INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Comparative Analysis of Machine Learning Algorithms for Cross-Site Scripting (XSS) Attack Detection

Khairatun Hisan Hamzah [a], Mohd Zamri Osman [a,*], Tumusiime Anthony [a], Mohd Arfian Ismail [b], Zubaile Abdullah [c], Alde Alanda [d]

[a] *Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Johor Bahru, Malaysia*
[b] *Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Pahang, Malaysia*
[c] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Johor, Malaysia*
[d] *Department of Information Technology, Politeknik Negeri Padang, Padang, Indonesia*
Corresponding author: [*]mohdzamri.osman@utm.my

***Abstract*—Cross-Site Scripting (XSS) attacks pose a significant cybersecurity threat by exploiting vulnerabilities in web applications to inject malicious scripts, enabling unauthorized access and execution of malicious code. Traditional XSS detection systems often struggle to identify increasingly complex XSS payloads. To address this issue, this research evaluated the efficacy of Machine Learning algorithms in detecting XSS threats within online web applications. The study conducts a comprehensive comparative analysis of XSS attack detection using four prominent Machine Learning algorithms, which consist of Extreme Gradient Boosting (XGBoost), Random Forest (RF), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). This research utilizes a comparative methodology to assess the selected Machine Learning algorithms by analyzing their performance metrics, including confusion matrix, 10-fold cross-validation, and assessment of training time to thoroughly evaluate the models. By exploring dataset characteristics and evaluating the performance metrics of each selected algorithm, the study determined the most robust Machine Learning solution for XSS detection. Results indicate that Random Forest is the top performer, achieving 99.93% accuracy and balanced metrics across all criteria evaluated. These findings will significantly enhance web application security by providing reliable defenses against evolving XSS threats.**

***Keywords*—Cross Site Scripting (XSS); machine learning; RF; XGBoost; KNN; SVM; cybersecurity; web application security.**

Manuscript received 5 Mar. 2024; revised 17 Jul. 2024; accepted 24 Sep. 2024. Date of publication 30 Nov. 2024.

I. Introduction

Cross-site scripting (XSS) remains a challenging threat in cybersecurity, exploiting vulnerabilities in online web applications to inject malicious scripts into web pages. XSS is a web security vulnerability where attackers inject malicious scripts into trusted websites, exploiting the site's failure to validate or encode user input properly. This poses a significant risk to users, enabling attackers to gain unauthorized access to sensitive information and execute malicious code. The traditional XSS detection system needs to be improved, considering the increasingly diverse forms of XSS payloads [2]. OWASP's 2021 report indicates that 94% of the applications tested are susceptible to injection vulnerabilities, with 33 Common Weakness Enumerations (CWEs) falling into this category [1]. Traditional methods for detecting cross-site scripting (XSS) focus on signature-based approaches, which involve investigating known attack patterns. As a result, online web applications and users that utilize traditional methods are left vulnerable. This calls for a dynamic and adaptive solution that can overcome the constantly evolving payloads of XSS.

This research implemented a machine learning (ML) approach to XSS detection to address the increasing complexity of XSS payloads. The study focuses on utilizing four prominent algorithms, specifically Extreme Gradient Boosting (XGBoost), Random Forest (RF), K-Nearest Neighbors (KNN), and Support Vector Machine (SVM). Each model is carefully tuned to enhance its ability to distinguish between malicious scripts and benign code. Our study aims to comprehensively analyze these algorithms to determine the most effective model for robust XSS detection in web applications.

The performance of each model is evaluated based on multiple metrics, including training time, confusion matrix, and 10-fold cross-validation. By assessing these metrics, the

study aims to identify the optimal approach for XSS detection that can adapt to the evolving nature of web-based attacks. The findings of this research are expected to contribute significantly to the advancement of XSS detection models and provide valuable insights for enhancing cybersecurity in the digital realm.

The findings propose recommendations for enhancing the accuracy and performance of the XSS detection model by selecting the most effective approach for classifiers to identify various types of XSS attacks.

A. Type of XSS Attack

Distinguishing XSS attacks can be exceedingly challenging due to the concealed aspect of the malicious script, whether it happens to be on the server side or the client side. Within the framework of an XSS attack, it is possible to classify the threats into three distinct types: Stored XSS, Reflected XSS, and DOM-based XSS [3]. Each type of XSS attack demonstrates the diverse methods attackers use to exploit web application vulnerabilities. Understanding these XSS variants' mechanisms and potential impacts is expected to protect web applications and their users against these persistent threats.

B. XSS Detection Model Machine Learning-based

XSS attacks can be prevented by employing a machine-learning algorithm in an XSS detection model [4]. This model aims to differentiate the dataset for the proposed approach effectively. Differentiating between XSS attacks and non-XSS inputs, which are regular web application inputs. These standard inputs can include text, numbers, and other types of inputs, including combinations of these elements. Hence, it is the one to monitor and detect XSS attacks on web application inputs.

C. Related Studies

The related research on XSS attack detection using an ML-based model includes efforts to understand the analysis of XSS attack patterns, applying machine learning algorithms for XSS attack identification, and evaluating various ML algorithms for XSS detection. Research on XSS attack patterns has explored various aspects, including XSS attack payloads' characteristics, XSS attack methods' evolution, and the impact of XSS attacks on web applications. TABLE I shows that the studies provide valuable insights into the nature of XSS attacks, making way for more effective detection strategies [3], [5], [7], [9]. A proposed fusion verification method that combines traffic detection and XSS payload detection. The approach, utilizing Random Forest and a novel Web Application Intrusion Detection Prevention Firewall System (WAIDPFS), demonstrated superior real-time detection capabilities [15].

A comprehensive evaluation of multiple machine learning algorithms was performed on a dataset comprising 13,686 instances. The analysis focused on the efficacy of AdaBoost, Random Forest, Decision Tree, SVM, KNN, Logistic Regression, and XGBoost. Findings revealed that AdaBoost, Random Forest, and Decision Tree exhibited superior performance regarding accuracy and F1-score [16].

A hybrid feature methodology integrating n-gram modeling and feature selection techniques was proposed. Utilizing logistic regression on 16,361 samples, the method attained exceptional accuracy with minimal false positives. This hybrid strategy exhibited enhanced efficacy relative to standalone linguistic and feature selection techniques [17]. The n-gram was also studied comprehensively in [20] for email spam detection. In [21], a hybrid method for phishing attack detection was employed for better performance.

An extensive evaluation of multiple ML models, including Random Forest, XGBoost, and ensemble methods. Their study utilized a large dataset of 138,569 samples and incorporated feature selection techniques. The Random Forest model achieved high accuracy, while their ensemble models combining Random Forest with Decision Trees and Gradient Boosting also showed high performance [18]. The Isolation Forest, meanwhile, was deployed to detect diabetes mellitus, reducing the complexity of staking.

A comparative analysis of five ML algorithms using a Kaggle dataset. The study evaluated AdaBoost, XGBoost, Decision Tree, Logistic Regression, and Naive Bayes. Among these, AdaBoost demonstrated the highest accuracy. AdaBoost also excelled in precision, specificity, and F1-score, further establishing its effectiveness for XSS attack detection. [19].

TABLE I
RELATED RESEARCH PAPERS

Ref.	Description	Result
[9]	Used XGBoost in a hybrid learning approach for XSS detection.	Hybrid (XGBoost+RF):96.3% XGBoost: 94.8% RF: 93.6%, SVM: 91.9% KNN: 89.2%
[3]	Compared XGBoost, RF, KNN, and SVM for XSS detection.	XGBoost: 95.2% RF: 93.8%, SVM: 92.1%, KNN: 89.7%
[7]	Compared KNN and SVM for XSS detection.	SVM: 93.2%, KNN: 88.5%
[5]	Compared XGBoost, KNN, RF, and SVM for XSS detection.	RF: 94.5%, XGBoost: 94.1% SVM: 91.7%, KNN: 88.9%
[15]	Proposed Random Forest, WAIDPFS	Random Forest: 99.91% accuracy
[16]	Compared AdaBoost, Random Forest, Decision Tree, SVM, KNN, LR, XGBoost	AdaBoost: 99.69%, Random Forest:99.67%
[17]	Proposed Logistic Regression, N-gram, Feature Selection	Logistic Regression: 99.87% accuracy, 0.039% false positive rate
[18]	Evaluated Random Forest, XGBoost, Decision Trees, Gradient Boosting, MLP, Ensemble Learning	Random Forest: 99.78%, Ensemble (RF+DT+GB): 99.76%, Ensemble (RF+MLP): 99.65%
[19]	Used AdaBoost, XGBoost, Decision Tree, Logistic Regression, Naive Bayes	AdaBoost: 97.92%, XGBoost: 96.82%, Decision Tree: 95.76%, Logistic Regression: 94.41%, Naive Bayes: 86.89%.

D. Proposed Solutions

This research proposes the use of machine learning algorithms, specifically XGBoost, RF, SVM, and KNN for detecting XSS attacks. From the summarizations of related studies, as shown in Table II, XGBoost is the superior algorithm based on its performance in XSS detection, and it will be one of the selected algorithms for the proposed solution. However, this research will include a comparative analysis between other prominent machine learning algorithms to assess their effectiveness in identifying XSS attacks. In addition to XGBoost, other machine learning

algorithms such as RF, SVM, and KNN will be explored as part of the comparative analysis.

TABLE III
COMPARATIVE OF RESEARCH PAPERS

Selected ML Algorithms	Advantages	Disadvantages
KNN	Simple and effective for small datasets, it handles multi-class classification well [10]	Computationally expensive with large datasets, sensitive to irrelevant features [10].
SVM	Effective in high-dimensional spaces, robust to overfitting [3], [7].	It requires careful tuning of parameters and is computationally intensive with large datasets [3], [10].
XGBoost	High performance, handles missing data, scalable [2], [14].	It requires careful tuning and complex implementation [14].
RF	High accuracy, handles non-linear data, reduces overfitting [5], [7].	Computationally intensive, less interpretable [10][3].

E. Extreme Gradient Boosting (XGBoost)

XGBoost's iterative learning method constructs an ensemble of decision trees that leverage knowledge acquired from previous iterations. This iterative nature empowers XGBoost to continually enhance its accuracy in predicting XSS attacks [14]. Regardless of the volume and complexity of XSS data, XGBoost shows a high level of preparedness to process the information efficiently and reveal concealed patterns. This dual capability of iterative learning sets XGBoost as an outstanding algorithm for XSS detection [3].

F. Random Forest (RF)

Theoretically, each tree in the forest is trained on a random subset of the data, and the final prediction is determined by combining the predictions of individual trees. RF functions as an ensemble technique, successfully preventing overfitting and exhibiting strong performance with diverse data sources. An outstanding feature is its capacity to analyze complex data sets with several dimensions, which allows it to be versatile in detecting XSS in various settings [5].

G. K-Nearest Neighbors (KNN)

KNN offers a more straightforward approach. It is a non-parametric algorithm for classification. It assigns an object to the class most common among its k-nearest neighbors, where k is a user-defined parameter. KNN can be computationally expensive for large datasets and needs help dealing with noisy or imbalanced data.

H. Support Vector Machine (SVM)

SVM can effectively handle high-dimensional data. It works by finding the hyperplane that best separates the classes of data points in the feature space. This separation margin is maximized to ensure optimal classification performance. Despite its high computational cost, it has a strong theoretical foundation and can generalize well to unseen data, making it valuable for XSS detection.

I. Support Vector Machine (SVM)

The selected dataset "XSS_dataset.csv," obtained from the Kaggle platform [6], is a suitable and deliberate choice for the literature evaluation in this study context. The dataset's specific nomenclature, which clearly indicates its emphasis on XSS threats, perfectly matches the study goal of training machine learning models to detect XSS. Conclusively, the validity and importance of this dataset in evaluating machine learning methods for XSS detection result from its specific focus on XSS attacks and applicability for training machine learning models.

II. MATERIAL AND METHOD

This section thoroughly explains the research methods employed in planning and evaluating the experiment. Additionally, the research workflow will be outlined to clarify the methodologies proposed at each stage of the research process. The chapter justifies the tools, datasets, and procedures used to carry out the experiment for the comparative analysis. A list of the performance metrics used in this investigation is also included.

A. Workflow

This research follows a three-stage workflow, each aligned to specific research objectives. In the initial stage, data preparation and cleaning processes are executed, followed by a rigorous assessment and examination of methods and attributes. In the second stage, the selected machine learning algorithms are applied to train and test the model. Finally, in the third stage, the experimental results are thoroughly analyzed and discussed in Fig. 1. The research framework is covered in three phases:

1) Phase 1: Phase 1 (Recognition of Dataset & Data Preparation to Train Machine Learning Classifiers): Understanding existing machine learning algorithm findings from literature review and exploring a dataset for machine learning detection model.

2) Phase 2: Phase 2 (Development of Proposed XSS Detection and Classification Model): Create an efficient model for detecting XSS attacks using machine learning algorithms XGBoost, RF, SVM, and KNN. Model training and testing.

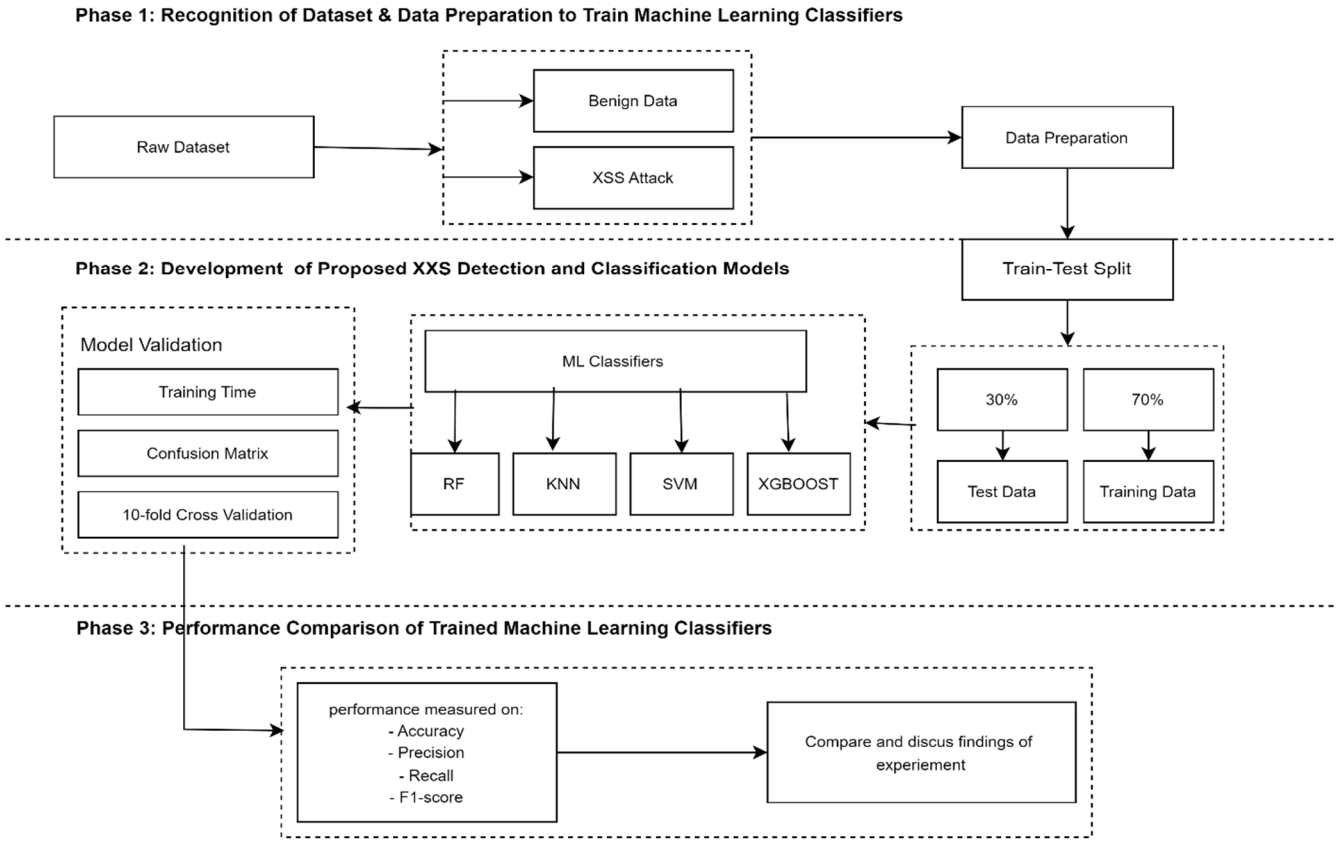

Fig. 1 Flowchart of proposed work

3) Phase 3: Phase 3 (Performance comparison of trained Machine Learning Classifiers): The performance comparison findings demonstrate the efficacy of the suggested XSS detection model in terms of performance metrics measurement. The comparison analysis will uncover insights into the most effective model for XSS detection.

B. Data Labelling

The dataset has 13,686 raw data that has been prepared with two primary columns, which are "Sentence" and "Label" [6]. The "Sentence" column contains textual data in the form of scripts, including both benign and malicious occurrences associated with XSS attacks. The "Label" column assigns binary values, "0" and "1," to each script, indicating the lack or existence of XSS attacks, accordingly. Fig. 2 shows benign data, which is "Label 0," is 6,316, while malicious data, which is "Label 1," is 7,373.

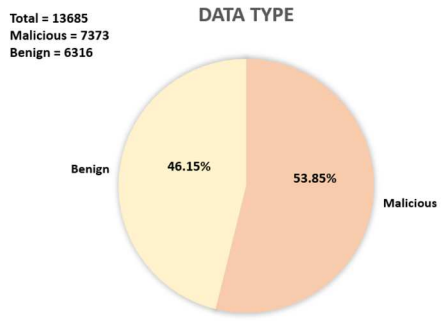

Fig. 2 Distribution of dataset

The labeling outline is straightforward. "Label 0" likely indicates cases without possible XSS attacks, representing safe scripts. In contrast, "Label 1" signifies the existence of XSS attacks, explicitly referring to malicious scripts. This binary classification enables the training of machine learning models to differentiate between these two categories.

C. Performance Measurement

This section analyzes the performance metrics used to facilitate future comparative analysis. First, training time is calculated by calculating how long it takes to train machine learning using the selected dataset. This step is essential for assessing the machine learning classifier's efficiency because a shorter training period will result in lower computational costs. The training time can be computed using the formula below.

$$Training\ Time\ (TT) = \frac{Number\ of\ hosts\ (Nh)}{final\ Time\ (fT) - initial\ Time\ (iT)} \qquad (1)$$

Second, the confusion matrix is a concept, and data related to the Confusion Matrix are true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Based on Table III, FN is the model that incorrectly classifies positive events as negative. TN is the model that accurately identifies negative instances as negative. TP is a model that accurately identifies the number of positive cases. Meanwhile, FP is the frequency with which the model incorrectly identified a negative case as a positive example.

TABLE III
CONFUSION MATRIX

	Actually Positive (1)	**Actually Negative (0)**
Predicted Positive (1)	True Positive (TP)	False Positive (FP)
Predicted Negative (0)	False Negative (FN)	True Negative (TN)

Then, using this data, accuracy, precision, recall, and F1-score are calculated. These will be compared to previous models to evaluate this proposed XSS detection model's performance. Accuracy is calculated by dividing the number of predictions the model makes by the number of correct predictions.

$$Accuracy = \frac{(TN + TP)}{(TN + FP + TN + FN)} \quad (2)$$

Precision is the percentage of total positive predictions; it expresses the model's percentage of true positive predictions.

$$Precision = \frac{TP}{(TP + FP)} \quad (3)$$

Recall is the total number of positive cases in the dataset; it computes the detection model's real positive prediction rate.

$$Recall = \frac{TP}{(TP + TN)} \quad (4)$$

F1-score is the data that combines recall and accuracy of a model into a single score.

$$F1\ score = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \quad (5)$$

Cross-validation is a technique known as 10-fold cross-validation that involves training and testing a machine learning model on several subsets of a dataset to assess its performance. In a series of ten iterations, the dataset is resampled into ten equal-sized folds, of which nine are used for training and one for testing. To visualize the iteration, show how it works.

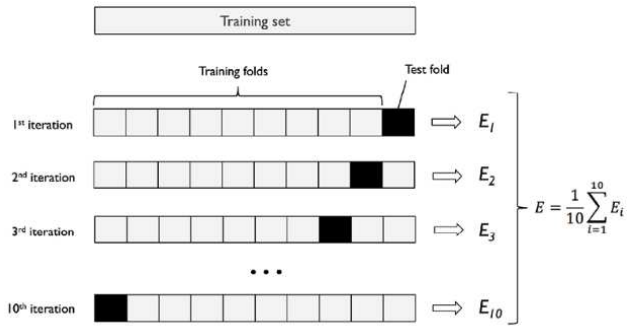

Fig. 3 10-fold cross-validation

D. Research Design and Implementation

This section provides an overview of the research methodology, which involves selecting the appropriate software and measurements to assess the effectiveness of machine learning classifiers in detecting and classifying XSS attacks. It also explains the machine learning classifiers that have been trained, assessed, and compared.

E. Experiment Setup

The experiment is based on a Python environment. A personal laptop running Jupyter Notebook with Anaconda Navigator is utilized. The exploratory data analysis is conducted by analyzing the raw dataset obtained from Kaggle [6].

F. Experiment Design

To effectively achieve the research objectives, the experiment's design provides a comprehensive explanation of the implementation procedure, and the tools utilized in the experiment. This includes detailing how the dataset was prepared, the specific machine learning models chosen, and the evaluation metrics employed to assess their performance in detecting Cross-Site Scripting (XSS) vulnerabilities:

1) Data Preprocessing: The experiment begins with loading the 'XSS_dataset.csv' dataset into a Panda DataFrame (df). Each entry contains sentences labeled for Cross-Site Scripting (XSS) vulnerabilities. As observed initially, the raw dataset contains various forms of HTML tags and JavaScript snippets. The preprocessing stage involves transforming the raw dataset into a cleaned version where significant elements like HTML tags and JavaScript event handlers are systematically removed. This cleaning process consists of several steps, such as converting all text to lowercase for uniformity, reducing case sensitivity, and tokenizing sentences into individual words to filter out non-informative words.

2) Vectorization Data: In this experiment, the text data undergoes vectorization using the CountVectorizer from the sci-kit-learn library, a crucial step in natural language processing (NLP) tasks. This process transforms textual data into a numerical format suitable for machine learning algorithms. **Error! Reference source not found.** includes the output result, which displays the transformed data as a NumPy array (dtype=int64), where each element represents the frequency count of a specific term in its corresponding document. This numerical representation allows machine learning models to process and learn from the textual data effectively.

```
X

array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

Fig. 4 Data after vectorization

3) Splitting Data and Model Development Execution Setup: The performance of the models in detecting XSS threats was assessed, the dataset was ratioed precisely 70% of the data was allocated for training, and the remaining 30% was reserved for testing. The splitting process was executed with a fixed random state to ensure reproducibility of the results. The function begins by recording the start time to calculate the duration of the training. It then performs 10-fold cross-validation on the training data to estimate the model's

performance stability. After cross-validation, the model is trained on the entire training set, and the end time is recorded to determine the total training duration. The trained model is then used to predict the labels of the test set, and several performance metrics are computed, such as accuracy, precision, recall, and F1 score. Then, the function evaluate_model is applied to four different classifiers.

4) Model Evaluation: Throughout the model evaluation phase, each model was tested thoroughly to determine how well it detected XSS vulnerabilities. The evaluate_model function was used with the models' classifiers and the training and testing datasets. This function provided several performance metrics, including cross-validation scores, accuracy, precision, recall, F1 score, confusion matrix, and training time.

5) Model Selection: A ranking approach was applied using the performance metrics stored in the results_df DataFrame to determine the most effective model for detecting XSS vulnerabilities. This method allows DataFrame to assess and compare the performance of various classifiers based on metric measurements. It is to identify and present the metrics of the best-performing model.

III. Result and Discussion

This section provides the results and discussion, along with an analysis of the experiments conducted. The proposed models are evaluated in terms of performance metrics and cross-validation in detecting Cross-Site Scripting (XSS) threats, specifically in online web applications. This chapter also provides an overview of the most effective model identified from the evaluation and offers insights for future research.

A. Result of Confusion Matrix

The study evaluates the performance of four key algorithms: Random Forest, XGBoost, K-Nearest Neighbors, and SVM. Each model is trained and evaluated using accuracy, precision, recall, and F1-score metrics. To get the metrics being assessed, each model needs to go through the confusion matrix

TABLE IV
CONFUSION MATRIX FOR RANDOM FOREST

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	1919	0
Predicted Negative (0)	3	2184

TABLE V
CONFUSION MATRIX FOR XGBOOST

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	1917	2
Predicted Negative (0)	7	2180

TABLE VI
CONFUSION MATRIX FOR KNN

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	1912	7
Predicted Negative (0)	12	2175

TABLE VII
CONFUSION MATRIX FOR SVM

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	1919	0
Predicted Negative (0)	9	2178

TABLE VIII
RESULT OF EVALUATION USING PERFORMANCE METRICS

Metric	Random Forest	XGBoost	K-Nearest Neighbors	SVM
Test Accuracy	0.9993	0.9978	0.9953	0.9978
Precision	1.0000	0.9990	0.9967	1.0000
Recall	0.9986	0.9967	0.9945	0.9958
F1-Score	0.9993	0.9979	0.9956	0.9979

It can be concluded that XGBoost demonstrates a balanced performance with a notable accuracy rate (0.9978). At the same time, Random Forest shows superior results, likely due to its ensemble approach that combines multiple decision trees for enhanced prediction. Despite its simplicity, K-Nearest Neighbors performs competitively, underscoring its efficiency in handling text classification tasks. SVM also shows high performance, although its training time is significantly longer. The confusion matrices measure each model's strengths and weaknesses in predicting XSS threats.

B. Result of Cross-Validation

Cross-validation results provide a robust evaluation of the models by partitioning the data into subsets, training the model on some subsets while validating on others, and repeating this process to ensure that the evaluation metrics are not biased by a particular data split. The 10-fold cross-validation accuracy for each model is detailed, showing the stability of the algorithms.

TABLE IX
CROSS-VALIDATION FOR MODELS

Metric	Random Forest	XGBoost	K-Nearest Neighbors	SVM
10-fold CV Accuracy	0.9979	0.9960	0.9929	0.9942
Test Accuracy	0.9993	0.9978	0.9953	0.9978

C. Result of Training Time

Training time is important, especially for large datasets or real-time applications. The training time for each model is recorded and compared to highlight the models' efficiency.

TABLE X
TRAINING TIME RESULTS FOR MODELS

Metric	Random Forest	XGBoost	K-Nearest Neighbors	SVM
Training Time (s)	80.616	36.221	11.698	544.12

D. Comparison and Result Discussion

The algorithms' comparative analysis focuses on the advantages and limitations of each approach in the context of XSS detection models.

TABLE XI
TRAINING TIME RESULT FOR MODELS

Metric	Random Forest	XGBoost	K-Nearest Neighbors	SVM
10-fold CV Accuracy	0.9979	0.9960	0.9929	0.9942
Test Accuracy	0.9993	0.9978	0.9953	0.9978
Precision	1.0000	0.9990	0.9967	1.0000
Recall	0.9986	0.9967	0.9945	0.9958
F1-Score	0.9993	0.9979	0.9956	0.9979
Training Time (s)	80.616	36.221	11.698	544.12
Metric	Random Forest	XGBoost	K-Nearest Neighbors	SVM

Referring to Table XI, Random Forest emerges as the top performer with an accuracy of 99.93%, benefiting from its ability to manage complex decision boundaries through ensemble learning. XGBoost follows closely with an accuracy of 99.78%, showcasing its efficacy in handling linear and non-linear relationships. K-Nearest Neighbors (KNN), achieving an accuracy of 99.53%, is a valuable model due to its simplicity and computational efficiency. SVM also shows strong performance, though its training time is significantly longer, which could be a drawback for time-sensitive applications.

IV. CONCLUSION

This research provides a comprehensive summary of the primary objectives, methodologies, and techniques employed in the comparative analysis of machine learning algorithms for detecting Cross-Site Scripting (XSS) threats in online web applications The fast-changing nature of web security requires effective detection systems. This research aimed to determine how well different machine learning models can identify XSS attacks and which model is the most effective. The study used thorough research and careful evaluation to improve understanding and application of machine learning in web security

Based on the findings and constraints of this research, several suggestions are made for future improvements and further research. Future work should explore advanced feature extraction techniques like word embeddings and deep learning-based methods to capture more detailed patterns in the data. Expanding the dataset to include more samples and a wider variety of XSS attack patterns will help the models generalize better. Using techniques like grid search, random search, or Bayesian optimization for more extensive hyperparameter tuning can improve model performance.

REFERENCES

[1] OWASP. OWASP Top Ten. Retrieved from Owasp.org website: https://owasp.org/www-project-top-ten/. 2021.

[2] F. M. M. Mokbal, W. Dan, W. Xiaoxi, Z. Wenbin, and F. Lihua, "XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization," Journal of Information Security and Applications, vol. 58, p. 102813, May 2021, doi:10.1016/j.jisa.2021.102813.

[3] P. Roy, R. Kumar, P. Rani, and T. S. Joy, "XSS: Cross-site Scripting Attack Detection by Machine Learning Classifiers," 2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART), pp. 1535–1539, Dec. 2022, doi:10.1109/smart55829.2022.10046960.

[4] I. K. Thajeel, K. Samsudin, S. J. Hashim, and F. Hashim, "Machine and Deep Learning-based XSS Detection Approaches: A Systematic Literature Review," Journal of King Saud University - Computer and Information Sciences, vol. 35, no. 7, p. 101628, Jul. 2023, doi:10.1016/j.jksuci.2023.101628.

[5] R. Banerjee, A. Baksi, N. Singh, and S. K. Bishnu, "Detection of XSS in web applications using Machine Learning Classifiers," 2020 4th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), pp. 1–5, Oct. 2020, doi:10.1109/iementech51367.2020.9270052.

[6] S. H. Shah and S. S. Hussain, "Cross site scripting (XSS) dataset for deep learning," Kaggle, Jan. 11, 2024. [Online]. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning/data.

[7] B. Gogoi, T. Ahmed, and H. K. Saikia, "Detection of XSS Attacks in Web Applications: A Machine Learning Approach," International Journal of Innovative Research in Computer Science & Technology, vol. 9, no. 1, pp. 1–10, Jan. 2021, doi:10.21276/ijircst.2021.9.1.1.

[8] J. Kaur, U. Garg, and G. Bathla, "Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review," Artificial Intelligence Review, vol. 56, no. 11, pp. 12725–12769, Mar. 2023, doi:10.1007/s10462-023-10433-3.

[9] Q. Abu Al-Haija, "Cost-effective detection system of cross-site scripting attacks using hybrid learning approach," Results in Engineering, vol. 19, p. 101266, Sep. 2023, doi:10.1016/j.rineng.2023.101266.

[10] A. E. Mohamed, "Comparative study of four supervised machine learning techniques for classification", Int. J. Appl. Sci. Technol., vol. 7, no. 2, pp. 5-18, 2017.

[11] A. Hannousse, S. Yahiouche, and M. C. Nait-Hamoud, "Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey," Computer Science Review, vol. 52, p. 100634, May 2024, doi: 10.1016/j.cosrev.2024.100634.

[12] B. Panda, D. Chaturya, I. Sahil, C. Dinesh, and A. Prakash, "Hazard Identification and Detection using Machine Learning," Shu Ju Cai Ji Yu Chu Li/Journal of Data Acquisition and Processing, vol. 38, pp. 4418–4427, 2023. doi:10.5281/zenodo.7766139.

[13] A. W. Marashdih, Z. F. Zaaba, K. Suwais, and N. A. Mohd, "Web Application Security: An Investigation on Static Analysis with other Algorithms to Detect Cross Site Scripting," Procedia Computer Science, vol. 161, pp. 1173–1181, 2019, doi:10.1016/j.procs.2019.11.230.

[14] T. Chen and C. Guestrin, "XGBoost," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, Aug. 2016, doi: 10.1145/2939672.2939785.

[15] J. Lu, Z. Wei, Z. Qin, Y. Chang, and S. Zhang, "Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning," Mathematics, vol. 10, no. 20, p. 3787, Oct. 2022, doi:10.3390/math10203787.

[16] J. Harish Kumar and J. J Godwin Ponsam, "Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis," 2023 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–9, Jan. 2023, doi:10.1109/iccci56745.2023.10128470.

[17] D. A. Prasetio, K. Kusrini, and M. R. Arief, "Cross-site Scripting Attack Detection Using Machine Learning with Hybrid Features," Jurnal Infotel, vol. 13, no. 1, pp. 1–6, Feb. 2021, doi:10.20895/infotel.v13i1.606.

[18] R. Alhamyani and M. Alshammari, "Machine Learning-Driven Detection of Cross-Site Scripting Attacks," Information, vol. 15, no. 7, p. 420, Jul. 2024, doi: 10.3390/info15070420.

[19] A. Kumar and I. Sharma, "Performance Evaluation of Machine Learning Techniques for Detecting Cross-Site Scripting Attacks," 2023 11th International Conference on Emerging Trends in Engineering & Technology - Signal and Information Processing (ICETET - SIP), pp. 1–5, Apr. 2023, doi: 10.1109/icetet-sip58143.2023.10151468.

[20] E. H. Tusher, M. A. Ismail, M. A. Rahman, A. H. Alenezi, and M. Uddin, "Email Spam: A Comprehensive Review of Optimize Detection Methods, Challenges, and Open Research Problems," IEEE Access, vol. 12, pp. 143627–143657, 2024, doi:10.1109/access.2024.3467996.

[21] N. F. Idris, M. A. Ismail, M. I. M. Jaya, A. O. Ibrahim, A. W. Abulfaraj, and F. Binzagr, "Stacking with Recursive Feature Elimination-Isolation Forest for classification of diabetes mellitus," PLOS ONE, vol. 19, no. 5, p. e0302595, May 2024, doi:10.1371/journal.pone.0302595.

[22] N. S. Nordin and M. A. Ismail, "A hybridization of butterfly optimization algorithm and harmony search for fuzzy modelling in phishing attack detection," Neural Computing and Applications, vol. 35, no. 7, pp. 5501–5512, Nov. 2022, doi: 10.1007/s00521-022-07957-0.