International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-9, Issue-1, January 2021
https://doi.org/10.21276/ijircst.2021.9.1.1
Article ID D10962, Pages 1-10
www.ijircst.org

Detection of XSS Attacks in Web Applications: A Machine Learning Approach

Bronjon Gogoi,[1] Tasiruddin Ahmed,[2] and Hemanta Kumar Saikia[3]

[1,2,3] Scientist, Regional Centre of Excellence for Application Security,
National Informatics Centre, Guwahati, Assam, India

Correspondence should be addressed to Bronjon Gogoi; bronjon.du@gmail.com

ABSTRACT- With the increased use of the internet, web applications and websites are becoming more and more common. With the increased use, cyber-attacks on web applications and websites are also increasing. Of all the different types of cyber-attacks on web applications and websites, XSS (Cross-Site Scripting) attacks are one of the most common forms of attack. XSS attacks are a major problem in web security and ranked as number two web application security risks in the OWASP (Open Web Application Security Project) Top 10. Traditional methods of defence against XSS attacks include hardware and software-based web application firewalls, most of which are rule and signature-based. Rule-based and signature-based web application firewalls can be bypassed by obfuscating the attack payloads. As such, rule-based and signature-based web application firewalls are not effective against detecting XSS attacks for payloads designed to bypass web application firewalls. This paper aims to use machine learning to detect XSS attacks using various ML (machine learning) algorithms and to compare the performance of the algorithms in detecting XSS attacks in web applications and websites.

KEYWORDS- Web Application, XSS Attacks, Machine Learning

I. INTRODUCTION

In today's digital world, a web application is the most cost-effective mechanism of software delivery and used by millions of businesses and other organizations to deliver their services over the internet. Any user equipped with a computer and an internet connection can access web applications. This easy accessibility of web applications also makes them vulnerable to a wide range of attacks from malicious users. Among many categories of attack, the XSS attack ranks as the number two web application security risk. Figure. 1 shows the rank of XSS attacks in comparison to other attacks. XSS vulnerabilities are easily exploitable as many freely available tools are there to enable anyone with minimum knowledge to attack web applications. XSS attacks can lead to session hijacking, sensitive data disclosure, cross-site request forgery (csrf) attacks, and other security vulnerabilities including impersonation of the victim. XSS attacks can also lead to code execution on the server, depending on the application and privileges of the user's account [1]. At present, web applications adopt an XSS prevention or XSS detection and prevention approach. In XSS prevention approach, it is the responsibility of the programmer or developer to adopt the necessary means to prevent XSS attacks.

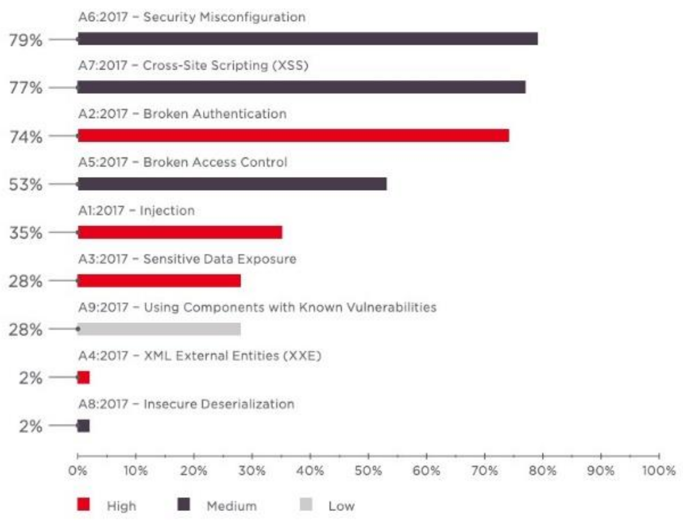

Fig. 1: XSS Attacks Compared to others

Developers use filtering, input and output encoding to prevent XSS injection attacks [1]. Developers also use automated tools to perform static, dynamic or hybrid analysis to detect XSS vulnerabilities in web applications and to take effective measures to defend against XSS attacks. In XSS detection and prevention approach, WAFs (Web Application Firewall) are used as a part of XSS detection and prevention approach [2]. In WAF based approach, WAFs are deployed outside the web application servers. WAFs analyse all inbound traffic, detects attacks using policy rules or expressions, and block detected attacks. In static analysis, the source code of the application is analysed using automated tools to discover vulnerabilities in the source code. Dynamic analysis tests the application by executing tests in real time. The hybrid analysis combines both static and dynamic analysis to detect injection attacks. The above-mentioned approaches of XSS detection and prevention has drawbacks. In the first approach of XSS detection using secure implementation, the responsibility of defending against XSS attacks rests on the developers. There is no way of defending against developer carelessness and hence can lead to XSS vulnerabilities in the code, which can be later exploited. Moreover, using static, dynamic and hybrid analysis also has their drawbacks. The disadvantage of static analysis is that results generated by static analysis tools have a high False Positive Rate [2] and static analysis tools does not support all programming languages. Dynamic analysis tools have the disadvantage that they do not cover all execution paths and produce high false positives and false negatives [3]. The hybrid analysis combines both static and dynamic analysis to detect injection attacks. Hybrid analysis has the disadvantage that the analysis process is cumbersome and time-consuming [4]. WAFs have the drawback that most WAF's are signature or rule based. Signature or rule based WAF's cannot detect zero day attacks and cannot detect obfuscated XSS attacks [2]. There also lies the possibility of improperly configured rules or signatures that can cause the WAF to fail to detect XSS attacks. In this paper, we propose a machine learning based WAF to detect the most common XSS attacks in web applications. Our approach includes the implementation of a Web Application Firewall that uses machine learning to detect XSS attacks. The remainder of this paper is organized as follows: In section II, we discuss XSS attack types work; in section III, we discuss related work; in section IV, we discuss our proposed approach; in section V we draw our conclusion and present the future scope of this study.

II. XSS ATTACKS

XSS scripting is a type of injection attack where malicious scripts are inserted into the vulnerable web application by attackers. The consequences of a successful XSS attack can be session hijacking, sensitive data exposure, csrf, and impersonation of victims and can even lead to code execution on the server, depending on the application and privileges of the user's account [1].
There are three categories of XSS attacks [1].

A. Stored XSS

In Stored XSS attack, the attacker inputs malicious scripts into a web application, which is then stored in, the target server in form of a database record, or log in a webserver. A benign user as a part of the website then retrieves the malicious script. The malicious script can then access the user's session cookies, and perform actions on behalf of the user. Figure 2 illustrates the dataflow of stored XSS attacks.

B. Reflected XSS

In Reflected XSS attack, the web application immediately returns the data without making it safe for the browser. Figure 3 illustrates the dataflow of reflected XSS attacks.

C. DOM Based XSS

In the DOM-based XSS attack, the attack payload never leaves the browser and the data flow between source and sink is entirely within the browser.Figure 4 illustrates the dataflow of DOM-based XSS attacks.

III. LITERATURE REVIEW

JavaScript has become an unavoidable component when developing interactive web applications. XSS attacks pose a serious threat to web applications and as such, many works has been done to detect XSS attacks. In [5], Di Lucca el al. used a combination of static and dynamic analysis to detect XSS attacks. Static analysis results in many false positives. In order to eliminate false positives, dynamic analysis was used to verify the vulnerabilities reported by static analysis. Dynamic, static and hybrid analysis is not XSS detection but a method of detecting vulnerabilities. Noxes is a tool proposed by Kirda et al[6] which is a rule based system to detect XSS attacks.

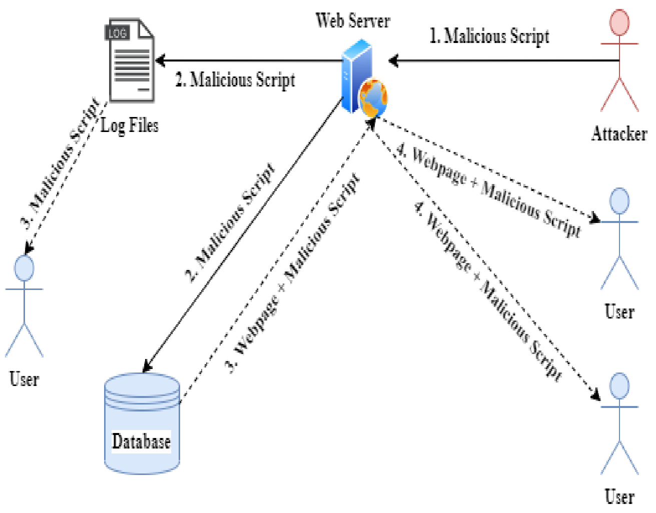

Fig. 2: Stored XSS attack dataflow

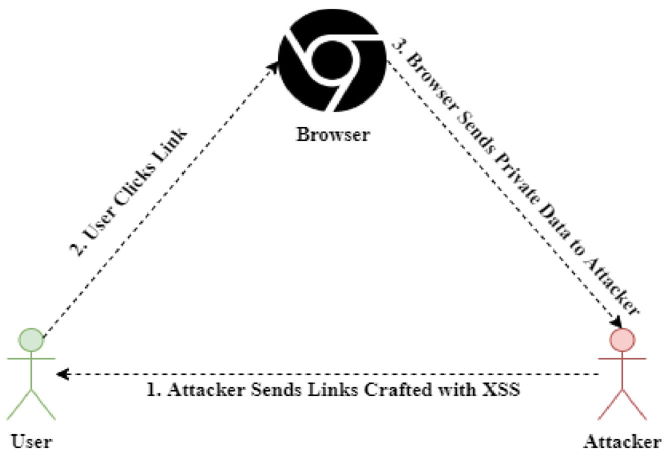
Browser
3. Browser Sends Private Data to Attacker
2. User Clicks Link
1. Attacker Sends Links Crafted with XSS
User
Attacker

Fig. 3: Reflected XSS attack dataflow

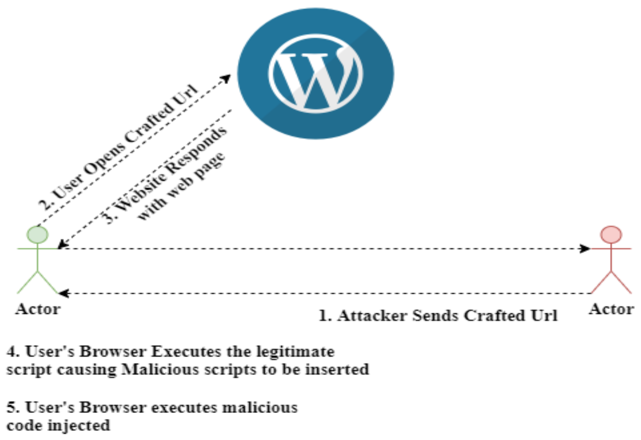
2. User Opens Crafted Url
3. Website Responds with web page
Actor
1. Attacker Sends Crafted Url
Actor
4. User's Browser Executes the legitimate script causing Malicious scripts to be inserted
5. User's Browser executes malicious code injected

Fig. 4: Dom-based XSS attack dataflow

Rule based systems has the drawback that new and zero day attacks cannot be detected. xHunter was proposed by ,Athanasopoulos et al [7]. xHunter checks JavaScript parse tree depth and if depth is greater than some threshold value, it considers it as suspicious. In [8], Wurzinger et al. developed a reverse proxy based solution SWAP(Secure Web Application Proxy). Gupta et al. [9] proposed a method called XSS-SAFE for XSS attack detection and prevention based on automated feature injection statements and placement of sanitizers in the injected code of JavaScript. The main advantage of this method is that it can detect XSS attacks without any modification to client- and server-side commodities. Chun et al. proposed XSS attack Detection Method based on SkipList[10]. In [11], Salas et al. propose an approach, which uses security-testing methods like penetration testing, and fault injection for detection of XSS attack. Vishnu B A et al. [12] proposed a method of detecting XSS attacks using machine learning algorithms, extracting the characteristics of URLs and JavaScript code and using three machine learning algorithms (naive Bayes, SVM, and J48 decision trees) to detect XSS. In their approach, they used a manual feature extraction technique and the number of features used to train the classifiers where very less. Komiya et al. [13] used machine-learning techniques to classify user input to detect malicious web code. Feature extraction depended on two methods, blank separation, and tokenizing. The dataset used was very small in their case. Likarish et al. [14] evaluated Naive Bayes, ADTree, SVM, and RIPPER classifiers in detecting obfuscation of scripts (as a proxy for malicious), using features that track the number of times symbols appears in benign and malicious scripts. Their approach also used a manual feature extraction and the number of XSS attack samples used to train was very low. Wang et al. [15] where the main idea of feature extraction is that some functions are of limited use in the benign scripts, but are used much more in malicious scripts, such as the DOM-modifying functions, the eval function, the escape function. Their approach too had a very few samples and used manual feature extraction. Nunan et al. [16] expanded the approach of Likarish et al., where features were categorized into three groups: (1) obfuscation based, (2) suspicious patterns and (3) HTML/JavaScript schemes. They also used a manual extracted feature set and number of features were very less and only works on obfuscated JavaScript.

The machine learning approaches mentioned above focuses on separating JavaScript samples, which are XSS attacks, and JavaScript samples, which are not XSS attacks. In our approach, we focus on separating JavaScript samples, which are XSS attacks from normal web application inputs, which are not XSS and can include text, numbers and other categories of inputs, which can be a combination of text, number, and punctuation.

IV. THE PROPOSED APPROACH

Figure 5 shows the architecture of the proposed approach of detecting XSS attacks using ML algorithms. The proposed system is implemented as Apache web server module. We developed a custom Apache webserver module that intercepts requests to the web application using hooks provided by Apache webserver. The intercepted requests are then analysed using the ML approach discussed in the following section. If the requests are XSS injection attacks, the request are dropped and the web application is protected from XSS attacks. Figure 6 shows ML pipeline adopted in our approach. The data collection, pre-processing, feature extraction, and training and evaluation are the most important steps in machine learning.

A. Data Collection

To train supervised machine learning algorithms we need both positive and negative samples. In our case, we need samples that are XSS attack payloads and also samples that are normal or benign web application inputs. The positive samples, which are XSS attack payloads, are collected using XSS attack tools XSSTIKE and XSSER. We also collected XSS attack payloads from various sites including GitHub and exploit-db. Figure 7 shows the data collection process for positive samples.

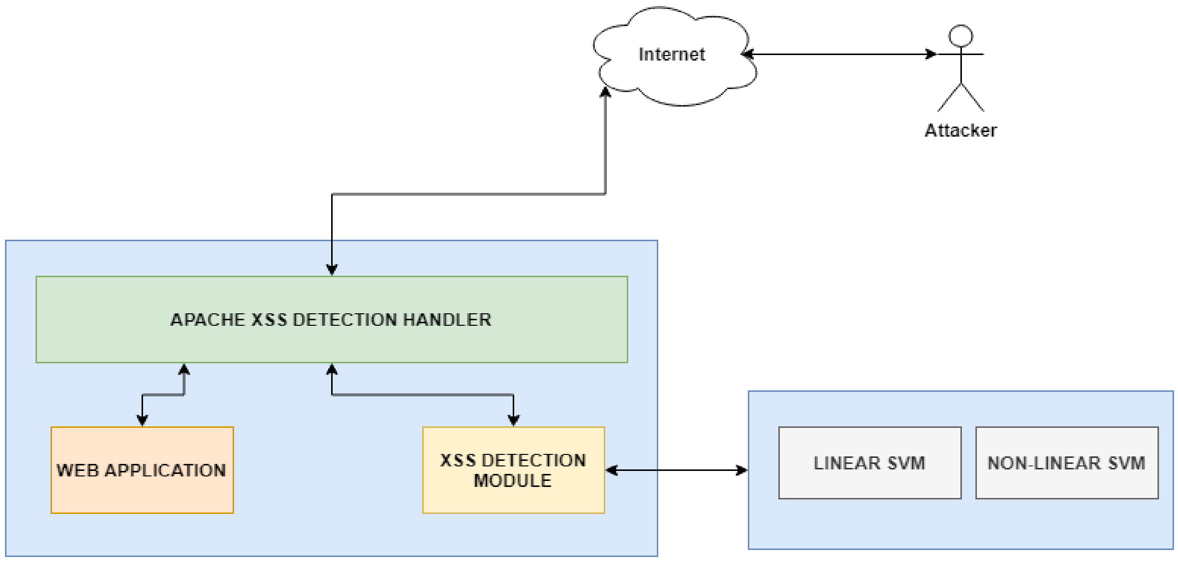

Fig. 5: Overall System

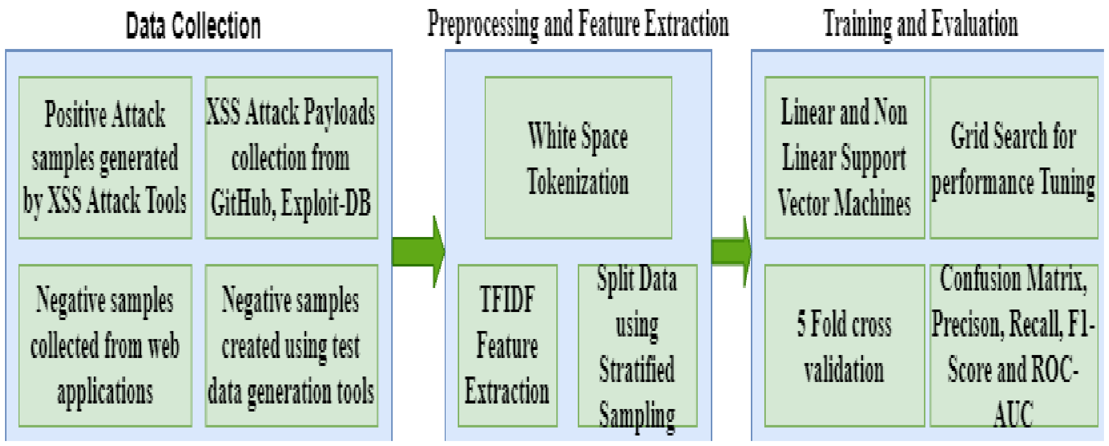
Data Collection
Positive Attack samples generated by XSS Attack Tools
XSS Attack Payloads collection from GitHub, Exploit-DB
Negative samples collected from web applications
Negative samples created using test data generation tools
Preprocessing and Feature Extraction
White Space Tokenization
TFIDF Feature Extraction
Split Data using Stratified Sampling
Training and Evaluation
Linear and Non Linear Support Vector Machines
Grid Search for performance Tuning
5 Fold cross validation
Confusion Matrix, Precison, Recall, F1-Score and ROC-AUC

Fig. 6: Machine Learning Pipeline

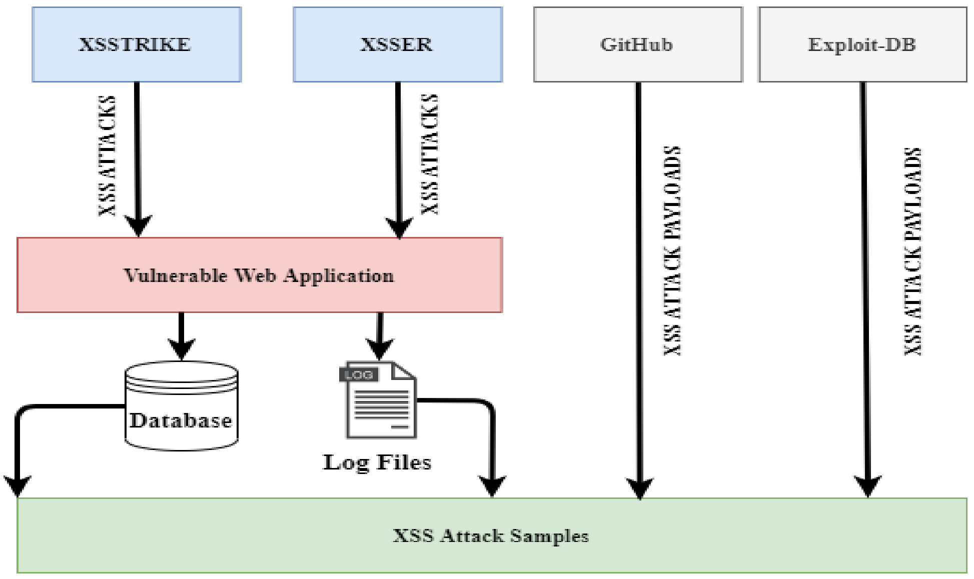

Fig. 7: Positive XSS attack sample collection process

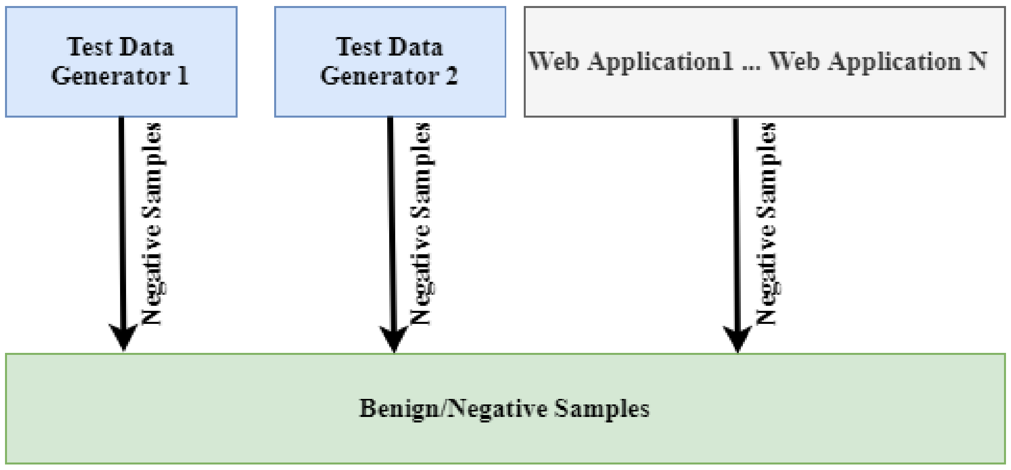

Fig. 8: Benign samples collection process

To generate negative benign samples, we use automated test data generation tools. Different types of datasets are generated using test data generation tools. Web application inputs consist mainly of text and numerical data and accordingly, we use the data generation tools to generate data sets consisting mainly of text and numerical data. Figure 8 shows the process of generating negative samples. Our dataset consists of 15000 XSS attack samples and 15000 benign samples.

B. Pre-processing and Feature Extraction

Figure 9 shows our data preparation process. The raw data collected is in the form of text data. The raw data is split into a training set and a test set in a ratio of 70 to 30. The training data is used to train the ML model and the test data set is used to test the performance of the model. The training and test data set are then transformed into a form that is suitable as input to machine learning algorithms. A whitespace tokenizer then breaks the text into terms whenever it encounters whitespace. TFIDF (Term Frequency Inverse Document Frequency) is used for feature extraction from the tokenized text. TFIDF is a statistical technique that is used to evaluate the importance of a word in a document in a collection or corpus [14].

TFIDF measure is composed of two statistics, term frequency, and inverse document frequency. The formula for calculating TFIDF given a word t, a document d, and document set D is as follows

$$tfidf(t,d,D) = tf(t,d) * idf(t,D)$$

Where:

$$tf(t,d) = \log(1 + freq(t,d))$$
$$idf(t,D) = \log(N/count\ (d \in D{:}\, t \in d))$$

The output of TFIDF feature extraction is a sparse matrix, which indicates the TFIDF score for all non-zero values in the word vector for each document, which in our case is a single positive attack sample or a negative attack sample. The sparse matrix becomes the input to our machine learning algorithms.

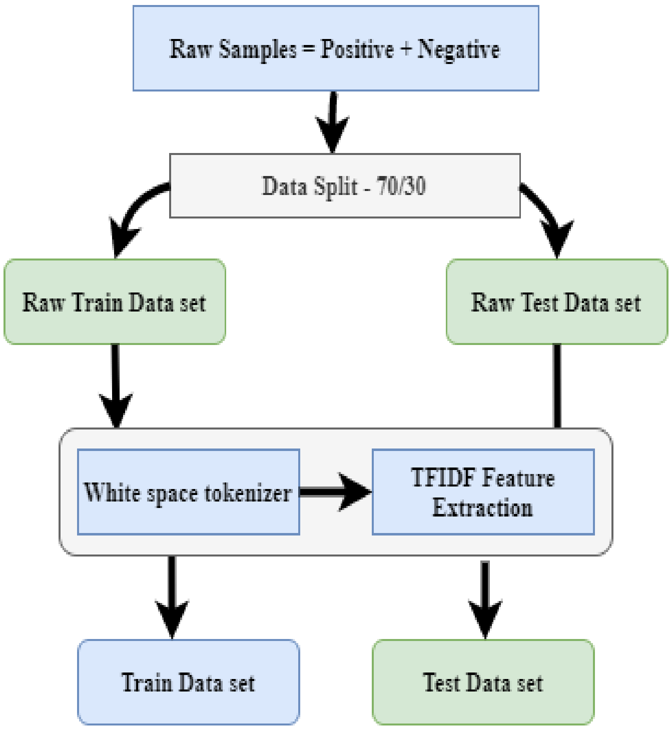

Fig. 9: Data preparation process

C. Training And Evaluation

Our task is to separate the positive attack samples from negative attack samples, which is a binary classification task. The most commonly used classification algorithms are Logistic Regression, Naïve Bayes, Support Vector Machines (SVM), Random Forest, Neural Networks, Bagging and Boosting, Deep Learning. In our approach, we used the SVM algorithm. SVM has the advantage of being robust against overfitting problems, especially for text data due to high dimensional space [15].

1) Algorithms

In our approach, we have used the SVM algorithm for classifying web application inputs as XSS attacks or benign inputs. Vapnik and Chervonenkis [16] developed the linear version of Support vector machines. Later B.E. Boser et al [17] developed the nonlinear version of SVM. SVM can be used for binary and multiclass classification but here we only consider the binary class SVM since our task is a binary classification task.

Linear Binary class SVM

Let D be a dataset

$D = \{(x_i, y_i) | x_i \in R^n, y_i \in \{-1,1\}\}_{i=1}^{m}$.

SVM tries to find $w \in R^n$ and $b \in R$ such that the prediction is given by $sign(wx + b)$ is correct for most samples. SVM tries to find w and b by solving the following optimization problem:

$$\min_{w,b} \frac{1}{2} \parallel w \parallel^2$$
$$subject\ to\ y_i(wx + b) - 1 \geq 0, i = 1, \dots, m$$

The above optimization problem is a convex quadratic optimization problem.

Non Linear Binary class SVM

The linear binary class SVM can only work for linearly separable data but most of the real-world datasets are not linearly separable. To overcome this limitation of Linear Binary class SVM, Non Linear Binary Class SVM was developed. The problem of hard margin classifier is solved by adding slack variables ζ_i and regularization parameter C. The regularized optimization problem then becomes

$$\min_{w,b,\zeta} \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{m} \zeta_i$$
$$subject\ to\ y_i(wx_i + b)\ \geq 1 - \ \zeta_i, \zeta_i \geq 0, i = 1, \dots, m$$

2) Training

Figure 10 shows our training approach. We trained two classifiers. The first classifier was trained using linear SVM while the second was trained using non-linear SVM. The accuracy and performance of the classifiers depend on the hyperparameters of the algorithm. Hyperparameters control the learning process. Hyperparameters cannot be derived via training but must be specified by the user of the ML algorithm. The process of finding a set of optimal hyperparameters for a given ML algorithm is known as Hyperparameter Optimization. Grid search, Random search, Bayesian Optimization, Gradient-based optimization are some approaches to Hyperparameter optimization. In our approach, we used Grid search for Hyperparameter optimization.

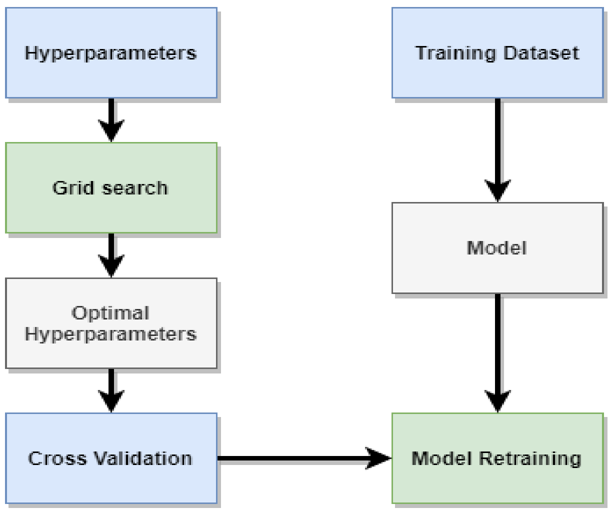

Fig.10: Training Process

Machine learning models often suffer from the problem of overfitting. Overfitting occurs when the ML model too closely models the details and noise in the data and fails to generalize the relationship between dependent and independent variables.

This failure to generalize leads to the poor ability of the model to predict new data. Non-linear ML algorithms are more susceptible to overfitting due to the flexibility in learning the target function. To prevent overfitting in the SVM algorithm, we used k-Fold cross-validation. k-Fold cross-validation is a statistical resampling technique where the ML model is trained and tested on k subsets of training data. In our approach, we used a cross-validation approach with a value of k as 10. The Learning curve for linear SVM is shown in figure 11. and the learning curve for non-linear SVM is shown in figure 12. The learning curve shows that the MSE (Mean Squared Error) against the number of training samples. From the curve, it is seen that the MSE of non-linear SVM is lower than that of linear SVM. Figure 13. shows the training and validation score for linear SVM and Figure 14 shows the training and validation score for non-linear SVM. The train and validation curve for linear SVM and non-linenar SVM shows that the model is able to generalize well and is not overfitting. The score of the models can be further improved with more samples.

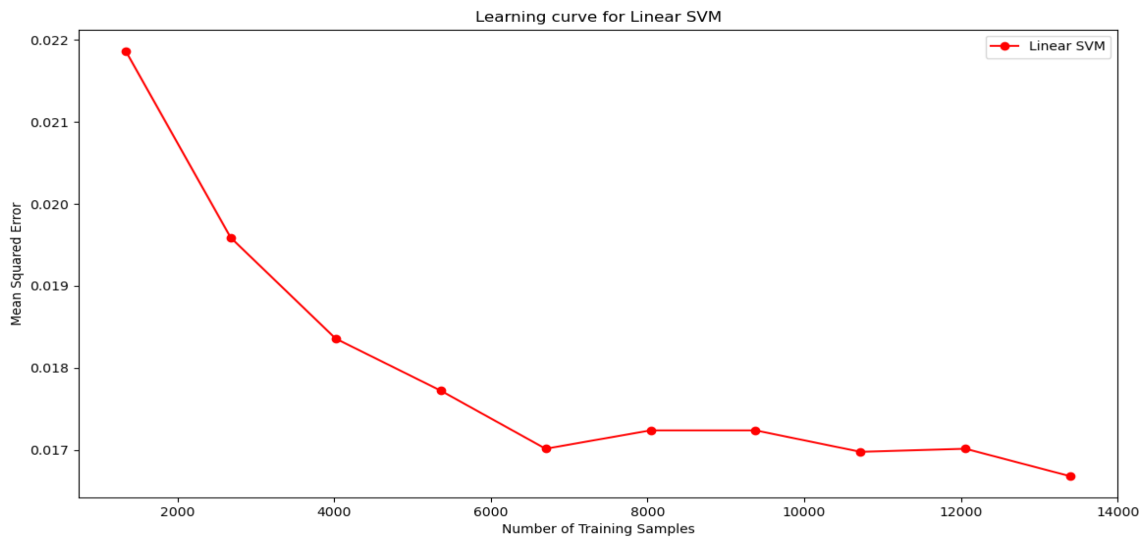

Fig. 11: Learning curve linear SVM

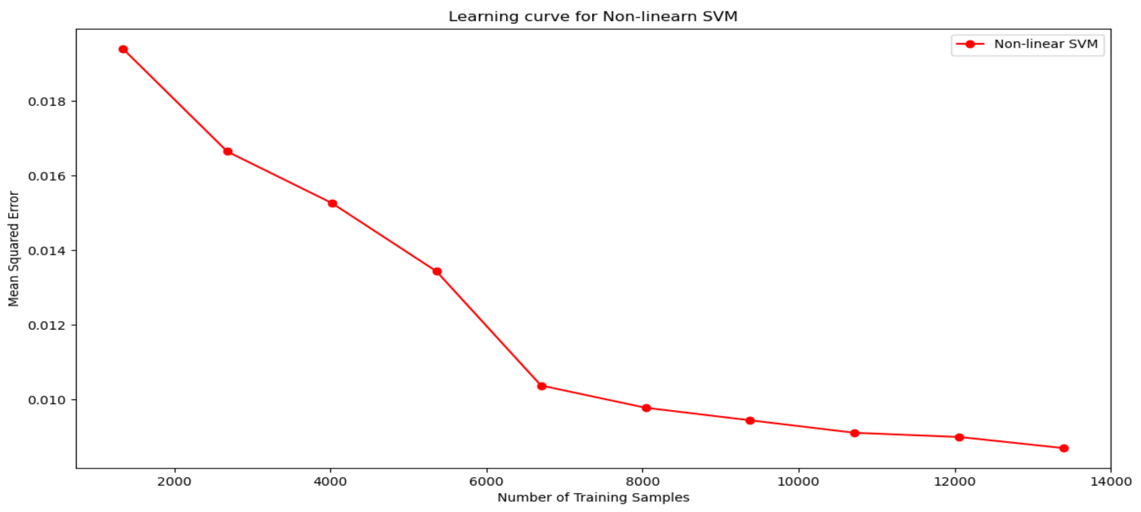

Fig.12: Learning curve non-linear SVM

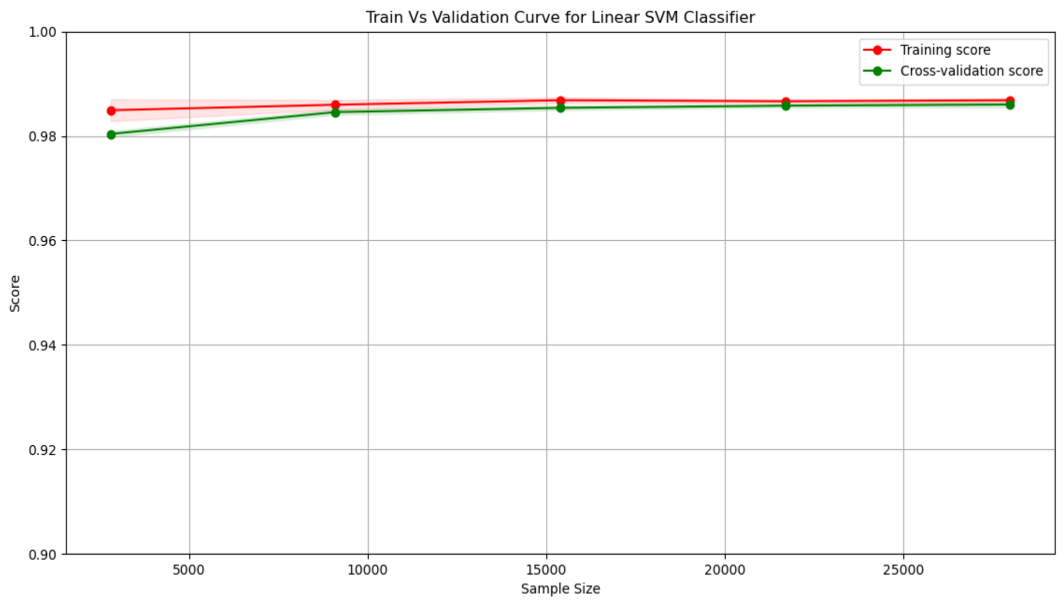

Fig. 13: Training and validation curve linear SVM

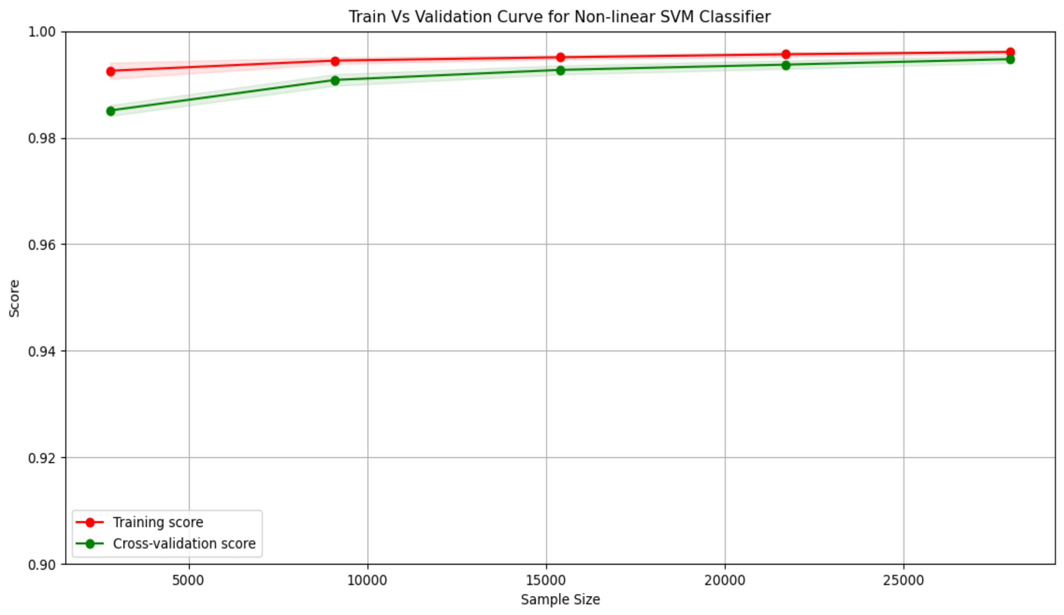

Fig. 14: Training and validation curve non-linear SVM

3) *Evaluation*

The accuracy of an ML model is measured based on TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative). TP is the number of correctly classified positive samples; TN negative is the number of correctly classified negative samples; FP is the number of samples that are incorrectly classified as positive and FN is the number of samples that are incorrectly classified as negative. Concerning our problem, TP is the number of samples correctly classified as XSS attacks, and TN is the number of samples correctly classified as benign inputs.

The following performance metrics were used to measure the performance of our machine learning algorithms.

a) *Precision*

Precision is the ratio of correctly predicted positive observations to the total predicted observations. The formula for calculating Precision is

$$Precision = \frac{True\ Positives}{True\ Positive + False\ Positive}$$

b) *Recall*

The recall is the ratio of correctly predicted positive observations to all observations in an actual class. The formula for calculating Recall is

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

c) F1 Score

F1 score - F1 score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. The formula for calculating F1 score is

$$F1\ Score = \frac{2 * Recall * Precision}{Recall + Precision}$$

We evaluated both the linear SVM model and non-linear SVM model against the test data that was set aside for evaluation during the data preparation and pre-processing phase.
Figure 15. show the confusion matrix for linear SVM and Figure 16. show the confusion matrix for non-linear SVM.

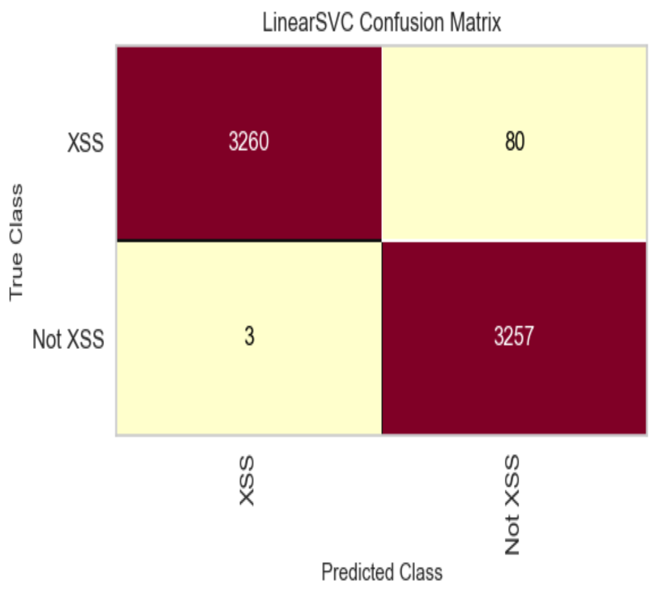

Fig. 15: Confusion matrix linear SVM

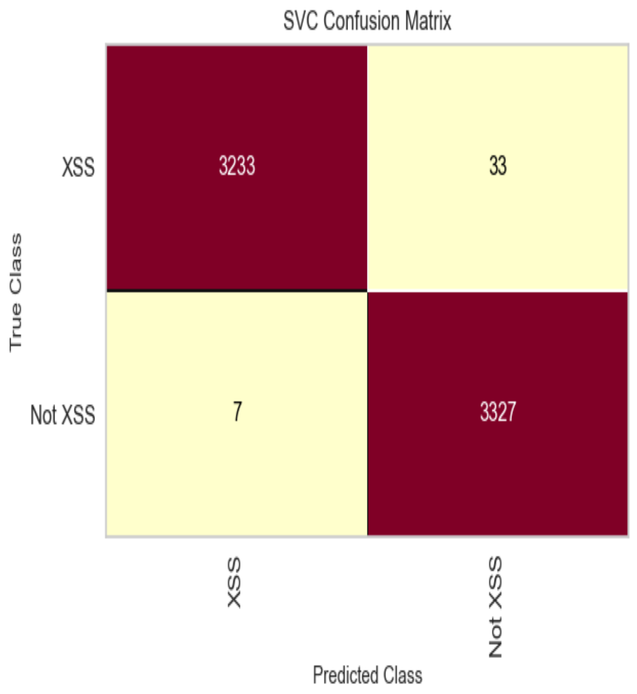

Fig. 16: Confusion matrix non-linear SVM

Figure 13 and 14 shows the graph of training loss and validation loss of linear and non-linear SVM respectively. The validation loss is lesser than the training loss indicating that the model is not overfitting. The precision, recall and F1 score for both linear and non-linear SVM are shown in Table.

Table 1: F1 score for both linear and non-linear SVM

Metric	Value
Precision	0.97
Recall	0.99
F1 Score	0.97

Table 2: F1 score for both linear and non-linear SVM

Metric	Value
Precision	0.98
Recall	0.99
F1 Score	0.98

The precision, recall and f1-score of both linear and non-linear SVM indicates that the algorithms are able to successfully separate the XSS attack inputs from benign web application inputs. The precision of both linear SVM and non-linear SVM shows that false positive rate is less. The recall show that false negative rate is less. The f1-score shows that there is a balance between precision and accuracy, which is desirable in our case as it, shows that both false positive rate and false negative rates are low. Figure 17 and 18. shows the scalability of linear SVM and non-linear SVM respectively. It plots the model fitting times in seconds against the number of training samples. Though the non-linear SVM performs better in terms of accuracy and f1-score, it takes much longer time than the linear SVM for model fitting.

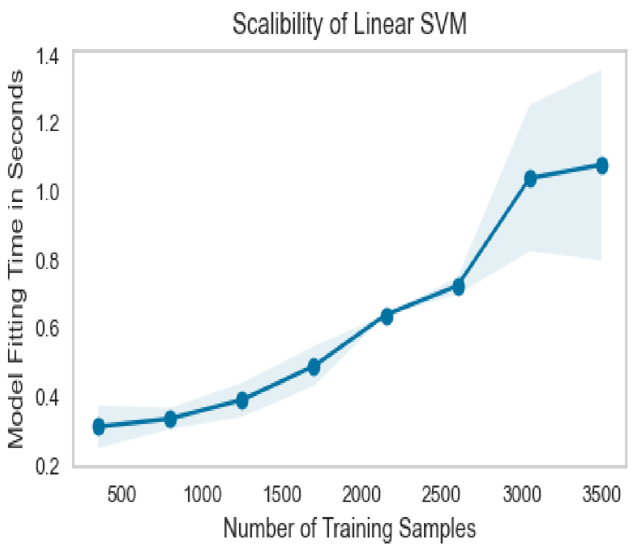

Fig. 17: Scalability linear SVM

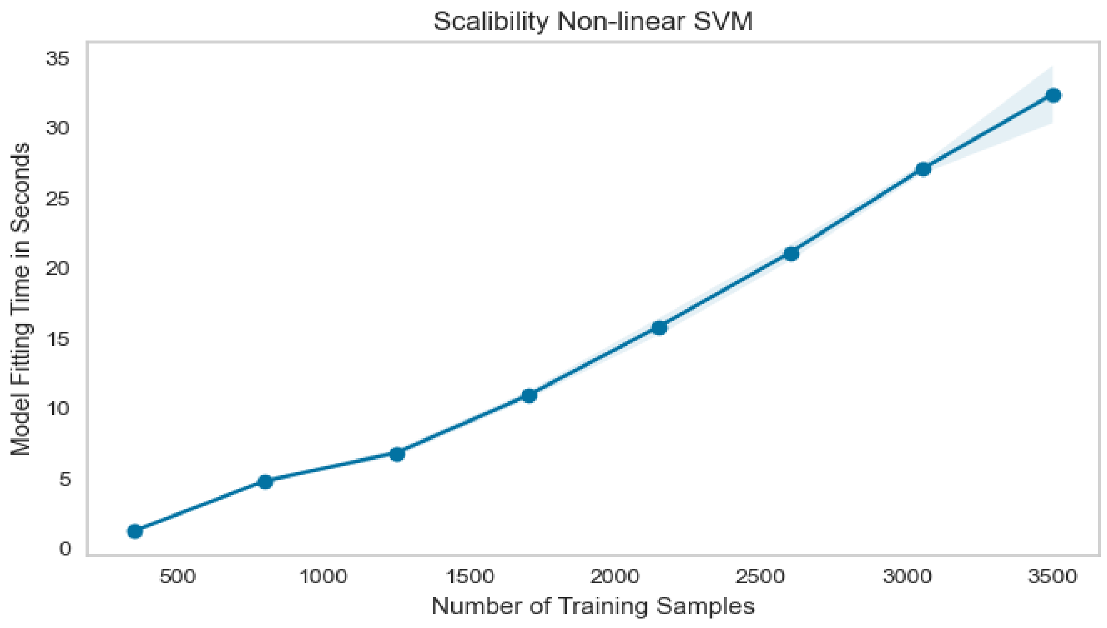

Fig. 18: Scalability non-linear SVM

V. CONCLUSION

From the experimental results, we can conclude that ML approaches have advantages over traditional approaches of detecting XSS attacks. ML approach combined with traditional approaches can detect XSS attacks with a higher accuracy. In this paper, we combined the traditional method of using WAF with ML to detect XSS attacks in web applications. The future scope of this approach is that ML approaches can be combined with other traditional approach like static analysis, dynamic and hybrid analysis to detect and prevent XSS attacks in web applications.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

[1] Cross Site Scripting Exploits and Defense, Jeramiah Grossman, Rober Hansen, Petko D. Petkov, Anton Rager, Seth Fogie, Syngress, pp. 2-11

[2] H. Huang, Z. Zhang, H. Cheng and S. W. Shieh, "Web Application Security: Threats, Countermeasures, and Pitfalls," in Computer, vol. 50, no. 6, pp. 81-85, 2017, doi: 10.1109/MC.2017.183.

[3] Anderson, P. (2008). The Use and Limitations of Static-Analysis Tools to Improve Software Quality. CrossTalk-Journal of Defense Software Engineering. 21.

[4] Rami Sihwail, Khairuddin Omar, K. A. Z. Ariffin, A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid

[5] G. A. Di Lucca, A. R. Fasolino, M. Mastoianni, and P. Tramontana, “Identifying cross site scripting vulnerabilities in web applications,” in 26th Annual International Telecommunications Energy Conference, pp. 71–80, 2004.

[6] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic, “Noxes: A client-side solution for mitigating crosssite scripting attacks,” in Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 330– 337, New York, NY, USA, 2006.

[7] E. Athanasopoulos, A. Krithinakis, and E. P. Markatos, “Hunting cross-site scripting attacks in the network,” in Third International Conference on Advanced Computing (ICoAC’11), pp. 89–92, 2011.

[8] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel, “SWAP: Mitigating XSS attacks using a reverse proxy,” in Proceeding of 5th International Workshop on Software Engineering for Secure Systems, IEEE Computer Society, 2009.

[9] B. B. Gupta, S. Gupta, S. Gangwar, M. Kumar, and P. K. Meena, “Cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense,” Journal of Information Privacy and Security, vol. 11, no. 2, pp. 118–136, 2015.

[10] S. Chun, C. Jing, H. ChangZhen, X. JingFeng, W. Hao, and M. Raphael, “A xss attack detection method based on skip list,” International Journal of Security and Its Applications, vol. 10, no. 5, pp. 95– 106, 2008.

[11] M. I. P. Salas and E. Martins, “Security testing methodology for vulnerabilities detection of XSS in web services and ws-security,” Electron Notes in Theoritical Computer Science, vol. 302, pp. 133–154, 2014.

[12] Vishnu, B.A.; Jevitha, K.P. Prediction of cross-site scripting attack using machine learning algorithms. In Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing, Amritapuri, India, 10–11 October 2014; p. 55.

[13] Komiya, R., Paik, I., Hisada, M.: Classification of malicious web code by machine learning. In: Awareness Science & Technology (iCAST), pp. 406–411. IEEE (2011)

[14] Likarish, P., Jung, E., Jo, I.: Obfuscated malicious JavaScript detection using classification techniques. In: Malicious and Unwanted Software (MALWARE), pp. 47–54. IEEE (2009)

[15] Wang, W.H., Yin-Jun, L.V., Chen, H.B., Fang, Z.L.: A static malicious javascript detection using SVM. In: International Conference on Computer Science and Electronics Engineering, vol. 40, pp. 21–30. Atlantis Press (2013)

[16] Nunan, A.E., Souto, E., dos Santos, E.M., Feitosa, E.: Automatic classification of cross-site scripting in web pages using document-based and url-based features. In: Computers and Communications, pp. 702–707. IEEE (2012)

[17] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. ACM Press, New York, 1999

[18] Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, and Brown, “Text Classification Algorithms: A Survey,” Information, vol. 10, no. 4, p. 150, Apr. 2019.

[19] Vapnik, V.; Chervonenkis, A.Y. A class of algorithms for pattern recognition learning. Avtomat. Telemekh 1964, 25, 937–945

[20] Boser, B.E.; Guyon, I.M.; Vapnik, V.N. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992; pp. 144–152

ABOUT THE AUTHORS

Mr. Bronjon Gogoi is a Scientist at Regional Centre of Excellence for Application Security, National Informatics Centre, Guwahati Assam India. His main area of works include application security, communication and network, AI and machine learning.

Mr. Tasirudding Ahmed is a Scientist at Regional Centre of Excellence for Application Security, National Informatics Centre, Guwahati Assam India. His main area of works include application security, communication and network, AI and machine learning.

Mr. Hemanta Kumar Saikia is a Scientist at Regional Centre of Excellence for Application Security, National Informatics Centre, Guwahati Assam India. His main area of works include application security, communication and network, AI and machine learning.