THE OVERVIEW OF XSS ATTACK DETECTION METHODS ON WEB APPLICATIONS

Divya Sri Bevara
Computer Science & Engineering,
University of North Texas, Denton,
DivyaSriBevara@my.unt.edu

Karishma Bollineni
Computer Science & Engineering,
University of North Texas, Denton,
KarishmaBollineni@my.unt.edu

Parthiv Kumar Gunjari
Computer Science & Engineering,
University of North Texas, Denton,
ParthivKumarGunjari@my.unt.edu

Santhosh Yadlapally
Computer Science & Engineering,
University of North Texas, Denton
SanthoshYadlapally@my.unt.edu

Abstract: Web application is widely used in these days, which is also a good opportunity for the attackers to target for security beaches. Cross-site scripting (XSS) is a important security threat where harmful code is injected into web pages, which can manipulate or steal sensitive data, causing serious attacks. In this paper we explore different XSS vulnerabilities and discuss the most effective methods for detecting and preventing these attacks, including their impact. We propose using machine learning techniques for detecting XSS attacks because of their powerful ability to recognize anomalies and patterns. Specifically, we use various machine learning algorithms such as Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), and Naïve Bayes (NB) to improve security measures.

Keywords: XSS, web application, security, cross site scripting, vulnerability, ML.

I. INTRODUCTION

In today's digital world, most of the business and organization use web applications to increase business with market competition and technological advancements. This has led to a rise in cross-site scripting (XSS) attacks, levelling major risks to user privacy an data security. These attacks occur when harmful code is injected into a web application through its vulnerable parts, allowing attackers to steal sensitive user data and gain unauthorized access to personal information. This highlights the urgent need for robust security measures to understand and combat various types of XSS attack, including persistent, non-persistent, DOM-based, and including XSS attacks.

As technology is evolving, attackers are finding new method to steal data, despite ongoing efforts to stop them using detection tools like WebKnight, XSS Hunter, Acunetix, ModSecurity with OWASP CRS, and other countermeasures.

In order to identify and prevent XSS attacks, it is mainly divided into 4 types of approaches that is, static analysis, hybrid analysis, and machine learning. So, as mentioned earlier we choose the machine learning method to detect and prevent XSS attacks.

So, we opted for a machine learning approach because it is flexible like scalability, adaptability, real-time detection, and accuracy and useful to identify unusual activity or new vulnerabilities, when compared to other methods.

II. BACKGROUND

A thorough review of the existing literature which explains the historical development, current practices, and new trends in addressing XSS (Cross-Site Scripting) attacks. In the 1990s, researchers like Rsnake began to explore how to improve input validation, understand XSS attack methods, and enhance the output encoding. Over time, researchers have developed various techniques for detecting and mitigating XSS vulnerabilities, including static, dynamic, and hybrid methods.

Over the years, researchers have pursued different strategies to identify and correct XSS vulnerabilities. Static analysis, for example, aims to spot problematic code patterns during the development stage by examining the code and the abstract syntax tree (AST). Dynamic analysis involves ongoing monitoring and tracking of code during execution. Hybrid analysis combines elements of both static and dynamic methods to probe XSS vulnerabilities. In these days , artificial intelligence (AI) and machine learning (ML) are also being applied to improve detection capabilities.

III. METHODOLOGY

This task primarily consists of three steps that is selecting datasets, preprocessing the data, and then classifying and predicting the data.

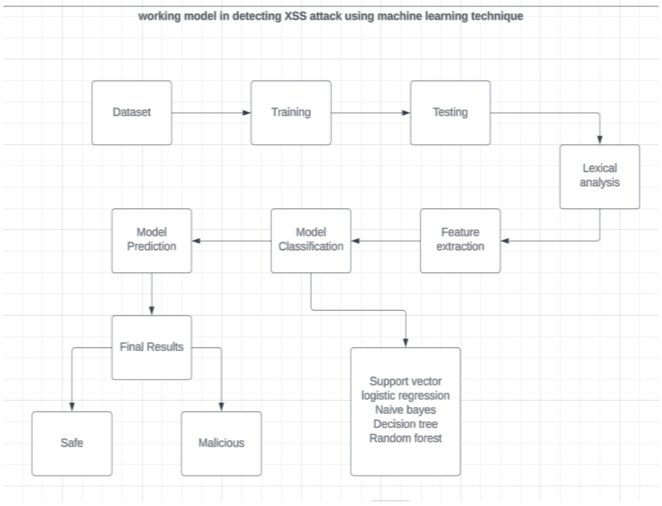
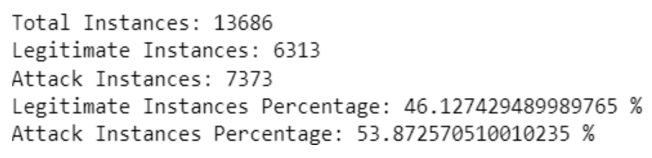


Fig. 1. Working model of detecting XSS attack using ML.

A. Datasets

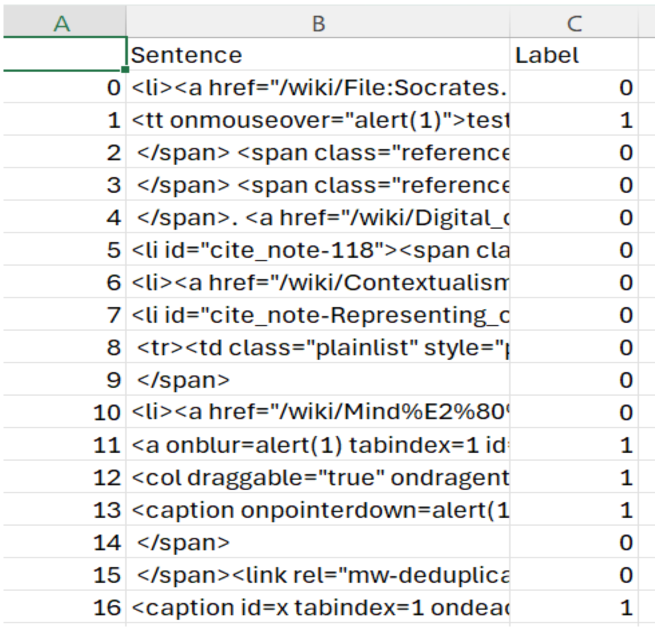
For our research, we used a dataset from Kaggle that deals with Cross-Site Scripting. It includes 13,686 entries, of which 6,313, or about 46%, are legitimate, and 7,373, or about 54%, are examples of attack instances.



```
Total Instances: 13686
Legitimate Instances: 6313
Attack Instances: 7373
Legitimate Instances Percentage: 46.127429489989765 %
Attack Instances Percentage: 53.872570510010235 %
```

Fig. 2. Total instances and types in the selected dataset.

The dataset is divided into XSS scripts, we divided into various columns. Each script is labelled as a Sentence, and it’s marked as (0 or 1) based on a detailed analysis. We have split the data set into two parts, in that 80% is used for the training set, which helps in feature extraction and model training, while the remaining 20% is the testing set, used to evaluate the model’s performance on new data that hasn’t been tested before.

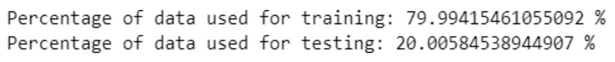


A	B	C
	Sentence	Label
0	<li><a href="/wiki/File:Socrates.	0
1	<tt onmouseover="alert(1)">test	1
2	</span> <span class="reference	0
3	</span> <span class="reference	0
4	</span>. <a href="/wiki/Digital_(	0
5	<li id="cite_note-118"><span cla	0
6	<li><a href="/wiki/Contextualism	0
7	<li id="cite_note-Representing_c	0
8	<tr><td class="plainlist" style="|	0
9	</span>	0
10	<li><a href="/wiki/Mind%E2%80	0
11	<a onblur=alert(1) tabindex=1 id	1
12	<col draggable="true" ondragent	1
13	<caption onpointerdown=alert(1	1
14	</span>	0
15	</span><link rel="mw-deduplica	0
16	<caption id=x tabindex=1 ondeac	1

Fig. 3. Image of data of the selected dataset.

B. Preprocessing of data

The dataset consists of different queries, each containing a unique set of parameters. These queries are first written in Unicode and later converted into a readable format for additional processing. During this process, regular expression is used to identify the parameters in each query. Queries that have significantly high number of parameters are removed from the dataset to ensure the quality of the data.



```
Percentage of data used for training: 79.99415461055092 %
Percentage of data used for testing: 20.00584538944907 %
```

Fig. 4. Percentage of data used for training and testing.

The Fig. 5 shows how we prepared our data. First, we split the raw text data into two parts: 80% for training and 20% for testing.

Next, we cleaned the training data by removing the errors and irrelevant information. We then formatted both the training data and the testing datasets to ensure the consistency. Finally, we used this prepare data to train our model to accurately predict the outcomes.

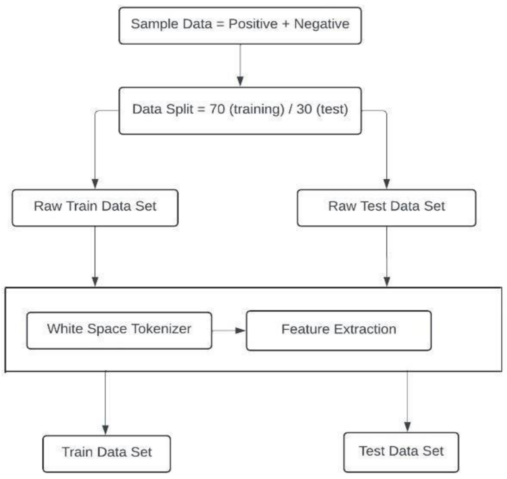
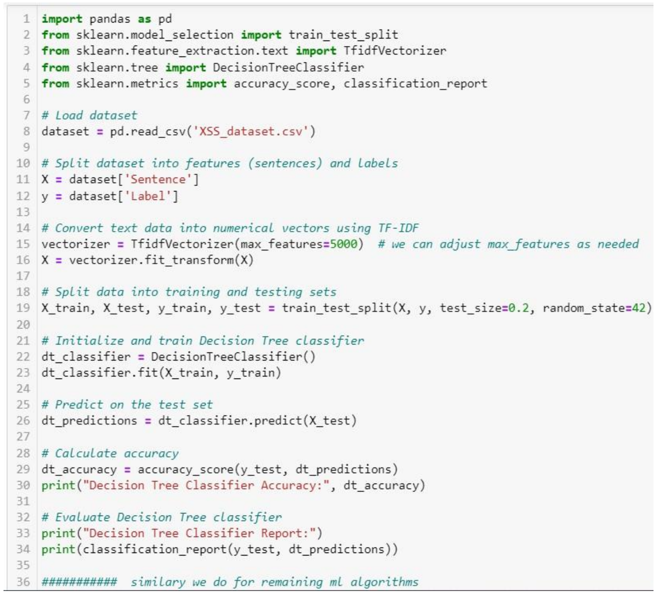


Fig. 5. The diagram on preprocessing of data

Here, we use the training subset to build the machine learning model and we test the subset to check it efficiency. The two subsets are transformed to fit the format that is used for the machine learning algorithm. After that, a whitespace tokenizer is used, that breaks down the text into separate words at each space, making it easier to extract features from the tokenizer text for in depth analysis.

C. Building a learning model

In this step we developed our machine learning model which uses four algorithms that is logistic regression, naïve bayes, support vector machine and decision tree.



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
dataset = pd.read_csv('XSS_dataset.csv')

# Split dataset into features (sentences) and labels
X = dataset['Sentence']
y = dataset['Label']

# Convert text data into numerical vectors using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # we can adjust max_features as needed
X = vectorizer.fit_transform(X)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Decision Tree classifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)

# Predict on the test set
dt_predictions = dt_classifier.predict(X_test)

# Calculate accuracy
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Classifier Accuracy:", dt_accuracy)

# Evaluate Decision Tree classifier
print("Decision Tree Classifier Report:")
print(classification_report(y_test, dt_predictions))

###########  similary we do for remaining ml algorithms
```

Fig. 6. Code used to measure parameters.

- Support Vector Machines (SVM)
  Support Vector Machine (SVM) is the best machine learning algorithm which is used for both classifying and predicting data. SVM works by creating a hyperplane line which separates different classes of data as widely as possible within a designated boundary. Let us consider an example, in binary classification we divide data into two groups where SVM constructs a hyper plane that maximizes the distance between these two groups within a given margin.

  The formula for binary classification using linear SVM is f(x) = sign (w * x + b)

- Logistic Regression (LR)
  Logistic Regression (LR) is another type of machine learning algorithm used for classification. It estimates the probability that the input relates to a specific class. Firstly, we start by setting initial values by calculating the log odds of the input being in one class using a linear equation. It then uses a logistic function which is also known as sigmoid function, which is used to transform these log odds into probabilities. During the training of the model, it adjusts its parameters to minimize the difference between the predicted probabilities and the actual outcomes. Once the model is trained then the model makes predictions by applying a threshold to these probabilities to make evaluations.

  The formulas are given as , Log odds = w * x + b and
  $P(y) = \frac{1}{1+e^{-(w * x + b)}}$

- Decision Tree (DT)
  Decision tree is a supervised ML algorithm that look like a tree structure which can be used for both classification and regression tasks. Decision Tree have nodes and branches, where the nodes used for representing the questions and the branches are used for representing the outcomes. The process starts at the root node which has the initial data and continuously splits the data into smaller

subsets at each internal node. The process of splitting the data is repeated until it reaches the leaf node, where the model makes the predictions by monitoring the growth process from the root node till the leaf node.

- Naïve Bayes (NB)
  NB uses the bayes theorem which calculates the probability of a class given input features. It chooses the class with highest probability for prediction and during training period it calculates the probabilities from the data by assuming that the features are independent when condition is on the class label.

  The formula used in NB is as follows:

$$P\,(y \mid x_1, x_2, \ldots, x_n) = \frac{P(y) * \prod_{i=1} P(x_i \mid y)}{P(x_1, x_2, \ldots, x_n)}$$

In using these algorithms, we consider some design metrices such as precision, recall, accuracy, and F-score to compare and calculate its effectiveness. The formulas is as follows:

1. Precision: $\frac{TP}{TP + FP}$
2. Accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$
3. Recall: $\frac{TP}{TP + FN}$
4. F-score: $(1 + \beta^2) * \frac{Precision \, * \, Recall}{(\beta^2 * Precision) * Recall}$

Where:
True Positive (TP) is used for identification of an event that occurred.

True Negative (TN) is used for identification of an even that has not occurred.

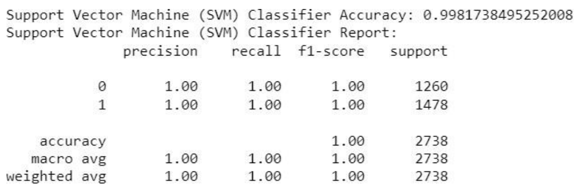
False Positive (FP) is used for identification of incorrect identification of an event as occurring.

False negative (FN) is used as an failure to identify an event that has not occurred.

IV. RESULTS

We successful design and implementation of the machine learning algorithms such as support vector machines (SVM), logistic regression (LR), decision tree (DT), naïve bayes (NB) on evaluating these with parameters such as accuracy, precision, recall, f-score resulted in showcasing outstanding performance and outputs as shown below.

The support vector machine performed the best with nearly perfect accuracy and precision overcoming the other three.

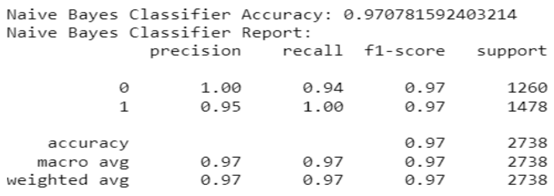


```
Support Vector Machine (SVM) Classifier Accuracy: 0.9981738495252008
Support Vector Machine (SVM) Classifier Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1260
           1       1.00      1.00      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 7. Measure of accuracy using support vector machine.

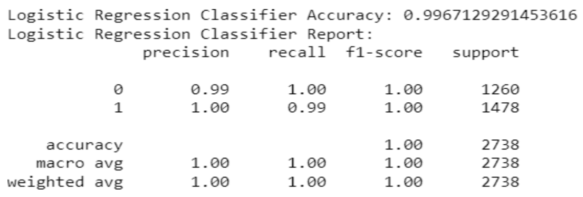


```
Naive Bayes Classifier Accuracy: 0.970781592403214
Naive Bayes Classifier Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      1260
           1       0.95      1.00      0.97      1478

    accuracy                           0.97      2738
   macro avg       0.97      0.97      0.97      2738
weighted avg       0.97      0.97      0.97      2738
```

Fig. 8. Measure of accuracy using naïve bayes.

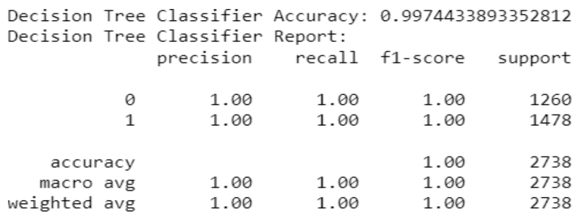


```
Logistic Regression Classifier Accuracy: 0.9967129291453616
Logistic Regression Classifier Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1260
           1       1.00      0.99      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 9. Measure of accuracy using logistic regression.

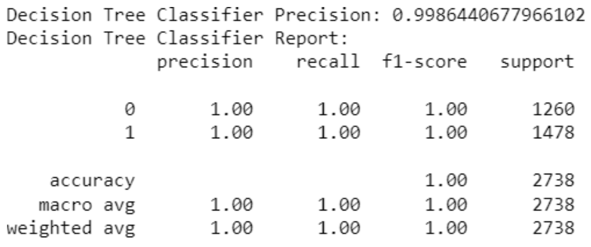


```
Decision Tree Classifier Accuracy: 0.9974433893352812
Decision Tree Classifier Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1260
           1       1.00      1.00      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 10. Measure of accuracy using decision tree.

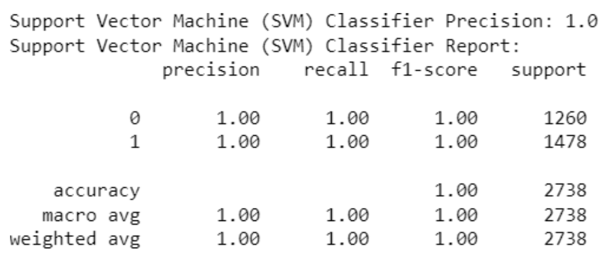


```
Decision Tree Classifier Precision: 0.9986440677966102
Decision Tree Classifier Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1260
           1       1.00      1.00      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 11. Measure of precision using decision tree.

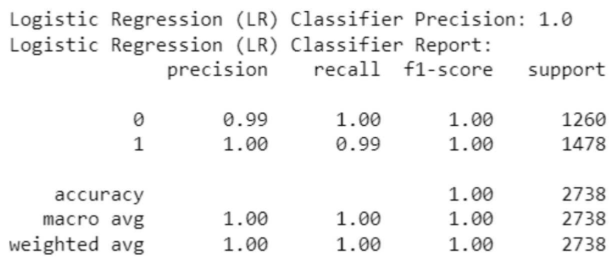


```
Support Vector Machine (SVM) Classifier Precision: 1.0
Support Vector Machine (SVM) Classifier Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1260
           1       1.00      1.00      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 12. Measure of precision using support vector machine.

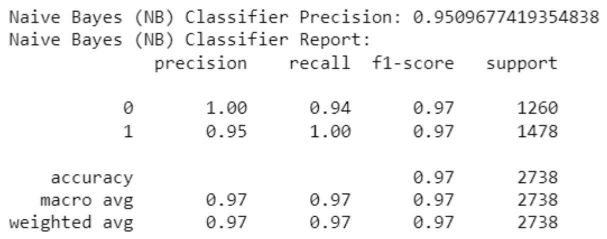


```
Logistic Regression (LR) Classifier Precision: 1.0
Logistic Regression (LR) Classifier Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      1260
           1       1.00      0.99      1.00      1478

    accuracy                           1.00      2738
   macro avg       1.00      1.00      1.00      2738
weighted avg       1.00      1.00      1.00      2738
```

Fig. 13. Measure of precision using logistic regression.



```
Naive Bayes (NB) Classifier Precision: 0.9509677419354838
Naive Bayes (NB) Classifier Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      1260
           1       0.95      1.00      0.97      1478

    accuracy                           0.97      2738
   macro avg       0.97      0.97      0.97      2738
weighted avg       0.97      0.97      0.97      2738
```

Fig. 14. Measure of precision using naïve bayes.

V. DISCUSSION

Based on the results, we came to know that the support vector machine is performed very well with an accuracy of 99.81% and 100% precision compared to other machine learning algorithms such as logistic regression followed by decision tree and naïve bayes showed least performance in terms of parameters like accuracy (97.07%) and precision (95.09%).

Table 1. Comparison of ML algorithms vs parameters.

Algorithms vs Parameter	Accuracy	Precision
Support Vector Machines	0.9981	1.0
Logistic Regression	0.9967	1.0
Naïve Bayes	0.9707	0.9509
Decision Tree	0.9974	0.9986

VI. CONCLUSION

We mainly focused on evaluating various XSS detection methodologies using supervised machine learning models like support vector machines, logistic regression, naïve bayes, decision tree to understand their efficiency, boundaries, and reconnoitring various levels of web application to make them better and secure by detecting several XSS vulnerabilities in advancement and deployment process and obtained outstanding results.

VII. REFERENCES

1. Vikas K. Malviya, Saket Saurav, Atul Gupta, "On Security Issues in Web Applications through Cross Site Scripting (XSS)", *20th Asia-Pacific Software Engineering Conference*, 2013.
2. Hsing-Chung Chen, Aristophane Nshimiyimana, Cahya Damarjati, Pi-Hsien Chang, "Detection and Prevention of Cross-site Scripting Attack with Combined Approaches", *International Conference on Electronics, Information, and Communication (ICEIC)*, 2021.
3. Miao Liu, Boyu Zhang, Wenbin Chen, Xunlai Zhang, "A Survey of Exploitation and Detection Methods of XSS Vulnerabilities", *IEEE Access (Vol: 7)*, 2019.
4. Mohit Dayal Ambedkar, Nanhay Singh Ambedkar, Ram Shringar Raw, "A Comprehensive Inspection of Cross Site Scripting Attack", *International Conference on Computing, Communication and Automation (ICCCA)*, 2016.
5. Stanislav Kascheev; Tatyana Olenchikova, "The Detecting Cross-Site Scripting (XSS) Using Machine Learning Methods", 2020 *Global Smart Industry Conference (GloSIC)*.
6. Gupta, B. B., C Badve, O. P. (2016). Taxonomy of SQL injection and XSS attacks. Procedia Computer Science, 78, 471-478.
7. Shar, L. K., C Tan, H. B. K. (2013). Defending against cross-site scripting attacks. Computer, 45(3), 55-62.
8. Shahriar, H., C Zulkernine, M. (2012). Mitigating program security vulnerabilities: Approaches and challenges. ACM Computing Surveys (CSUR), 44(3), 1-46.
9. Chen, X., Li, M., Jiang, Y., C Sun, Y. (2019). A Comparison of Machine Learning Algorithms for Detecting XSS Attacks. In Artificial Intelligence and Security (pp. 259-270). Springer, Cham.

10. Gupta, S., C Govil, M. C. (2015). Web application vulnerabilities: A survey and comparison of detection techniques. In 2015 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC) (pp. 7-12). IEEE.
11. Wang, R., Xu, G., Zeng, X., Li, X., C Feng, Z. (2018). TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. Journal of Parallel and Distributed Computing, 118, 100-106.
12. Zhou, Y., C Wang, P. (2019). An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. Computers C Security, 82, 261-269.