Indonesian Journal of Artificial Intelligence and Data Mining (IJAIDM)
Vol. 8, No. 3, November 2025, pp. 788 – 799
p-ISSN: 2614-3372 | e-ISSN: 2614-6150

Analysis of SQL Injection and Cross-Site Scripting (XSS) Attacks on Web Server Logs Using Machine Learning

[1*]Adi Septian, [2]Atep Aulia Rahman,
[1,2]Department of Informatics Engineering, Widyatama University, Indonesia
Email: [1]adi.septian@widyatama.ac.id, [2]atep.aulia@widyatama.ac.id

Article Info

Article history:
Received Sep 29th, 2025
Revised Nov 15th, 2025
Accepted Nov 31th, 2025

Keyword:
Machine Learning
Nginx
Random Forest
SQL Injection
XSS

ABSTRACT

The increasing complexity of cyber threats requires accurate detection systems to identify attack patterns on web servers. This study aims to detect SQL Injection and Cross-Site Scripting (XSS) attacks in Nginx access logs using machine learning algorithms. Log data were processed through regular expressions for parsing and labeling, resulting in 1,650,615 samples. Data imbalance was addressed using a combination of ADASYN and Random Undersampling. Two algorithms, Random Forest and Support Vector Machine (SVM), were compared based on accuracy, precision, recall, F1-score, and ROC curve metrics. The results show that Random Forest achieved the best performance with 99.92% accuracy, 99.94% F1-score, and 0.9994 AUC, while SVM obtained an accuracy of 96.45%. The combination of resampling and ensemble learning significantly enhances the effectiveness of log-based attack detection, providing a promising foundation for the development of adaptive Intrusion Detection Systems (IDS) in web server environments.

Corresponding Author:
Third Author,
Department of Informatics Engineering, Widyatama University,
Street of Cikutra No. 204A, Sukapada, Cibeunying Kidul, Bandung City,
West Java 40124, Indonesia.
Email: adi.septian@widyatama.ac.id

DOI: http://dx.doi.org/10.24014/ijaidm.v8i3.38397

1. INTRODUCTION

The increasing complexity of cyber threats has placed web server security as a top priority. Attacks such as SQL Injection and Cross-Site Scripting (XSS) have become serious concerns, as both exploit vulnerabilities in web applications to steal sensitive data, manipulate systems, or disrupt operations [1]. According to OWASP, in 2021 SQL Injection ranked 3rd with a total of 274,000 incidents, and XSS is one of the major vulnerabilities included in the list [2]. OWASP Top 10 Web Vulnerabilities 2017–2021 can be seen Figure 1.

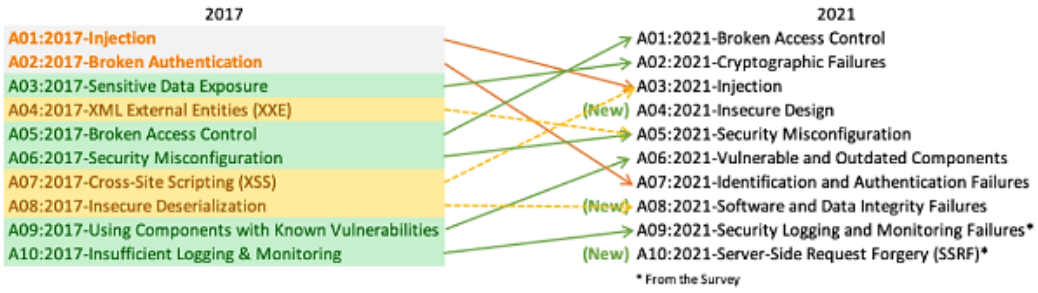

Figure 1. OWASP Top 10 Web Vulnerabilities 2017–2021

In this context, it is crucial to develop detection methods that are not only accurate but also efficient in maintaining the integrity of information systems. One relevant approach is the application of machine

learning as a tool for classification and attack pattern detection. This study covers the process from collecting attack datasets to implementing machine learning algorithms specifically Random Forest and Support Vector Machine (SVM) in detecting SQL Injection and XSS attacks through the analysis of user input patterns and relevant log attributes. The dataset used includes various SQL Injection and XSS attack patterns obtained from Nginx web server access log files, which were previously parsed and labeled using regular expression (regex) techniques according to the attack types being tested. Validation and preprocessing procedures were carried out to ensure data quality, including normalization and relevant feature extraction. With this approach, the resulting model is expected to detect attacks with high accuracy while providing deeper insight into complex attack patterns [3].

In its application, the Random Forest algorithm is a relevant choice because it can process data with many features and is less prone to overfitting. In previous research, the Random Forest algorithm demonstrated the highest performance in detecting SQL Injection vulnerabilities with 99.78% accuracy, while the SVM algorithm ranked next with 94% accuracy [4]. The approach was further extended by [5] hrough the integration of Random Forest with a multi-channel deep learning architecture for XSS detection, resulting in nearly perfect accuracy across multiple public datasets. Based on these values, both algorithms will be adopted in this study. The study conducted by [6] implemented several machine learning algorithms, including SVM, KNN, and Naïve Bayes, to detect SQL Injection attacks based on payload analysis and demonstrated competitive classification performance across various testing scenarios. A similar approach was proposed by [7] who developed a two-stage classification pipeline utilizing feature extraction and ensemble learning, resulting in consistent improvements in SQL Injection detection accuracy on structured datasets. In the context of XSS attacks, [8] introduced a hybrid CNN–LSTM model capable of detecting both SQL Injection and XSS simultaneously, with experimental results on the HTTP CSIC 2010 dataset showing high accuracy in identifying content-based attack patterns in HTTP requests. Overall, these studies affirm the effectiveness of machine learning in detecting web-based attacks, although most still rely on public or structured datasets rather than real-world web server logs.

The research gap addressed in this study lies in the aspects of data sources, detection levels, and combined attack types. Most previous studies detect SQL Injection or XSS attacks at the application level using simulated payloads or public datasets such as CSE CIC IDS2018 and the Kaggle SQL Injection Dataset, which are structured and do not represent real-world server log conditions. Moreover, previous research generally focuses on detecting a single type of attack either SQL Injection or XSS without considering the possibility of hybrid attacks that may occur simultaneously within a single web traffic flow. The novelty of this research bridges these gaps by introducing a new approach that detects both SQL Injection and XSS attacks based on a local dataset derived from actual Nginx web server logs rather than public datasets, thereby producing results that more accurately reflect real world conditions. Additionally, this study proposes a preprocessing pipeline based on regular expression (regex) for the parsing and labeling process of both SQL Injection and XSS attacks.

Furthermore, this study compares the performance results of the two algorithms in detecting attacks using evaluation criteria such as accuracy, precision, recall, and F1 score as indicators of model performance. In addition, an analysis is conducted on input features that have a significant influence on the classification process, enabling the model not only to detect attacks but also to provide deeper insight into the characteristics of each attack type. With this approach, the findings of this research are expected to make a tangible contribution to the development of automated detection-based security systems that can adapt to the rapidly evolving landscape of cyber threats.

2. RESEARCH METHOD

This study applies a comparative experimental method with a quantitative approach to evaluate the performance of various machine learning algorithms in identifying SQL Injection and XSS attacks based on web server log data. This experiment compares the performance of the Random Forest and SVM algorithms using the same Nginx web server log dataset to ensure the validity of the comparison results. The research flow is illustrated in Figure 2. In general, the research process is divided into six main stages:

1. Collection of Nginx Log Data
2. Log Parsing and Information Extraction
3. Data Labeling
4. Preprocessing and Feature Extraction
5. Sequential Resampling Strategy
6. Model Training and Evaluation

2.1. Data Collection

The raw data used in this study were obtained from Nginx access logs collected from a real web server environment. These logs are typically stored as text files on the server and contain records of every request

made to the web server. The format of Nginx logs may vary depending on the configuration and type of request. The default format typically includes information such as the client's IP address, timestamp, HTTP method (e.g., GET, POST), requested URL, HTTP status code (e.g., 200, 404, 503), user agent (which typically indicates the browser name), and other relevant parameters. An example of a log entry is shown in Figure 3.

Each log line represents a single user request, including the IP address, HTTP method, accessed URL, response status code, and data size. This raw data is stored as a log file (access.log) and serves as the primary source for analysis.

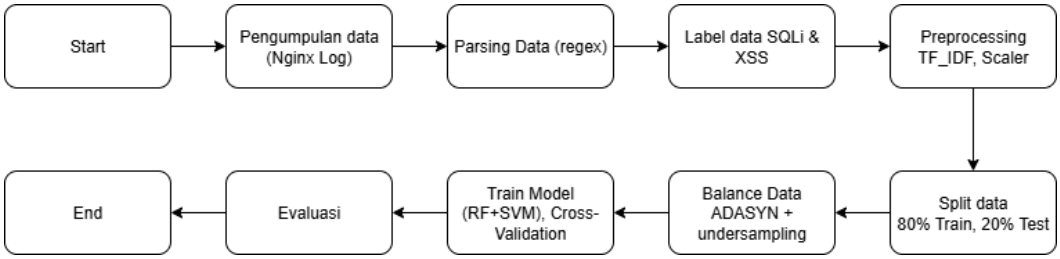

Figure 2. Research Flow

```
192.168.1.1 - - [10/Oct/2023:13:55:36 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.
110 Safari/537.3"
```

Figure 3. Example of Log Entry

2.2. Data Preprocessing

Data preprocessing is a crucial stage in this research because the obtained web server log data is still in raw form and contains various information such as IP address, timestamp, HTTP method, URL, status code, and user agent. Without preprocessing, this data cannot be directly utilized by machine learning algorithms, thus, it needs to be processed to ensure it is more structured, clean, and suitable for analysis [9].

2.3. Log Data Parsing

Parsing is performed to extract relevant information from each log line, which will later be used to construct a structured dataset. Regex techniques were chosen because of their flexibility in matching complex text patterns and their suitability for extracting specific information from logs, such as IP address, timestamp, HTTP method, URL, status code, and response size [10]. Additionally, regex is easy to integrate with the Python programming language used in this study [11]. Log parsing regex script is shown in Figure 4.

```
^(?P<ip>\d+\.\d+\.\d+\.\d+) - - \[(?P<timestamp>[^\]]+)\] "(?P<method>\w+) (?P<url>[^ ]+) HT
TP/\d\.\d" (?P<status>\d+) (?P<size>\d+) "-" "(?P<user_agent>[^"]+)"
```

Figure 4. Log Parsing Regex Script

The result of the parsing process is raw data that has been successfully extracted and organized into a CSV format, enabling further processing by machine learning algorithms. Irrelevant elements have been removed, leaving only the necessary information for the next stages. With a cleaner and more organized structure, the parsed data is now ready for further processing. The first ten rows of parsing results can be seen Table 1.

Table 1. The first ten rows of parsing results.

No	IP	Date	Method	URL	Status_code	Size
0	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1724930667.81729698..	200	31
1	52.167.144.222	29-Aug-24	GET	/?kxs2967w4yw1	200	26101
2	52.167.144.138	29-Aug-24	GET	/?product%2FP%3F662747=wgatemenz	200	26101
3	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1724930811.96527004..	200	31
4	52.167.144.222	29-Aug-24	GET	/?yqwzcwma	200	26101
5	85.208.96.206	29-Aug-24	GET	/page/3/?MecDisplay=default&eventDate=2022-10-...	200	22792

No	IP	Date	Method	URL	Status_code	Size
6	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1742265457.27636504..	200	31
7	66.249.66.196	29-Aug-24	GET	/kocok/?burung=BONUS123	404	19593
8	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1742265543.67830801..	200	31
9	52.167.144.145	29-Aug-24	GET	/index.php?z/R3868983	301	5

This table consists of 1,650,615 rows and 6 columns, where each column represents important information related to user or bot activities accessing a website. From all features in the Nginx access log file, only the IP, date, method, URL, status code, and size features are used for analysis, as shown in Table 2 for the attribute dataset. The remaining features, namely timestamp and user-agent, are not included because they are less relevant to the detection patterns of SQL Injection and XSS attacks.

Table 2. Dataset Attributes

Attribute	Description
IP	Client IP address
Date	Access date
Method	HTTP request method sent to the server
URL	URL path accessed by the client
Status_code	HTTP response code returned by the server
Size	Response size (in bytes)

The regex pattern used was specifically designed to extract six primary attributes. These attributes include the client IP address (ip), date and time components (date), HTTP method (method), accessed URL path (url), HTTP response status code (status_code), and response size (size). The extraction of the url column is particularly critical because it serves as the main textual input expected to contain potential attack payloads.

2.4. Data Labeling

After the parsing stage, the dataset is labeled based on detected attack patterns. The labeling technique was performed using regex to identify characteristics of malicious behavior. In this research, two types of attacks are the primary focus: SQL Injection and XSS. Both are categorized as injection attacks, which remain among the most critical threats to web applications. The regex patterns used to detect SQL Injection attacks in the log, particularly within the url attribute, involve matching specific indicators such as single quotes followed by harmful SQL commands like UNION, SELECT, INSERT, DROP, or logical conditions like OR 1=1, database table names such as information_schema, wp, xp, or command separators commonly exploited to manipulate queries [12]. For XSS detection, the regex rules target malicious scripts injected on the client side, such as the <script> tag, inline event attributes (on...=), JavaScript: scheme usage, and access attempts to the document. cookie, DOM manipulation actions (innerHTML, window.location), as well as the execution of functions like eval(), base(), and innerHTML [13].

The data labeling process provides contextual categorization to each parsed log entry, enabling the dataset to include attributes that can be effectively utilized in advanced analysis, such as machine learning model training. This labeling aims to identify and classify request types into specific categories based on detected patterns, particularly from the url column where attack parameters are typically embedded. The results shows the newly formed dataset containing an attack type column as a label for each data row.

Table 3. The first ten rows of data labeling results.

No	Ip	Date	Method	url	Status_ code	Size	Attack_ type
0	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1724930667.81729698...	200	31	normal
1	52.167.144.222	29-Aug-24	GET	/?kxs2967w4yw1	200	26101	normal
2	52.167.144.138	29-Aug-24	GET	/?product%2FP%3F662747=wgatemenz	200	26101	normal
3	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1724930811.96527004...	200	31	normal
4	52.167.144.222	29-Aug-24	GET	/?yqwzcwma	200	26101	normal
5	85.208.96.206	29-Aug-24	GET	/page/3/?MecDisplay=default&eventDate=2022-10-...	200	22792	normal
6	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1742265457.27636504...	200	31	Normal

No	Ip	Date	Method	url	Status_ code	Size	Attack_ type
7	66.249.66.196	29-Aug-24	GET	/kocok/?burung=BONUS123	404	19593	normal
8	103.78.195.111	29-Aug-24	POST	/wp-cron.php?doing_wp_cron=1742265543.6783 0801...	200	31	normal
9	52.167.144.145	29-Aug-24	GET	/index.php?z/R3868983	301	5	normal

2.5. Data Cleaning

The data cleaning stage was carried out to improve the quality of the dataset by removing irrelevant special characters, checking and ensuring that there are no missing values in essential attributes, and eliminating duplicate data that could potentially introduce bias into the model [14]. Data cleaning was performed by removing duplicate records, deleting empty values in the URL and attack type columns, and removing non-alphabetic characters to maintain consistency during tokenization. The Missing Value Check Result is presented in Table 4.

Table 4. Missing Value Check Result

Attribute	Missing value
IP	0
Date	0
Method	0
URL	0
Status_code	0
Size	0
Attack_size	0

It has been confirmed that there are no missing values in the dataset, allowing the process to proceed to the next stage without further processing.

2.6. Feature Extraction

The sampled data must then be converted into a numerical representation suitable for machine learning algorithms. Feature engineering involves handling both textual features (url) and numerical features (status code, size).

2.6.1. Textual Feature Extraction Using TF-IDF Vectorization

The URL column is the most informative feature vector. The Term Frequency Inverse Document Frequency (TF-IDF) method is selected for text transformation. The principle of TF-IDF is to assign specific scores or weights to each word or character in the data [15]. Words that frequently appear within a single URL but rarely across the entire dataset are assigned higher weights. In the context of IDS, attack-related terms (e.g., union, script, eval) are uncommon in normal traffic, making TF IDF useful for highlighting important tokens that indicate attacks. The TfidfVectorizer module from the scikit-learn library is used to transform the url column into a high-dimensional feature matrix. The parameters used are as follows:

1. Max_features = 5000
   Limits the number of features to the top 5000 TF-IDF-weighted terms, preventing excessive dimensionality and reducing computation time, especially given the high variability of URLs in the log dataset.
2. Ngram_range = (1, 2)
   Considers both unigrams (single words) and bigrams (two-word combinations).
3. Lowercase = True
   Converts all tokens to lowercase to maintain consistency and prevent duplicate features due to case differences.
4. token_pattern = r'[A-Za-z0-9_@./?&=-]+'
   Customized for URL structure, which includes alphanumeric characters and symbols such as /, =, ?, and &.

The output of TF-IDF is a high-dimensional, sparse feature matrix, well-suited for classification models such as SVM.

2.6.2. Numerical Feature Transformation Using Standard Scaler

Two numerical features, status_code and size, must be standardized before being combined with TF-IDF output. The StandardScaler technique from scikit-learn is applied. StandardScaler normalizes each feature so that its mean becomes 0 and its standard deviation becomes 1 [16].

The parameter with_mean = False is used because TF-IDF produces a sparse matrix, making mean-centering impractical without disrupting sparsity. This ensures efficient memory usage while maintaining an equal feature scale. Scaling is crucial because the size attribute may range from tens to millions of bytes, significantly larger than other attributes. Without standardization, models, especially distance-based ones like SVM, would be dominated by this feature, diminishing the influence of attack-related patterns captured by TF-IDF. Standard scaling ensures all features contribute proportionally, enabling optimal learning.

2.7. Data Splitting

In this study, the data is split using an 80:20 ratio, where 80% is used for model training and 20% for performance testing. This ratio is widely applied in text classification and intrusion detection research, as it effectively balances learning and validation datasets. The splitting process utilizes the train_test_split function from scikit-learn, with the stratify parameter enabled to maintain class balance in both the training and testing sets. This approach is crucial for web attack datasets, which are often imbalanced, ensuring that minority attack classes are properly represented and reducing model bias toward majority classes. The data splitting can be seen Figure 5.

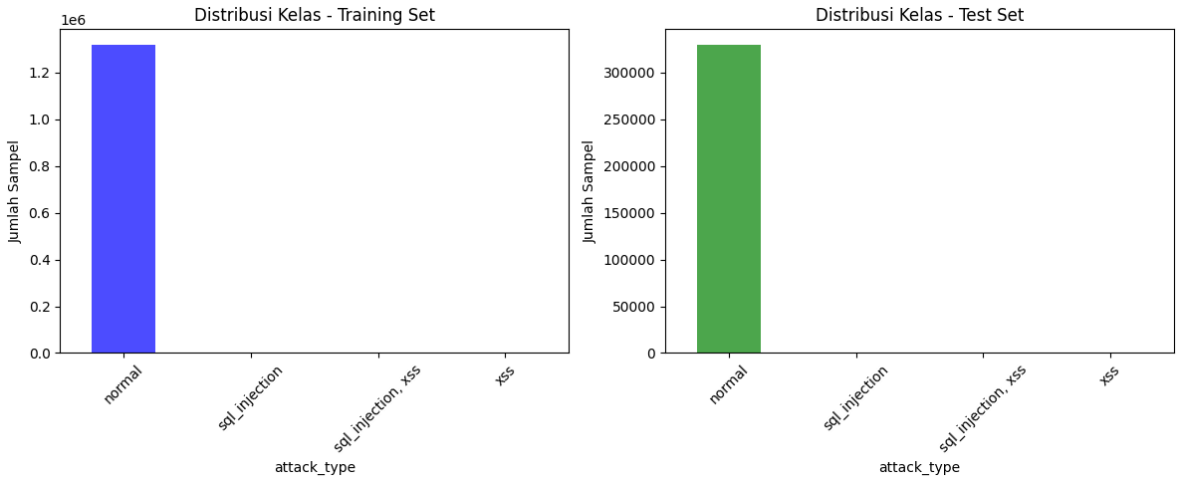

Figure 5. Data Split

2.8. Data Imbalance Analysis

The class distribution in the dataset indicates a severe class imbalance issue. Out of a total of 1,650,615 samples, the majority represent normal traffic. Initial class distribution of the dataset can be seen Table 5.

Table 5. Initial Class Distribution of the Dataset

Attack Type	Number of Samples	Percentage
Normal	1,649,331	99.92%
SQL Injection	852	0.05%
XSS	125	0.007%
SQL Injection, XSS	307	0.018%
Total Attacks	1,284	0.08%

There are also samples classified as dual-label (sql_injection, xss), representing a single request that carries attack payloads from both types. This severe imbalance has serious implications: a model trained without mitigation will tend to classify all requests as “normal,” resulting in high accuracy but an attack detection rate close to zero [17].

2.9. Sequential Resampling Strategy

The collected dataset exhibited a severe class imbalance, with normal traffic (majority class) dominating over 99.9% of the samples, while attack instances (minority class) constituted less than 0.1%. To mitigate the bias towards the majority class and improve the model's sensitivity to attacks, a sequential hybrid resampling technique was implemented.

To ensure that the model is not biased toward the majority class, resampling techniques must be integrated into the training method. In the context of IDS, minimizing False Negatives is the highest operational priority. This process was executed in two sequential steps:

1. Adaptive Synthetic Sampling (ADASYN)
   Used to generate synthetic samples from the minority classes. ADASYN is chosen because it focuses on creating new samples around minority class samples that are difficult to classify (those near the decision boundary) [18].

2. Random Undersampling
   Used to reduce the number of samples from the majority class (normal). This step has a dual purpose: reducing the dominance of the normal class during training and saving significant computational resources on such a large dataset [19].

By sequentially applying ADASYN followed by Random Undersampling, the dataset achieved a balanced distribution. This hybrid approach ensures that the model learns complex attack boundaries effectively (via ADASYN) while maintaining computational efficiency and reducing majority class bias (via Undersampling).

Resampling is integrated into the pipeline (using ImbPipeline from imblearn) and is strictly applied only to the 80% training data. This procedure prevents data leakage, ensuring that the 20% test data remains untouched and provides an unbiased performance evaluation.

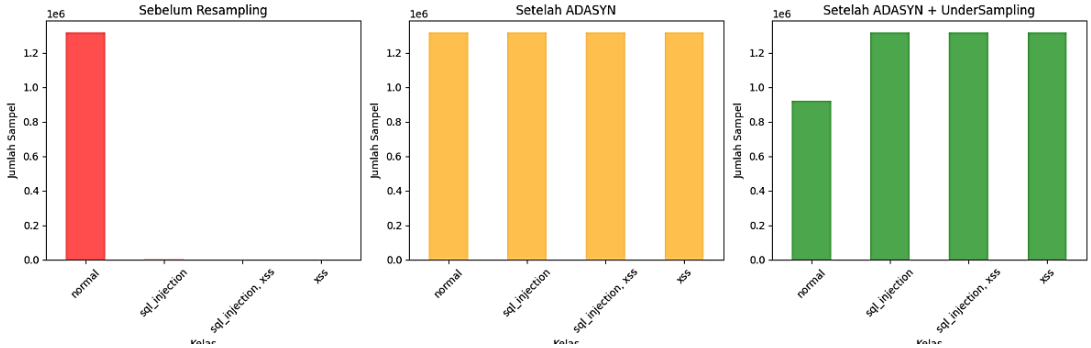

Figure 6. Before and After Resampling

2.10. Machine Learning Model Training

The training process compares the performance of two primary algorithms, SVM and Random Forest, selected due to their complementary characteristics and proven effectiveness in text classification and cyber-attack detection. The parameters used for training both algorithms will be discussed in the table 6.

Table 6. Machine Learning Model Parameters

Model	Parameter	Description
Support Vector Machine	Kernel = 'linear'C = 1.0	Suitable for high-dimensional data
Random Forest	n_estimators = 200max_depth = 15	Prevents overfitting and handles nonlinear data

To ensure that the obtained results are not limited to a specific dataset, a Stratified K-Fold Cross-Validation technique [20] with k = 3 was applied.

2.11. Model Evaluation

After training, the generated models are evaluated using a separate testing dataset. In this stage, the confusion matrix is employed, which provides several metrics such as precision, recall, and accuracy in classifying SQL Injection, XSS attacks, or both. Additionally, the ROC Curve will be presented along with comparative performance metric graphs for each of the machine learning algorithms utilized.

3. RESULTS AND ANALYSIS

The objective of this experiment is to detect SQL Injection and XSS attacks using the Random Forest and SVM algorithms. The initial analysis results indicate that the class distribution is highly imbalanced, where normal traffic dominates more than 99% of the data. Therefore, data balancing was performed using a combination of ADASYN and Random Undersampling techniques. All modeling processes were carried out using the scikit-learn library in a Google Colab environment, with an 80% and 20% split for training and

testing, respectively. The optimal parameters were obtained through hyperparameter tuning and cross-validation.

3.1. Model Training Results

The model training process utilized the K-fold cross-validation method with k = 3 on a subset of 100,000 resampled samples. Subsequently, the model was retrained on the entire synthetic dataset to obtain the final performance results on the test data, which consists of 20% of the total 330,123 samples.

Table 7. Cross-Validation Results

Model	CV Accuracy	CV F1	Test Accuracy	Test F1	Precision	Recall
Random Forest	0.9973 ± 0.0002	0.9974 ± 0.0003	0.9992	0.9994	0.9996	0.9992
SVM	0.8978 ± 0.0004	0.9008 ± 0.0002	0.9645	0.9813	0.9992	0.9645

Random Forest demonstrates the highest performance across all evaluation metrics, both during cross-validation and final testing. The testing accuracy reaches 99.92%, with an F1-score of 99.94%, indicating that the model effectively classifies both attacks and normal traffic. The difference in accuracy values between cross-validation and testing results is also very small (around 0.2%), which suggests that the model is stable and does not experience overfitting. Meanwhile, the SVM with a Linear kernel shows fairly good results with a testing accuracy of 96.45%. The precision and recall values are also considered high for the majority class, but the F1-score is relatively lower compared to Random Forest. This can be interpreted as meaning that although SVM is capable of correctly detecting most data patterns, the model tends to be less adaptive to feature variations in the minority class, especially due to the highly imbalanced nature of the data.

3.2. Classification Analysis

3.2.1. Random Forest

The model demonstrates very strong performance in detecting the normal class, with a precision of 1.0000, a recall of 0.9992, and an F1-score of 0.9996, indicating an almost perfect ability to accurately classify normal traffic. For the SQL_injection class, the model also performs quite well, with a precision of 0.7257, a recall of 0.9647, and an F1-score of 0.8283, indicating that the model can correctly identify most SQL Injection attacks with relatively low error rates. However, the performance decreases significantly in the classes sql_injection, xss and xss. The SQL injection and XSS classes have a low precision of 0.2533, although the recall is high (0.9194), indicating that there are many false positive predictions for these classes. Meanwhile, the XSS class achieves a precision of 0.3860 and a recall of 0.8000, with an F1-score of 0.5366, suggesting that the model still struggles to accurately recognize XSS attack patterns.

Table 8. Random Forest Classification Report

	Precision	Recall	F1-score	Support
Class Normal	10.000	0.9992	0.9996	329866
Class sql_injection	0.7257	0.9647	0.8283	170
Class sql_injection, xss	0.2533	0.9194	0.3972	62
Class xss	0.3860	0.8800	0.5266	25
Accuracy			0.9992	330123
Macro avg	0.5912	0.9408	0.6904	330123
Weighted avg	0.9996	0.9992	0.9994	330123

Overall, the model achieves an accuracy of 99.92%, but the macro average F1-score is only 0.6904, indicating a performance imbalance across classes. The high weighted average (0.9994) indicates that this high accuracy is primarily driven by the significantly larger number of normal class data. Therefore, although the model appears highly accurate in general, its performance on minority classes, particularly XSS attacks and the combined SQL Injection and XSS class, still needs improvement through data imbalance handling or model parameter optimization.

3.2.2. Support Vector Machine

The SVM classification report indicates that the model's performance remains suboptimal, particularly in recognizing minority attack classes. The normal class has a perfect precision score of 1.0000 and a recall of 0.9646, resulting in a high F1 score of 0.9819. This means that the model performs very well in identifying normal traffic and rarely makes errors for this class. However, performance drops drastically for the attack classes. For the SQL injection class, the precision score is 0.0784, with a recall of 0.8706 and an F1-score of 0.1438. This indicates that although the model is relatively successful in identifying SQL Injection

attacks, the rate of false positive predictions is extremely high. The SQL injection and XSS classes show even lower performance, with a precision of only 0.0060, a recall of 0.8387, and an F1-score of 0.0120, indicating that the model frequently misclassifies this combined attack type. Similarly, for the XSS class, precision is 0.0148, recall is 0.8400, and the resulting F1-score is 0.0290, indicating a severe imbalance between precision and recall. Overall, although the model achieves a total accuracy of 96.45%, this metric is not entirely reliable because it is heavily dominated by the significantly larger normal class. The low macro average F1-score (0.2917) and macro precision (0.2748) show that the model fails to provide balanced performance across classes. Thus, this SVM model has a strong bias toward the majority class (normal).

Table 9. SVM Classification Report

	Precision	recall	F1-score	Support
Class Normal	10.000	0.9646	0.9819	329866
Class sql_injection	0.0784	0.8706	0.1438	170
Class sql_injection, xss	0.0060	0.8387	0.0120	62
Class xss	0.0148	0.8400	0.0290	25
Accuracy			0.9645	330123
Macro avg	0.2748	0.8785	0.2917	330123
Weighted avg	0.9992	0.9645	0.9813	330123

3.3. Confusion Matrix Analysis

In the Random Forest model, the performance is remarkably high with an accuracy of 99.9%. The normal class is classified extremely well, where 329,607 samples (99.9%) are correctly detected as normal traffic, with only a very small portion misclassified into other classes. The sql_injection class also shows excellent results with 96.5% correct predictions, while the sql_injection, xss and xss classes each achieve correct classification rates of 91.9% and 88.0%, respectively. Misclassifications in this model are relatively very low, demonstrating the superior capability of Random Forest in distinguishing each type of attack accurately. Meanwhile, the SVM model results in a lower accuracy of 96.4%. The normal class is still detected well (96.5% correct), but there is an increase in misclassification into other classes such as sql_injection and sql_injection, xss. The sql_injection class has a correct prediction rate of 87.1%, while the sql_injection, xss and xss classes achieve only 83.9% and 84.0%, respectively. This indicates that SVM still struggles to differentiate attacks that share similar characteristics, especially between sql_injection and sql_injection, xss.

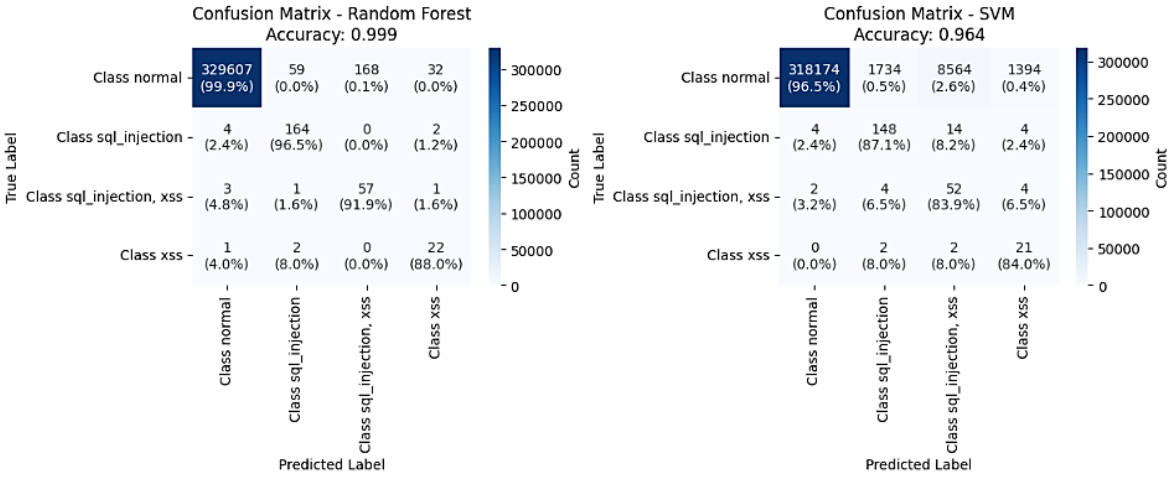

Figure 7. Confusion Matrix of Random Forest and SVM

3.4. ROC Curve Analysis

A performance comparison between the Random Forest model (left) and the SVM model (right) in distinguishing each attack class. In the Random Forest model, the curves for the normal, sql injection, sql_injection, xss, and xss classes lie very close to the top-left axis, indicating a high True Positive Rate (TPR) and an extremely low False Positive Rate (FPR). The AUC (Area Under the Curve) values for each class are also nearly perfect, namely 0.999 for normal, 0.999 for sql_injection, 0.999 for sql injection, xss, and 1.000 for xss. These findings indicate that the Random Forest model can distinguish normal traffic from various attacks with very high accuracy. In the SVM model, the results are slightly lower compared to Random Forest, although they still show reasonably good performance. The AUC value for the normal class is 0.996, 0.985 for

sql_injection, 0.970 for sql injection, xss, and 0.981 for xss. The ROC curves for SVM tend to be slightly farther from the top-left axis, indicating that this model has a higher detection error rate compared to Random Forest. Overall, both models demonstrate strong classification capabilities, but Random Forest proves to be superior in separating attack classes with higher accuracy and more consistent performance across all classes.

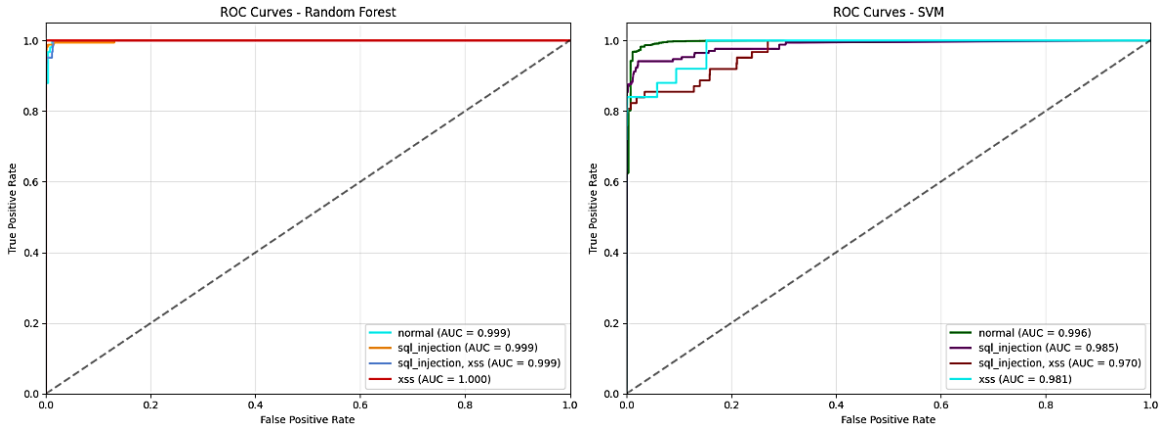

Figure 8. ROC Curve of Random Forest and SVM

3.5. Discussion

Based on all test results, Random Forest has been proven to provide the best performance in detecting web attacks from HTTP log data. Its ability to handle large datasets and non-linear features makes it superior compared to SVM. In addition, the ensemble learning mechanism in Random Forest helps reduce prediction variations caused by differences in training data and increases the stability of predictions. On the other hand, SVM with a Linear kernel performs well on binary data, but is less optimal for multi-class imbalanced data, especially when minority classes have very similar patterns. The low precision value indicates that there are still a relatively high number of false positives, which in the context of cybersecurity can lead to increased manual analysis workload.

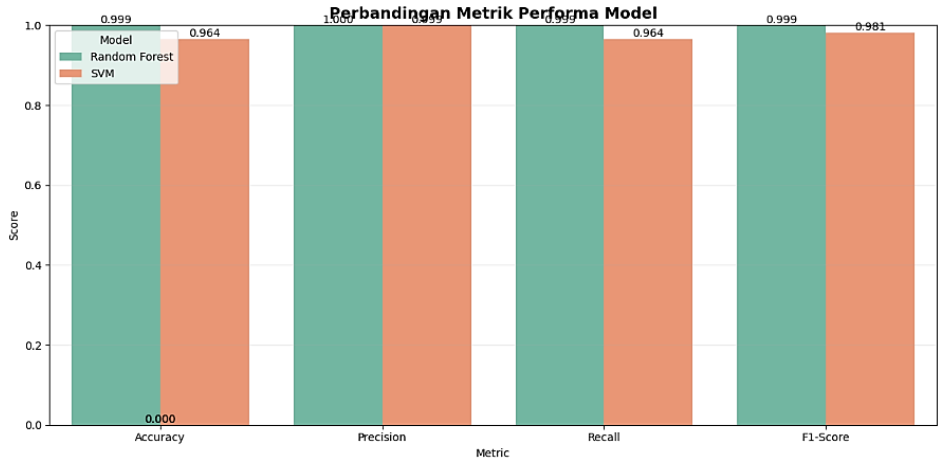

Figure 9. Performance Metric Comparison of Random Forest and SVM

The Random Forest model consistently achieves higher metric values compared to SVM across all evaluation aspects based on the graph. Overall, the integration of the ADASYN method with Random Forest has proven effective in addressing class imbalance and improving the detection of low frequency attacks. With an almost perfect AUC and an accuracy of 99.9%, this model is suitable to be used as the foundation for a machine learning based Intrusion Detection System (IDS) for web servers. Although the results show high performance on the Nginx log dataset, further evaluation is needed on different web servers, considering that the differences in log structure and attack patterns on other servers such as Apache or IIS may reduce model accuracy. Testing using external log datasets is recommended to assess the generalization ability and stability of the model, ensuring that threat detection remains effective across various server environments and is not limited to local data.

4. CONCLUSION

The primary issue in this study is data imbalance, where the number of normal data is significantly larger than that of the attack data. To address this issue, a combination of the ADASYN method and Random Undersampling was used, which proved effective in balancing the class distribution. The Random Forest algorithm demonstrated the best performance compared to SVM, achieving an accuracy of 99.92%, an F1-score of 99.94%, and an AUC of 0.9994. These results indicate the model's strong ability to distinguish between normal traffic and attacks during both training and testing stages.

The SVM algorithm also provided fairly good performance; however, it tends to struggle in identifying minority classes, especially in multi-class classification cases with uneven data distribution. The low precision value for attack classes indicates that this model still produces a considerable number of false positives, which may impact the efficiency of the detection system when implemented in real environments. The use of various evaluation metrics, such as precision, recall, F1-score, and the ROC Curve, provides a more comprehensive performance assessment than relying solely on accuracy. This approach ensures that the model achieves high accuracy while maintaining optimal capability in detecting attack patterns with minimal errors. Thus, this study successfully demonstrates that the combination of appropriate resampling strategies and ensemble learning approaches, such as Random Forest, can significantly enhance the performance of web attack detection systems. These findings suggest that machine learning–based methods are a viable solution for automated attack detection using HTTP logs.

Practically, the resulting Random Forest model has the potential to be implemented as part of an adaptive Intrusion Detection System (IDS) on web servers, as mentioned in the abstract. This system can monitor logs in real-time, detect SQL Injection and XSS patterns, and provide early warnings to enhance the security of web applications. For future research, it is recommended to conduct real-time testing in production environments, evaluate the model on log datasets from other web servers (such as Apache or IIS) to assess generalization capability, and explore Deep Learning algorithms (such as CNN-LSTM or Transformer-based models) that have the potential to capture more complex and varied attack patterns.

ACKNOWLEDGEMENTS

The author would like to express sincere gratitude to all lecturers, colleagues, and supporting institutions that provided guidance, technical assistance, and valuable feedback throughout the completion of this research on analyzing SQL Injection and Cross-Site Scripting (XSS) attacks on web server logs using machine learning.

REFERENCES

[1] Chen Y, Liang G, Wang Q. Research on SQL Injection Detection Technology Based on Content Matching and Deep Learning. Computers, Materials and Continua 2025;84:1145–67. https://doi.org/10.32604/cmc.2025.063319.
[2] OWASP Top 10:2021 n.d. https://owasp.org/Top10/A03_2021-Injection/ (accessed September 18, 2025).
[3] Kavitha C, Saravanan M, Gadekallu TR, Nimala K, Kavin BP, Lai WC. Filter-Based Ensemble Feature Selection and Deep Learning Model for Intrusion Detection in Cloud Computing. Electronics (Switzerland) 2023;12. https://doi.org/10.3390/electronics12030556.
[4] Rahayu A, Yulyanti E, Ghalib M. Systematic Literature Review: SQL Injection Detection Vulnerability Using Machine Learning. Jurnal Media Infotama 2025;21:15–20. https://doi.org/10.37676/jmi.v21i1.6702.
[5] Qin Q, Li Y, Mi Y, Shen J, Wu K, Wang Z. Detecting XSS with Random Forest and Multi-Channel Feature Extraction. Computers, Materials and Continua 2024;80:843–74. https://doi.org/10.32604/cmc.2024.051769.
[6] Triloka J, Hartono H, Sutedi S. Detection of SQL Injection Attack Using Machine Learning Based On Natural Language Processing. International Journal of Artificial Intelligence Research 2022;6. https://doi.org/10.29099/ijair.v6i2.355.
[7] Rosca CM, Stancu A, Popescu C. Machine Learning Models for SQL Injection Detection. Electronics (Switzerland) 2025;14. https://doi.org/10.3390/electronics14173420.
[8] Tadhani JR, Vekariya V, Sorathiya V, Alshathri S, El-Shafai W. Securing web applications against XSS and SQLi attacks using a novel deep learning approach. Sci Rep 2024;14. https://doi.org/10.1038/s41598-023-48845-4.
[9] Alsarhan A, Hussein F, Moh S, El-Salhi FS. The Effect of Preprocessing Techniques, Applied to Numeric Features, on Classification Algorithms' Performance. Data (Basel) 2021. https://doi.org/10.3390/data6020011.
[10] Chen Q, Banerjee A, Demiralp Ç, Durrett G, Dillig I. Data Extraction via Semantic Regular Expression Synthesis. Proceedings of the ACM on Programming Languages 2023;7. https://doi.org/10.1145/3622863.
[11] McKenzie J, Rajapakshe R, Shen H, Rajapakshe S, Lin A. A Semiautomated Chart Review for Assessing the Development of Radiation Pneumonitis Using Natural Language Processing: Diagnostic Accuracy and Feasibility Study. JMIR Med Inform 2021;9. https://doi.org/10.2196/29241.
[12] SQL injection cheat sheet n.d. https://portswigger.net/web-security/sql-injection/cheat-sheet (accessed September 18, 2025).
[13] Cross-site scripting (XSS) cheat sheet n.d. https://portswigger.net/web-security/cross-site-scripting/cheat-sheet (accessed September 18, 2025).

[14] Jin Z. Principle, Methodology and Application for Data Cleaning techniques. BCP Business & Management FIBA 2022;26. https://doi.org/10.54691/bcpbm.v26i.2032.
[15] Lan F. Research on Text Similarity Measurement Hybrid Algorithm with Term Semantic Information and TF-IDF Method. Advances in Multimedia 2022. https://doi.org/10.1155/2022/7923262.
[16] Firmansyah MR, Astuti YP. Stroke Classification Comparison with KNN through Standardization and Normalization Techniques. Advance Sustainable Science, Engineering and Technology 2024;6. https://doi.org/10.26877/asset.v6i1.17685.
[17] Kamal H. Advanced Hybrid Transformer-CNN Deep Learning Model for Effective Intrusion Detection Systems with Class Imbalance Mitigation Using Resampling Techniques. Future Internet 2024;16. https://doi.org/10.3390/fi16120481.
[18] Zakariah M, AlQahtani SA, Al-Rakhami MS. Machine Learning-Based Adaptive Synthetic Sampling Technique for Intrusion Detection. Applied Sciences (Switzerland) 2023;13. https://doi.org/10.3390/app13116504.
[19] Ratnasari AP. Performance of Random Oversampling, Random Undersampling, and SMOTE-NC Methods in Handling Imbalanced Class in Classification Models. International Journal of Scientific Research and Management (IJSRM) 2024;12:494–501. https://doi.org/10.18535/ijsrm/v12i04.m03.
[20] Yustanti W, Iriawan N, Irhamah. Categorical encoder based performance comparison in preprocessing imbalanced multiclass classification. Indonesian Journal of Electrical Engineering and Computer Science 2023;31:1705–15. https://doi.org/10.11591/ijeecs.v31.i3.pp1705-1715.

BIBLIOGRAPHY OF AUTHORS

Adi Septian is a student at Universitas Widyatama specializing in Applied Networking. Born in Bandung on September 30, 1990, his academic interests include network management, server security, and virtualization technologies for IT infrastructure optimization. He currently serves as a Network and Server Administrator at a private university in Indonesia, where he applies his technical expertise to support research and educational systems.

Atep Aulia Rahman, born on December 9, 1983, in Bandung, Indonesia, is a Lektor and specialist lecturer in IT networking. He holds a Master's in Information Systems from STMIK LIKMI Bandung (2017), a Bachelor's in Computer Engineering from STMIK LPKIA Bandung (2005–2007), and a Diploma from Politeknik LPKIA Bandung (2002–2005). Currently teaching at Universitas Widyatama (2021–present) in subjects including Linux System Administration, Computer Networking (Cisco & Mikrotik), Network Security, and Digital Forensics, he previously lectured at Telkom University (2023) and PKN STMIK LPKIA (2014–2023). Rahman also serves as a professional trainer for Cisco, Mikrotik, AI, and Cyber Security certifications at institutions like B One Corporation, OSTech Indonesia, and BNSP.