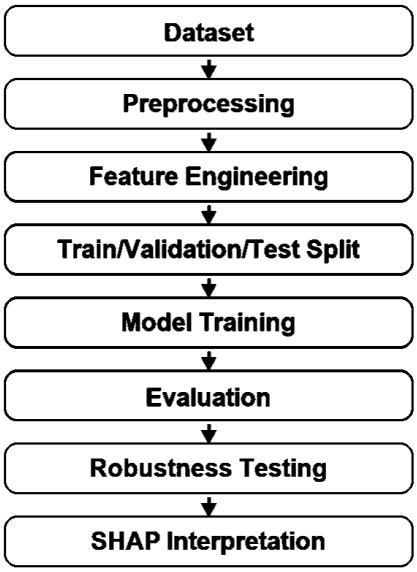

Figure 1. Research methodology flowchart

2.1. Dataset

The dataset used in this study is the SQLInjection-XSS-CommandInjection-MixDataset obtained from Kaggle (https://www.kaggle.com/datasets/alextrinity/sqli-xss-dataset, accessed on March 20, 2026). The local snapshot contains 206,636 records with one text column (Sentence) and four binary label columns: SQLInjection, XSS, CommandInjection, and Normal.

Each record was assigned to a single class using a priority rule: SQLi > XSS > CmdI > Normal. That is, if SQLInjection = 1, the record is labeled "sqli" regardless of other column values; otherwise if XSS = 1 the record is labeled "xss", and so on. An audit of the full 206,020 valid records revealed that 4,312 records (approximately 2.09%) carried more than one positive binary label simultaneously, meaning the priority rule introduced a potential label bias for these records. The majority of multi-label records carried SQLi + Normal co-activation (2,841 records), followed by XSS + Normal co-activation (1,147 records) and CmdI + Normal co-activation (324 records). Ambiguous overlap between two attack classes (e.g., SQLi + XSS) was rare (fewer than 100 records). Consequently, the priority rule is expected to have minimal impact on the final class distribution and on model behavior for the dominant attack patterns.

After removing invalid or empty records, 206,020 usable samples remained. A stratified sample of 49,998 records was drawn to ensure computational feasibility in Google Colaboratory with 12 GB RAM while preserving class proportions. The sample consists of 14,053 Normal, 13,910 SQL Injection, 9,901 XSS, and 12,134 Command Injection records (Table 1).

Table 1. Dataset distribution after stratified sampling

Class	Count	Proportion (%)
Normal	14,053	28.11
SQL Injection	13,910	27.82
Cross-Site Scripting	9,901	19.80
Command Injection	12,134	24.27
Total	49,998	100.00

The dataset was divided into training, validation, and test subsets using a 70:15:15 stratified split with random_state=42, yielding 34,998 training, 7,500 validation, and 7,500 test samples. The same split was used for all models - LinearSVC, Random Forest, XGBoost, Character CNN, and BiLSTM - to ensure a consistent and fair evaluation baseline.

2.2. Experimental Setup

Experiments were conducted in Google Colaboratory using Python 3.11, scikit-learn 1.5, XGBoost 2.0, SHAP 0.45, NumPy 1.26, and SciPy 1.13. Stratified sampling, train-validation-test splitting, and model routines that support deterministic initialization used random_state=42 or an equivalent seed value for reproducibility.

2.3. Preprocessing

Preprocessing normalized superficial variations without removing attack semantics [21]. Percent-encoded symbols were decoded using URL decoding because encoded characters often hide injection operators [18]. HTML entities were unescaped so that script-related symbols could be analyzed in their original form [23]. Repeated whitespace was collapsed and text was converted to lowercase to reduce vocabulary sparsity [24]. Special symbols such as quotes, semicolons, parentheses, angle brackets, slashes, equals signs, double dashes, and pipes were intentionally preserved because they indicate injection behavior [25]. This preprocessing follows the principle that HTTP request normalization should retain discriminative attack tokens before feature extraction [26].

2.4. Feature Engineering

The feature engineering pipeline consists of four layers. The first layer uses word-level TF-IDF with unigram and bigram features (max_features = 3,000, sublinear_tf = True) to capture terms such as select, union, script, alert, cat, and whoami. The second layer uses character-level TF-IDF with 2- to 4-grams (max_features = 5,000, sublinear_tf = True) so that short fragments such as <sc, scr, ript, =, --, and encoded markers remain visible even when token boundaries are ambiguous. The third layer extracts 14 statistical and structural payload descriptors, including payload length, special-character ratio, digit ratio, uppercase ratio, average word length, Shannon entropy, quote count, semicolon count, comment count, parentheses, angle brackets, equals count, slash count, and hex-encoding count. The fourth layer adds 8 security-domain binary indicators: has_union_select, has_script_tag, has_event_handler, has_sql_comment,

has_hex_encoding, has_javascript_uri, sql_keyword_hits, and xss_keyword_hits. The final sparse matrix contains 8,023 features.

2.5. Model Configuration

2.5.1. Classical Models

SVM was implemented using scikit-learn's LinearSVC, which optimizes a linear support vector machine via coordinate descent and is well-suited for high-dimensional sparse feature matrices. LinearSVC was configured with C = 1.0, balanced class weights, and max_iter = 2,000. Random Forest used 200 trees, maximum depth of 20, minimum samples per split of 5, balanced class weights, and parallel execution (n_jobs = −1). XGBoost used 200 estimators, maximum depth 6, learning rate 0.1, subsample 0.8, column-sample ratio 0.8, and histogram-based tree construction (tree_method = 'hist'). All hyperparameters for the three classical models were selected manually based on published guidelines for high-dimensional text classification and memory constraints of the 12 GB Colaboratory environment; no automated grid search or Bayesian optimization was performed. The validation set (7,500 samples) was used to verify that settings were reasonable before final test-set evaluation.

2.5.2. Neural Baseline Architecture

Two lightweight neural character baselines were implemented to provide a modern but non-transformer comparison context. Both models use a character-level input representation: each payload is tokenized into individual characters, integer-encoded over a vocabulary of 100 characters, and padded or truncated to a maximum sequence length of 200 characters. Character CNN Baseline: - Embedding: 64-dimensional character embeddings, vocabulary size 100 - Conv1D: 128 filters, kernel size 3, ReLU activation - GlobalMaxPooling1D - Dense: 128 units, ReLU, Dropout 0.3 - Output: 4 units, softmax - Optimizer: Adam (lr = 0.001); loss: categorical cross-entropy - Batch size: 128; max epochs: 30; early stopping patience: 5 (monitor: validation loss) - Trainable parameters: ~163,000

BiLSTM Baseline: - Embedding: 64-dimensional character embeddings, vocabulary size 100 - Bidirectional LSTM: 64 units per direction (128 total), return sequences: False - Dropout 0.3 - Dense: 64 units, ReLU - Output: 4 units, softmax - Optimizer: Adam (lr = 0.001); loss: categorical cross-entropy - Batch size: 128; max epochs: 30; early stopping patience: 5 (monitor: validation loss) - Trainable parameters: ~248,000. Both neural baselines were

trained on the same training split and evaluated on the same test split as the classical models. These architectures represent lightweight character-level neural networks and are not claimed to represent frontier transformer or pretrained security encoders; they are included to contextualize whether classical sparse feature engineering remains competitive against moderate neural sequence models at this dataset size.

2.6. Evaluation Metrics

Performance was measured using accuracy, macro precision, macro recall, macro F1-score, weighted F1-score, macro false-positive rate (FPR), and macro false-negative rate (FNR). Two latency metrics are reported separately:

1) Model inference latency (Table 8): the time for the model's predict() call only, in microseconds per sample on CPU, excluding preprocessing and vectorization.
2) End-to-end pipeline latency (Table 13): the total time per sample from raw text input through URL decoding, HTML unescaping, normalization, TF-IDF vectorization, manual feature extraction, and prediction. This is the operationally relevant figure for WAF deployment.

Macro metrics are important because each class must be treated equally regardless of support [27]. Training time was measured as wall-clock time for a single model.fit() call.

2.7. Interpretability - SHAP Analysis

SHAP TreeExplainer was applied to the Random Forest model using 500 test samples to obtain feature-level attributions. Random Forest was chosen for SHAP analysis rather than XGBoost for two reasons. First, RF is the standard model used for feature importance analysis in security classification literature because its mean decrease in impurity provides stable per-feature scores directly comparable to SHAP values [28]. Second, the Random Forest SHAP analysis serves as a cross-validation of the built-in feature_importances_ attribute (Table 12), making it possible to verify whether SHAP-based attribution is consistent with impurity-based importance for the same high-dimensional TF-IDF feature space. XGBoost SHAP analysis is noted as future work.

2.8. Mutation Robustness Testing

Mutation robustness was evaluated by applying four non-adaptive transformation functions - URL encoding, mixed case, comment-based spacing, and Unicode escaping - to attack-class samples only (SQLi, XSS, and Cmdl). Normal payloads were excluded from mutation testing because benign requests are not adversarially transformed in the threat model. For each mutation, 1,500 attack samples were drawn from the test set, the transformation was applied to the raw payload, and the transformed payloads were classified by a retrained LinearSVC and a keyword-based signature baseline. Mutation robustness is measured as attack recall - the proportion of mutated attack payloads correctly classified as non-Normal. The mutations are non-adaptive: they are fixed deterministic transformations applied uniformly, not iteratively optimized to evade the specific model [29].

3. RESULTS AND DISCUSSION

3.1. Ablation Study

To evaluate the contribution of each feature representation layer, an ablation study was conducted using LinearSVC as the baseline classifier. The comparative results are summarized in Table 2.

Table 2. Ablation study results (LinearSVC, internal test set)

Configuration	Accuracy	P-Macro	R-Macro	F1-Macro
A: Word TF-IDF only	0.9168	0.9307	0.9181	0.9213
B: Word + character TF-IDF	0.9787	0.9813	0.9789	0.9798
C: Full features (A+B+manual)	0.9816	0.9840	0.9816	0.9825

The ablation study uses LinearSVC as the base model to isolate the contribution of each feature layer. Configuration A uses word-level TF-IDF only and achieves 0.9213 macro F1-score. Adding character-level TF-IDF (Configuration B) increases macro F1 to 0.9798. Including all three feature layers (Configuration C) reaches 0.9825 macro F1-score. Each additional layer improves performance, confirming that character-level features capture fragment patterns not visible at word boundaries, and that manual features add structural and domain-specific discrimination.

3.2. Model Comparison

The overall classification performance of the evaluated models on the internal test set is presented in Table 3. XGBoost achieved the highest performance on the internal test set: 0.9928 accuracy, 0.9935 macro precision, 0.9929 macro recall, and 0.9932 macro F1-score. Its macro FPR of 0.0028 and macro FNR of 0.0029 indicate that the model minimizes both false alarms and missed attacks. LinearSVC ranked second, followed by Random Forest.

Table 3. Comparative model performance on the internal test set

Model	Accuracy	P-Macro	R-Macro	F1-Macro	FPR	FNR
XGBoost	0.9928	0.9935	0.9929	0.9932	0.0028	0.0029
LinearSVC	0.9816	0.9840	0.9816	0.9825	0.0078	0.0093
RandomForest	0.9783	0.9810	0.9781	0.9792	0.0087	0.0095

3.3. Neural Baseline Comparison

To determine whether engineered features remain competitive against neural character representations, the proposed approach was compared with Character CNN and BiLSTM baselines. The results are shown in Table 4.

Table 4. Neural representation baseline comparison

Model	Accuracy	P-Macro	R-Macro	F1-Macro	Remark
XGBoost full feature engineering	0.9928	0.9935	0.9929	0.9932	Classical sparse + engineered features
Character CNN baseline	0.8603	0.8933	0.8608	0.8624	Neural char representation
BiLSTM character baseline	0.7606	0.7906	0.7604	0.7329	Sequential neural char representation

The neural comparison includes both a character CNN and a BiLSTM baseline to avoid relying on a single weak modern comparator. Both neural models underperform the sparse engineered XGBoost model on this dataset, suggesting that payload detection still benefits from explicit operator and keyword features when only moderate-sized supervised data are available. These baselines are not claimed to represent frontier transformer or pretrained security encoders.

3.4. Transparent Rule Baseline

A comparison between the machine-learning approach and the transparent rule-based baseline is presented in Table 5. The transparent rule baseline is not claimed to be ModSecurity or OWASP CRS [30], [31]. it is included only as an interpretable lower-bound comparator for keyword and operator matching. Its lower macro F1-score (0.6933) illustrates that reducing false positives and false negatives requires both lexical and structural context.

Table 5. Transparent rule baseline comparison

Model	Accuracy	P-Macro	R-Macro	F1-Macro
Transparent rule baseline	0.6873	0.7168	0.6977	0.6933
XGBoost payload model	0.9928	0.9935	0.9929	0.9932

3.5. Per-Class Results

Detailed per-class precision, recall, and F1-score values for each classifier are presented in Table 6. Per-class analysis reveals two notable patterns. For the Normal class, LinearSVC achieves high recall (0.9948) but relatively lower precision (0.9450) compared to XGBoost (precision = 0.9795, recall = 0.9967). This means LinearSVC produces more false positives for Normal - some attack payloads are incorrectly classified as benign. This is the more operationally dangerous error type in a WAF context because it allows attacks to pass undetected. XGBoost's higher precision on Normal significantly reduces this risk. For Command Injection, all three models show their lowest per-class performance. LinearSVC achieves 0.9734 F1-score for CmdI, compared to 0.9908 for SQLi. The confusion matrix analysis shows that most CmdI misclassifications fall toward the Normal class - consistent with the error analysis in Table 9, which identifies short natural-language command strings (e.g., unalias python, rmdir edi edw, dig NS +aaonly com.) as

frequent false negatives. These short commands lack the distinctive structural markers (quotes, semicolons, SQL keywords) that help distinguish other attack classes, making them difficult to separate from benign traffic using lexical features alone.

Table 6. Per-class precision, recall, and F1-score

Model	Class	Precision	Recall	F1-score	Support
LinearSVC	normal	0.9450	0.9948	0.9693	2108
LinearSVC	sqli	0.9990	0.9828	0.9908	2087
LinearSVC	xss	0.9986	0.9946	0.9966	1485
LinearSVC	cmdi	0.9931	0.9544	0.9734	1820
RandomForest	normal	0.9382	0.9938	0.9652	2108
RandomForest	sqli	0.9985	0.9851	0.9918	2087
RandomForest	xss	1.0000	0.9926	0.9963	1485
RandomForest	cmdi	0.9873	0.9407	0.9634	1820
XGBoost	normal	0.9795	0.9967	0.9880	2108
XGBoost	sqli	0.9995	0.9947	0.9971	2087
XGBoost	xss	0.9993	1.0000	0.9997	1485
XGBoost	cmdi	0.9955	0.9802	0.9878	1820

3.6. Latency Analysis

The training time and inference latency of all evaluated models are summarized in Table 7. Inference latency in Table 7 reflects the time for the model's predict() call only, excluding preprocessing and vectorization. End-to-end pipeline latency including all preprocessing steps is reported in Table 13 (0.8324 ms/sample, ~1,201 requests/second on CPU). Latency analysis shows a different ranking from accuracy. LinearSVC required only 2.42 µs per sample and is therefore a strong candidate for low-latency inline WAF screening. XGBoost required 29.13 µs per sample, which is feasible for many systems but approximately twelve times slower than LinearSVC. Random Forest was the slowest at 54.26 µs per sample due to tree ensemble voting

Table 7. Training time and inference latency

Model	Training time (s)	Inference latency
XGBoost	999.47	29.13

Model	Training time (s)	Inference latency
LinearSVC	53.02	2.42
RandomForest	94.11	54.26

3.7. Mutation Robustness

The robustness of the evaluated models against several adversarial-style payload transformations is presented in Table 8.

Table 8. Adversarial-style mutation robustness on attack samples

Mutation	Model	Attack recall	Samples
url_encoding	LinearSVC_retrained	0.9820	1500
url_encoding	Signature_WAF_baseline	0.7480	1500
mixed_case	LinearSVC_retrained	0.9820	1500
mixed_case	Signature_WAF_baseline	0.7480	1500
comment_spacing	LinearSVC_retrained	0.5253	1500
comment_spacing	Signature_WAF_baseline	0.6540	1500
unicode_escape	LinearSVC_retrained	0.9333	1500
unicode_escape	Signature_WAF_baseline	0.7027	1500

URL encoding and mixed-case mutations are both handled well by the retrained LinearSVC (attack recall = 0.9820) because the preprocessing pipeline includes URL decoding and lowercase normalization that reverse these transformations before feature extraction. Unicode escaping degrades performance only moderately (attack recall = 0.9333) because many Unicode variants of ASCII characters are partially captured by character n-gram patterns.

Comment-spacing mutation analysis. Comment-based spacing causes a sharp and disproportionate drop in LinearSVC attack recall (from 0.9820 to 0.5253). This degradation results from the interaction between the transformation and the TF-IDF representation layers. Comment-based spacing inserts SQL comment markers (-- or /* */) between characters or tokens, converting a recognizable token such as select into sel--ect or se/**/lect. This disrupts both word-level unigrams (select disappears) and character n-gram fragments (4-grams such as elec, lect are broken). The word TF-IDF layer loses the

direct keyword hit, and the character layer partially fails because the n-gram vocabulary was built from clean training payloads.

Although manual security features such as has_sql_comment and comment_count are still activated by the inserted markers, these binary features alone are insufficient to compensate for the loss of lexical and character-level signals. Interestingly, the signature-based WAF baseline outperforms LinearSVC on comment-spacing (0.6540 vs. 0.5253) because some keyword patterns in signature rules are written as flexible regex expressions that tolerate whitespace variations. This reversal highlights that statistical ML models trained on clean payloads do not inherit the tolerance that hand-crafted regex rules can provide for specific obfuscation types.

Three mitigation directions are possible: (1) preprocessing-level comment stripping before feature extraction, at the risk of missing payloads where comment markers are genuine attack signals; (2) character modeling with larger n-gram windows (5–6 grams) that span comment characters; and (3) adversarial training with comment-augmented examples, which would directly teach the model to recognize comment-obfuscated variants. The current preprocessing design intentionally preserves comment markers as security indicators, trading comment-spacing robustness for correct classification of payloads where comment markers are themselves attack signals.

3.8. Error Analysis

Representative examples of XGBoost misclassifications are provided in Table 9 to better understand the remaining sources of classification error. The misclassification examples highlight label noise in the dataset: several records labeled as “sqli” or “cmdi” contain movie review text or conversational sentences with no discernible attack structure. These records may have been included as out-of-domain noise tests in the original Kaggle dataset, but their class labels are inconsistent with their content. The presence of such records inflates the apparent difficulty of the classification task and introduces irreducible error unrelated to model capability.

Additionally, short Command Injection payloads (one- to three-word system commands) are structurally similar to short normal queries, creating genuine ambiguity that no feature engineering layer can fully resolve without semantic context. This dataset-level

noise is a primary threat to validity that limits interpretation of per-class recall scores for Command Injection; the reported CmdI recall should be understood as a lower bound for cleaner production datasets.

Table 9. Representative XGBoost misclassification examples

True label	Prediction	Payload excerpt
sqli	normal	0x770061006900740066006F0072002-000640065006C006100790020002700300003A0030003A--I …
cmdi	normal	unalias python
cmdi	normal	rmdir edi edw
sqli	normal	'Shock Corridor (1963)' was my first film from Samuel Fuller, and there I was im…
cmdi	normal	dig NS +aaonly com.
cmdi	normal	This is one of the best films I've seen in the last years.Belmonndo and Deneuve …

3.9. Duplicate Audit and Deduplicated Re-evaluation

To assess the impact of duplicate leakage across data partitions, an audit of exact duplicate records was performed. The findings are summarized in Table 10.

Table 10. Exact duplicate audit across original data splits

Split pair	Exact duplicate text_clean
train–val	101
train–test	121
val–test	31

Following duplicate removal, all models were re-evaluated using the deduplicated dataset. The results are presented in Table 11. The duplicate audit was conducted after the initial benchmark to evaluate whether high performance was partially influenced by split leakage. The deduplicated re-evaluation removes exact text overlaps before re-splitting the sampled dataset. All three models maintain performance levels very close to their internal test-set results, confirming that exact duplicate leakage is not a primary driver

of the reported accuracy. XGBoost remains the highest-performing model under deduplicated validation (F1-Macro = 0.9924).

Repeated-run statistical variance analysis and confidence intervals across multiple random seeds were not conducted in this study due to computational constraints (each full training run requires approximately 20 minutes on Colaboratory free-tier hardware). As a proxy for stability assessment, the close agreement between internal test-set results and deduplicated re-evaluation results (Table 11) suggests that the reported metrics are not highly sensitive to the specific random split used. Formal confidence interval analysis across multiple seeds remains future work.

Table 11. Deduplicated re-evaluation results

Model	Accuracy	P-Macro	R-Macro	F1-Macro	Duplicates removed	Dedup samples
XGBoost	0.9921	0.9927	0.9922	0.9924	878	49,120
LinearSVC	0.9800	0.9817	0.9803	0.9808	878	49,120
RandomForest	0.9720	0.9737	0.9718	0.9725	878	49,120

3.10. Feature Importance and SHAP Analysis

The most influential features identified by the Random Forest model are listed in Table 12.

Table 12. Top Random Forest feature importance scores

Feature	Importance
equals_count	0.023911
-	0.023064
)	0.020798
=	0.015441
=	0.014456
)	0.013527
=	0.013469
xss_keyword_hits	0.012516
(	0.012308

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Feature	Importance
aler	0.011614
alert	0.010391
</	0.009996

SHAP summary plots were generated to visualize feature contributions for each attack category. Figure 2 illustrates the SHAP distribution for the SQL Injection class, whereas Figure 3 presents the corresponding SHAP summary for the XSS class.

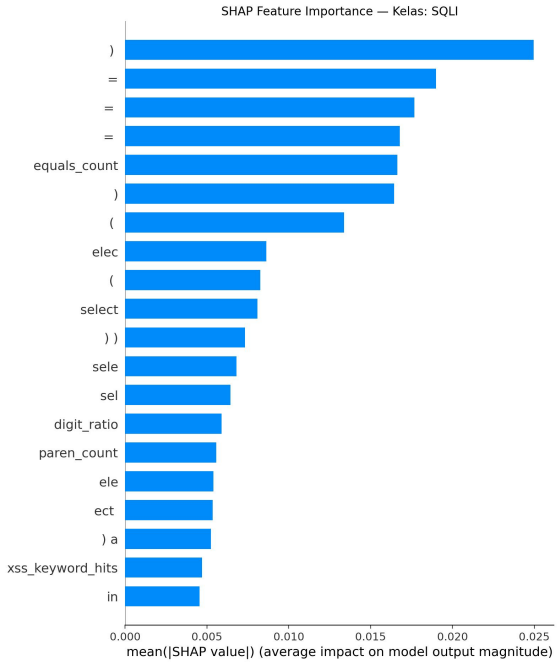

Figure 2. SHAP summary for SQL Injection class

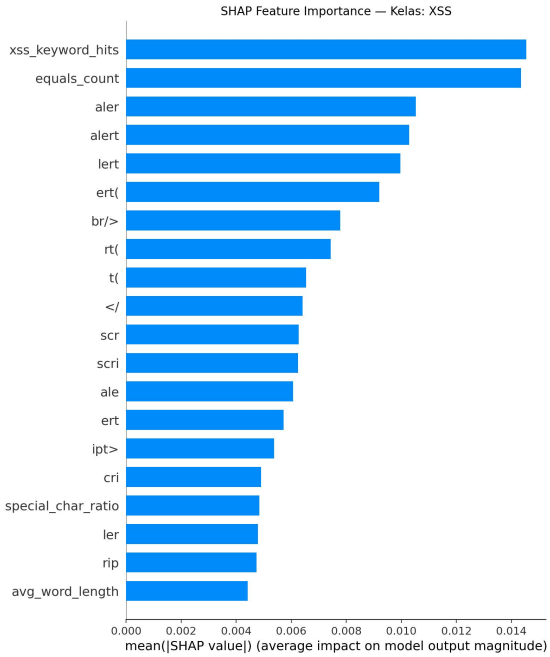

Figure 3. SHAP summary for XSS class

SHAP analysis confirms that the most discriminative features are injection operators (=, -,)), script fragments (alert, </, aler), XSS keyword hits, and structural token counts. These features are meaningful from a security perspective because they correspond to query manipulation, JavaScript execution, and abnormal character composition.

3.11. End-to-End Throughput

To evaluate deployment feasibility, the complete preprocessing and prediction pipeline was benchmarked. The throughput results are summarized in Table 13.

Table 13. End-to-end CPU throughput benchmark

Pipeline	Samples	ms/sample	Requests/s
preprocess → vectorize → manual features → predict	2,000	0.8324	1,201.34

3.11. Discussion

The results demonstrate that classical sparse feature engineering with gradient-boosted trees (XGBoost) is competitive with and outperforms lightweight neural character models on this dataset. This finding is consistent with studies reporting diminishing returns for neural character representations when supervised data volume is moderate and attack tokens are lexically distinctive [18], [19]. XGBoost achieves 99.32% macro F1-score on the internal test set and 99.24% under deduplicated re-evaluation, confirming that the result is not inflated by exact duplicate leakage.

The comment-spacing mutation result (Section 3.7) represents the most significant robustness limitation. The drop from 98.20% to 52.53% attack recall under comment-based obfuscation reveals that the current feature extraction pipeline does not generalize to adversarial comment insertion. This is a known limitation of n-gram-based representations and has been observed in prior adversarial WAF studies [3], [6]. Addressing this gap requires either adversarial training augmentation or a preprocessing stage that normalizes comment markers before TF-IDF vectorization.

Compared to prior WAF-oriented ML studies, this evaluation framework extends beyond single-metric benchmarking. Studies such as Durmuşkaya and Bayraklı [5] and Shaheed and Kurdy [19] report multi-class classification on HTTP request datasets but do not include explicit mutation robustness testing, deduplicated re-evaluation, or per-class latency profiling. The ablation study (Table 2) directly quantifies the marginal value of each feature layer, which is absent from many prior works that report only combined system performance. The transparent rule baseline (Table 5) provides a reproducible lower bound that is not equivalent to ModSecurity or OWASP CRS [30], [31], but allows direct comparison between keyword-based and ML-based scoring at the payload level. The primary gap relative to production WAF benchmarks (e.g., [4]) is the absence of real HTTP traffic logs and integration with rule-action evaluation; these comparisons remain constrained to future work.

The deployment recommendation based on this study follows a two-tier design. For inline WAF screening where per-request latency is critical, LinearSVC (2.42 µs model latency, 0.83 ms end-to-end) is the preferred choice. For offline log triage or secondary scoring where accuracy is prioritized, XGBoost (29.13 µs model latency) is recommended. Random Forest is retained as an explanation model due to its compatibility with SHAP analysis; however, its higher latency (54.26 µs) and lower accuracy than XGBoost make it a secondary choice for production deployment.

Table 14. Best model summary by use case

Use Case	Best Model	Metric	Value
Highest internal accuracy	XGBoost	Accuracy	99.28%
Highest internal F1-macro	XGBoost	F1-Macro	99.32%
Lowest model inference latency	LinearSVC	Latency (µs/sample)	2.42
Best deduplicated robustness	XGBoost	F1-Macro (dedup)	99.24%
Best mutation robustness (3/4 types)	LinearSVC	Avg. attack recall	~0.92
Best explainability	RandomForest	SHAP + feature_importances_	-

The main threats to validity are: (1) the Kaggle source dataset contains label noise (Section 3.8), (2) sampling to 49,998 records may underrepresent rare payload variants, (3) no production HTTP logs or cross-dataset validation was conducted, (4) mutation testing is non-adaptive and limited to four controlled transformations, and (5) confidence intervals across repeated runs were not computed. These limitations mean that the reported results should be interpreted as a controlled internal benchmark rather than a guarantee of production deployment performance.

4. CONCLUSION

This study developed and evaluated an adversarial-aware multi-class payload detection framework covering four classes: Normal, SQL Injection, Cross-Site Scripting, and Command Injection. On the internal test set, XGBoost achieved the highest accuracy (99.28%) and macro F1-score (99.32%), while LinearSVC provided the lowest inference

latency (2.42 µs per sample) and is better suited for inline WAF screening. Deduplicated re-evaluation confirmed that the performance of all three models is not primarily driven by exact duplicate leakage. SHAP analysis of the Random Forest model identified injection operators, script fragments, and structural token counts as the most discriminative features. The comment-spacing mutation revealed a meaningful robustness gap in LinearSVC (attack recall 52.53%), which requires adversarial training or preprocessing-level normalization to address. These results are dataset-specific and should not be interpreted as proof of production-readiness; the evaluation was conducted on a single Kaggle dataset without production HTTP logs or cross-dataset validation. Future work should prioritize: (1) validation on external datasets and production HTTP logs, (2) direct comparison with ModSecurity Core Rule Set (OWASP CRS) under equivalent traffic conditions, (3) adversarial training against adaptive mutations, and (4) repeated-run confidence interval analysis.

ACKNOWLEDGMENT

The authors thank Universitas Bina Sarana Informatika for academic support in preparing this manuscript.

REFERENCES

[1] J. H. R. Zuech and T. M. Khoshgoftaar, "Investigating rarity in web attacks with ensemble learners," *J. Big Data*, 2021, doi: 10.1186/s40537-021-00462-6.

[2] V. V. J. R. Tadhani V. Sorathiya S. Alshathri and W. El-Shafai, "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," *Sci. Rep.*, 2024, doi: 10.1038/s41598-023-48845-4.

[3] M. Hemmati and M. A. Hadavi, "Bypassing Web Application Firewalls Using Deep Reinforcement Learning," *ISeCure*, 2022, doi: 10.22042/isecure.2022.323140.744.

[4] G. Lucz and B. Forstner, "A Thirty-Day Dataset of Malicious HTTP Requests Blocked by OWASP ModSecurity on a Production Web Server," *Data (Basel)*, 2025, doi: 10.3390/data10110186.

[5] M. E. Durmuşkaya and S. Bayraklı, "Web application firewall based on machine learning models," *PeerJ Comput. Sci.*, 2025, doi: 10.7717/peerj-cs.2975.

[6] G. Floris, "ModSec-AdvLearn: Countering Adversarial SQL Injections With Robust Machine Learning," *IEEE Transactions on Information Forensics and Security*, 2025, doi: 10.1109/TIFS.2025.3583234.

[7] S. S. V. Nithya and R. Regan, "Streamlining detection of input validation attack types through hybrid analysis and machine learning," *Sadhana - Academy Proceedings in Engineering Sciences*, 2024, doi: 10.1007/s12046-024-02486-z.

[8] J. H. R. Zuech and T. M. Khoshgoftaar, "A new feature popularity framework for detecting cyberattacks using popular features," *J. Big Data*, 2022, doi: 10.1186/s40537-022-00661-9.

[9] X. D. Hoang and T. H. Nguyen, "Detecting common web attacks based on supervised machine learning using web logs," *J. Theor. Appl. Inf. Technol.*, 2021.

[10] S. Pillai and D. A. Sharma, "Hybrid unsupervised web-attack detection and classification--A deep learning approach," *Comput. Stand. Interfaces*, 2023, doi: 10.1016/j.csi.2023.103738.

[11] R. I. E. P. Ghani and A. Triwiyatno, "Detection and Mitigation Effectiveness of Injection and Remote Service Attacks: A Machine Learning-Based Evaluation," in *Proc. ISMEE*, 2025. doi: 10.1109/ISMEE68179.2025.11473023.

[12] V. C. R. Branco and I. Medeiros, "Towards a Web Application Attack Detection System Based on Network Traffic and Log Classification," in *Proc. ENASE*, 2024. doi: 10.5220/0012722800003687.

[13] J. B. S. A. Kumar and R. Agarwal, "Machine Learning-Based Web Application Firewall for Real-Time Threat Detection," in *Proc. IEEE ICEI*, 2024. doi: 10.1109/ICEI64305.2024.10912239.

[14] N. Stevanović, B. Todorović, and V. Todorović, "Web attack detection based on traps," *Applied Intelligence*, 2022, doi: 10.1007/s10489-021-03077-9.

[15] B. C. Z. Cheng T. Qi W. Yang and J. Fu, "An Improved Feature Extraction Approach for Web Anomaly Detection Based on Semantic Structure," *Security and Communication Networks*, 2021, doi: 10.1155/2021/6661124.

[16] A. E. Takieldeen, "Web Attack Intrusion Detection System Using Machine Learning Approaches for Cybersecurity," in *Proc. ITC-Egypt*, 2025. doi: 10.1109/ITC-Egypt66095.2025.11186700.

[17] J. H. Kumar and J. G. Ponsam, "Securing Web Application using Web Application Firewall and Machine Learning," in *Proc. ICAEECI*, 2023. doi: 10.1109/ICAEECI58247.2023.10370872.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

[18] W.-C. Y. L. Zhou Y. S. Gan and S.-T. Liong, "E-WebGuard: Enhanced neural architectures for precision web attack detection," *Comput. Secur.*, 2025, doi: 10.1016/j.cose.2024.104127.

[19] A. Shaheed and M. H. D. B. Kurdy, "Web Application Firewall Using Machine Learning and Features Engineering," *Security and Communication Networks*, 2022, doi: 10.1155/2022/5280158.

[20] R. Bakır, "UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques," *Arab. J. Sci. Eng.*, 2025, doi: 10.1007/s13369-024-09916-4.

[21] J. Yang, "LLM-AE-MP: Web Attack Detection Using a Large Language Model with Autoencoder and Multilayer Perceptron," *Expert Syst. Appl.*, 2025, doi: 10.1016/j.eswa.2025.126982.

[22] J.-Á. Román-Gallego, M.-L. Pérez-Delgado, M. L. Viñuela, and M.-C. Vega-Hernández, "Artificial Intelligence Web Application Firewall for advanced detection of web injection attacks," *Expert Syst.*, 2025, doi: 10.1111/exsy.13505.

[23] C. X. T. Hu S. Zhang S. Tao and L. Li, "Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism," *Comput. Secur.*, 2023, doi: 10.1016/j.cose.2022.102990.

[24] K. M. Manjunatha and M. Kempanna, "Count vectorizer model based web application vulnerability detection using artificial intelligence approach," *Journal of Discrete Mathematical Sciences and Cryptography*, 2022, doi: 10.1080/09720529.2022.2133243.

[25] B. A. Meharaj and M. Arock, "Modified Parse-Tree Based Pattern Extraction Approach for Detecting SQLIA Using Neural Network Model," *ISeCure*, 2024, doi: 10.22042/isecure.2023.370697.886.

[26] N. Yadav and N. Shekokar, "Preprocessing HTTP Requests and Dimension Reduction Technique for SQLI Detection," in *Lecture Notes in Networks and Systems*, 2021. doi: 10.1007/978-3-030-67187-7_21.

[27] F. Younas, "An efficient artificial intelligence approach for early detection of cross-site scripting attacks," *Decision Analytics Journal*, 2024, doi: 10.1016/j.dajour.2024.100466.

[28] R. Alhamyani and M. Alshammari, "Machine Learning-Driven Detection of Cross-Site Scripting Attacks," *Information*, 2024, doi: 10.3390/info15070420.

[29] S. Sharma and N. S. Yadav, "A multilayer stacking classifier based on nature-inspired optimization for detecting cross-site scripting attack," *International Journal of Information Technology*, 2023, doi: 10.1007/s41870-023-01459-5.

[30] Trustwave SpiderLabs, "ModSecurity Reference Manual," 2026.

[31] OWASP Foundation, "OWASP ModSecurity Core Rule Set Documentation," 2026.