An implementation of real-time detection of cross-site scripting attacks on cloud-based web applications using deep learning

Isaac Odun-Ayo[1], Williams Toro-Abasi[2], Marion Adebiyi[3], and Oladapo Alagbe[4]
[1,2,4]Department of Computer and Information Sciences, Covenant University, Ota, Nigeria
[3]Department of Computer Science, Landmark University, Omu-Aran, Nigeria
[3]Department of Information Technology, Durban University of Technology, Durban, 4000, South Africa

Article Info

Article history:

Received Nov 12, 2020
Revised Mar 17, 2021
Accepted Aug 16, 2021

Keywords:

Cross-site scripting
Deep learning
Real-time detection

ABSTRACT

Cross-site scripting has caused considerable harm to the economy and individual privacy. Deep learning consists of three primary learning approaches, and it is made up of numerous strata of artificial neural networks. Triggering functions that can be used for the production of non-linear outputs are contained within each layer. This study proposes a secure framework that can be used to achieve real-time detection and prevention of cross-site scripting attacks in cloud-based web applications, using deep learning, with a high level of accuracy. This project work utilized five phases cross-site scripting payloads and Benign user inputs extraction, feature engineering, generation of datasets, deep learning modeling, and classification filter for Malicious cross-site scripting queries. A web application was then developed with the deep learning model embedded on the backend and hosted on the cloud. In this work, a model was developed to detect cross-site scripting attacks using multi-layer perceptron deep learning model, after a comparative analysis of its performance in contrast to three other deep learning models deep belief network, ensemble, and long short-term memory. A multi-layer perceptron based performance evaluation of the proposed model obtained an accuracy of 99.47%, which shows a high level of accuracy in detecting cross-site scripting attacks.

Corresponding Author:

Isaac Odun-Ayo
Department of Computer and Information Sciences
Covenant University
Ota, Nigeria
Email: isaac.odun-ayo@covenantuniversity.edu.ng

1. INTRODUCTION

Cross-site scripting (XSS), whose title can be clearly distinguished from the cascade style sheets (CSS), has been discovered to be a consistent susceptibility in web applications that enables cyber hackers to insert harmful scripts into webpages to allure the attention of victims who may likely end up clicking them. Over the years, the XSS has been recorded as one of the open web application security project (OWASP) top 10 vulnerabilities [1]. XSS attacks would endanger the internet users ' protection, resulting in the loss of useful and sensitive information and user sessions. There are three XSS vulnerability types; document object model (DOM) XSS, reflected XSS, and stored XSS. The most frequently seen XSS type is the reflected XSS, whereby harmful scripts are transcribed into the URL to be used where an attacker invites the user, and the user clicks on such links**.** For stored XSS attack type, a destructive code is injected into the input data by the attacker also, the data stored in the database. It exploits the vulnerability by creating a duplicate webpage that correlates to the intended data and attacks users who access the webpage. The DOM-based XSS would be considered a unique instance of reflected XSS; the harmful coded script inserted by the attacker tends to change the document object model (DOM) and ends up harming the web client [2].

Cloud is an integrated and virtualized data network that is dynamically supported and viewed as a single or multiple centralized cloud services. Cloud computing contracts are supported by service-level agreements which are provided by the cloud service providers (CSPs) to the cloud users, and they govern the quality of service that is to be expected from the CSPs, by the customers [3], [4]. The primary cloud computing service offerings include software-as-a-service (SaaS), platform-as-a-service (PaaS), and infrastructure-as-a-service (IaaS). There are four cloud deployment models and cloud services: private, public, community, and hybrid clouds [3], [5]. Cloud computing, in its various offerings, has been adopted globally, and a lot of web applications are being deployed on cloud infrastructure [6]. This ubiquitous nature of cloud computing is reflected in the wide range of applications that utilize their platforms, which range from entertainment-based applications to time-critical health-based applications [7], [8].

Deep learning (DL) is categorized as a division of machine learning (ML) and is made up of three learning methods, i.e., unsupervised, semi-supervised, and supervised [9]. Consist of strata of artificial neural network (ANN), each layer has some neurons with activation functions that result in non-linear outputs; an approach that is influenced by the human brain's neuronal structure [10], [11]. Deep learning has been an emphasis for numerous scholars and corporations in recent years compared to traditional approaches to machine learning. In Mohammadi *et al.* [12] four ML algorithms were utilized to evaluate deep learning; namely, K-means, decision trees, support vector machine (SVM), and logistic regression, all using Google trends. The outcomes revealed deep learning as "computationally increasingly efficient." Deep learning algorithms are widely used in several domains for classification and prediction purposes [13]. This paper presents a secure framework for real-time detection and prevention of XSS attacks on cloud-based web applications, using deep learning, with a high level of accuracy. The subsequent portions of the paper are as shown in: related work is discussed under section 2; section 3 examines the deep learning framework, and section 4 is a conclusion of the paper.

2. RELATED WORK

In Kronjee, *et al.* [14], a WIRECAML method to detect structured query language (SQL) injection and cross-site scripting vulnerability in web applications running hypertext preprocessor (PHP) was examined. The tool combines machine learning algorithms with data flow analysis. Also, they obtain sufficient data from the National vulnerability database and the SAMPLE dataset to generate a dataset that includes an amount of source code files from PHP. Second, the tool analyses file present in the dataset using the Phply to create abstract syntax trees (ASTs). Control flow graphs (CFGs) are then generated based on the ASTs and uses data flow analysis techniques to extract features from CFGs. In conclusion, the extracted features are added to work and test out a range of different classifiers. This approach realizes a 79% precision rate and a 71% recall rate in terms of Cross-site scripting, not a very satisfying result.

According to Shar, *et al.* [15], presents a machine learning-based tool called Php Miner, which predicts web application XSS vulnerabilities. The tool can also detect File Inclusion vulnerabilities, Remote Code Execution vulnerabilities, and structured query language injection (SQLi) based vulnerabilities. The tool operates based on a combined method of static and dynamic analysis. The code used for validation and sanitization activites possesses features and attributes that can be utilized for vulnerability prediction. The code attributes and their capacities to be used for vulnerability prediction are the primary principles of operation. These attributes and features are known as input validation and sanitization (IVS) attributes. A node within the control flow graph indicates the interaction between a statement and other components like databases is known as a sink. The hybrid analysis is used for information extraction from sinks. Static analysis is used for calculating sink slices. Sanitization functions and validation function types are inferred via the use of dynamic analysis. The information generated via the hybrid analysis is categorized according to the various IVS features and attributes and is placed in a set of attributes for these classifications. Lastly, a supervised predictor and another semi-supervised predictor are constructed for the prediction of web vulnerabilities. The method achieves a 9 percent false score for predicting XSS vulnerabilities, which is quite high and can be improved to a higher level.

Khalid, *et al.* [16], presents a network vulnerability detection approach called NMPREDICTOR. Machine learning algorithms are utilized by this method, alongside the combination of various models of prediction. This experiment's data set is known as the vulnerability dataset of the hypertext preprocessor (PHP). The method is divided basically into two sections. In the first tier, six different models are created from the training set by NMPREDICTOR, which are then used to predict the vulnerability status. By implementing a supervised learning method, the training set data are classified as vulnerable or not. PHP source files are taken as input by the various models and they each output a probability which serves as the target probability of the file. The method also uses results from the six models within the second tier to create another model known as a meta classifier. The precision rate of 84.9% and the recall rate of 85.1% both serve as the best results from their approach.

Shailendra, *et al.* [17], the authors proposed an XSS vulnerabilities detection technique centered on machine learning algorithms, which can be used in social networking services (SNSs). Next, they described and separated XSS features into three categories: standard url features, hypertext markup language (HTML) tag features, and SNS features. Thirdly, their method collected 1000 SNS web pages to create datasets, and features, which consisted of 400 benign web pages and 600 malicious web pages. Finally, machine learning algorithms were used for the data training phase, and the outcomes generated were graded. The authors utilized ten machine learning algorithms to conduct the experiments. The best results derived from the experiment were a 97.2% accuracy rate and an 87% false-positive rate. However, it was observed that the dataset utilized was too small.

Wang, *et al.* [18], an approach to machine learning for detecting Cross-site Scripting in social networks was discussed. They manually removed the characteristic features of the web pages; categorized them into distinct four groups with each group comprising multiple attributes, where three of them are then associated with an online social network; also, adaptive boosting (AdaBoost) was utilized, and a 10-fold cross-validation ADTree algorithms; using AdaBoost, the highest recorded values of 0.941 precision value and 0.939 recall value were realized. These results depicted a false-positive rate of 4.20%, high, and a shallow detection rate.

Fang, *et al.* [19], an approach called deep XSS was used for the detection of cross-site scripting (XSS) based on deep learning. This approach, which entails decoding, generalization, and tokenization techniques, was examined. First of all, word2vec, an extraction application, was used to extract the XSS payloads features. These Cross-site Scripting payloads captured word order information, and each payload was linked to a characteristic vector. Using recurrent neural networks from long short-term memory (LSTM), the detection model was trained and tested. Using this approach, the experimental results showed a precision rate of 99.55% with a recall rate of 97.9% and an F1 score of 98.7%. Nonetheless, the webpage contained JavaScript and HTML code, which are principally sub-standard encoding; hence, word vectors' adoption made the training process too time-consuming and tedious. Their research was not extended to detection in real-time, either.

Goswami, *et al.* [20] examined a method of XSS attack detection based on the clustering of unsupervised attributes, using the Monte Carlo cross-entropy algorithm as the aggregation of ranks. This method was used to classify clusters into two classes, namely, malicious scripts and benign scripts. This method initially utilized a certain degree of heterogeneity for XSS vulnerabilities tests on the client. If the tests exceeded a given threshold value, the request is jettisoned. Otherwise, the request is forwarded to the proxy for additional processing. One downside of this approach is identifying client and proxy servers, thereby lacking flexibility in its mode of operation. Furthermore, the method interferes with system usage.

3. MATERIALS AND METHOD

3.1. Deep learning methods

The research method, the design and development of the deep learning model, and the tool employed in building the cloud-hosted web application are all discussed in this section. The primary aim, as earlier specified, is the development and implementation of a prototype of a model that can perform real-time detection of XSS attacks for cloud-based web applications that utilize deep learning to ensure a secured system. This section offers an overview of the proposed method and adapted framework used to implement the model, a detailed overview of the model architecture, and finally, the system's design and analysis. For this research, deep belief network (DBN), MLP, and LSTM deep learning models will be adopted.

– LSTM: has distinct units in the recurrent hidden layer referred to as memory blocks [21]. It is used to decide when to let the input enter the neuron, remember what was computed in the previous timestamp, and let the output pass on to the next timestamp [12], [21].
– DBN: is made up of a visible layer that corresponds to the inputs and many hidden layers that correspond to latent variables [22]. DBN training is conducted layer by layer; therefore, every layer is treated as a Restricted Boltzmann machine, which is trained on top of the previous layer that was trained [12], [23]. A DBN may also be advanced for cataloging problems under supervised learning [22]. Similar NLP related tasks, for example, classification of text, are performed by DBN models. The models can learn multiplex features within hidden layers and gather more compound functions, which are then used for data demonstration [24]. DBN uses RBM for planning [23].
– Multi-layer perceptron (MLP): is a neural feed-forward network with many hidden (multi-layer) layers. Hidden layers in MLP are fully connected, i.e., each node in each layer is associated with a specific weight to each node of the layer below [25], [26].

3.2. Neural networks notations

The deep learning representation and the forward and backward propagation is shown below. The following are the neural network notations [27].

General comments:

Superscript (i) shows the ith training example while superscript [l] shows the lth layer

Sizes:

·m: number of instances in the dataset
·nx: the size of the input
·ny: the size of the output (or number of classes)
n_h ^([*l*]): number of hidden units of the lth layer
In a for loop, it is possible to denote *n_x*=*n_h* ^([0]) and·ny = ·nh [number of layers +1].
·L: number of layers in the network.
Objects:
·X ∈ Rnx×m is the input matrix
·x(i) ∈ Rnx is the ith example represented as a column vector
·Y ∈ Rny×m is the label matrix
·y(i) ∈ Rny is the output label for the ith example
·W[l] ∈ R is the number of units in the next layer × number of units in the previous layer is the weight matrix, superscript [l] indicates the layer
·b[l] ∈ R is the number of units in the next layer is the bias vector in the lth layer
·yˆ ∈ Rny is the predicted output vector. It can also be denoted a [L] where L is the network's number of layers.

Forward propagation in (1)-(3):

$$a=g^{[l]}(W_x x^{(i)}+b_1)=g^{[l]}(z_1) \text{ where } g^{[l]} \text{ denotes the } lth \text{ layer activation function} \quad (1)$$

$$\hat{y}(i)=\text{softmax}(Whh+b2) \quad (2)$$

$$J(x,W,b,y) \text{ or } J(\hat{y},y) \text{ denote the cost function.} \quad (3)$$

3.3. Deep learning representations

Forward propagation means propagating the computations of all neurons within all layers moving from left to right. The process starts with converting your feature vector(s)/tensors into the input layer and ends with the final prediction generated by the output layer. Forward pass computations occur during training to evaluate the objective/loss function under the current network parameter settings in each iteration and during inference (prediction after training) when applied to new/unseen data. Intermediate variables are sequentially calculated and stored within the neural network's computational graph by forwarding propagation. Forward propagation progresses from the input to the output layer [27]. Backward propagation is a step executed during training to compute the objective/loss function gradient for the network's parameters for updating them during a single iteration of some form of gradient descent. When viewing a neural network as a computation graph, it is named because computing the objective/loss function derivatives at the output layer. It propagates them back towards the input layer to compute results and update all parameters in all layers. Backpropagation has performed an essential role in ANNs since 1982. It is a very effective gradient descent method [28]. It calculates and stores the gradients of intermediate variables and parameters sequentially, within the neural network, and in a reversed order [27].

a. Nodes represent inputs, activations, or outputs
b. Edges represent weights or biases

3.4. Metrics for model performance

The accuracy, ROC-AUC, precision, recall, and F1-Score measures are used as metrics for evaluation.

- Accuracy: the overall effectiveness, precision, and efficacy of the classification model can be described as accuracy [29], [30].
- ROC-AUC: ROC is a probability curve, and AUC represents the degree or measure of separability. They indicate how well a model can differentiate amongst classes [30].
- Precision: this is the ratio of predicted positives, which are accurately positive [30].
- Recall: also known as "sensitivity or true positive rate," speak of the amount of positively predicted actual positive occurrences [29], [30].

– F-Measure: the "F-measure" is recognized as the unifier means of precision and recall [29], [30].

3.5. Adapted model architecture

The adapted model architecture is by [17] titled "XSS Classifier: An efficient XSS attack detection approach based on machine learning classifier on social networking services SNSs." Their framework was dependent upon machine learning classifiers to classify webpages into two classes, namely, XSS or non-XSS. It covers four distinct steps entirely: feature engineering, collection of webpages, generation of datasets, and machine learning classification. The overview is depicted in Figure 1.

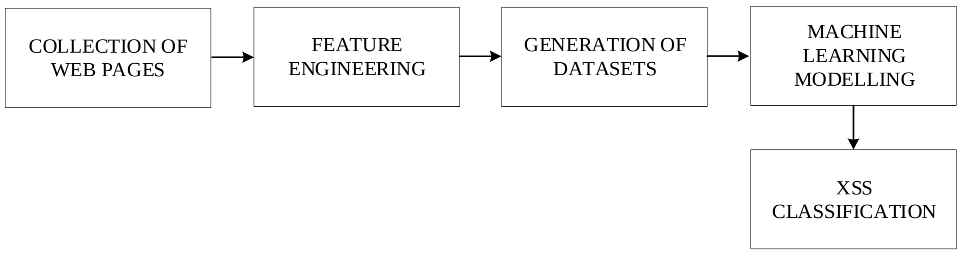


Figure 1. Adapted model architecture

In their method, XSS attack detection was performed using three features: URLs, websites, and SNSs. A dataset was prepared by collecting about 1,000 SNS web pages and removing the features from those web pages. Ten different machine learning classifiers were used in a trained dataset to categorize web pages into two groups: XSS or non-XSS. It was tested using precision and an F1 score to validate the performance.

3.6. Proposed model architecture

The proposed model architecture is adapted from [17] is depicted in Figure 2. Their model architecture comprises four key steps; feature engineering, collection of webpages, generation of datasets, and machine learning classification. The proposed framework utilized the three (3) phases from the adapted framework by [17]: Feature engineering, generation of datasets, and classification.

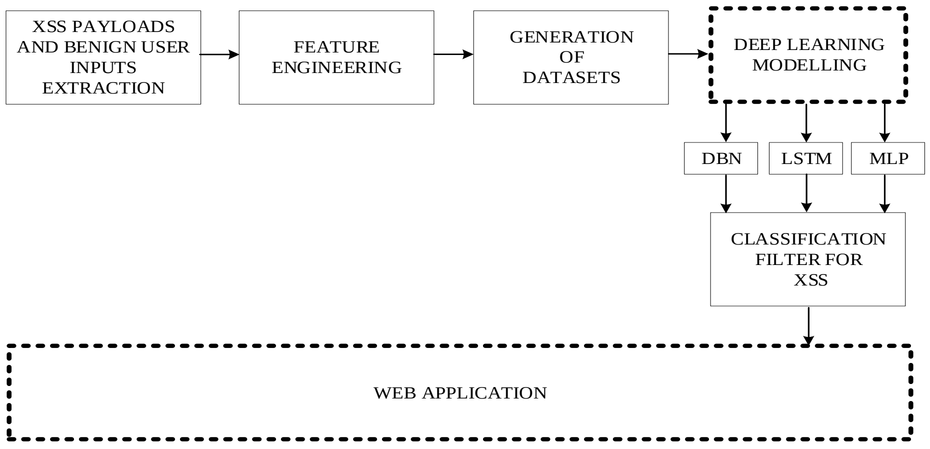


Figure 2. Proposed model architecture

a. XSS payloads and benign user inputs extraction: These are activities intended to generate a dataset for deep learning model training. XSS payloads are malicious queries hackers' type informs to have an illegal effect could consist of a web application. Benign user inputs are typical queries that appear in a web form. Extraction of payloads data would be from online repositories, the building of web crawlers for payloads data extraction from logged XSS websites, and manual curation of the dataset.

b. Feature engineering: this includes feature extraction techniques to extract relevant indicators from the payloads to be fed into the deep learning model. Explorative data analysis (EDAs) of the dataset should come before the feature engineering stage to gain insights on what features are relevant for extraction or transformation.
c. Generation of datasets: the dataset is separated into training and testing sets.
d. Deep learning modelling (DBN, LSTM, and MLP): after successfully extracting relevant features, these algorithms above would be used individually to train models for XSS input detection. Based on the evaluation of the testing dataset, the best performing model would be picked for use in the web application.
e. Ensemble learning: this involves combining the DBN, LSTM, and MLP models to derive another model.
f. Classification filter for malicious XSS queries: for an XSS injection to be successful, a web form containing the relevant XSS input must be successfully submitted to the webserver. The filter would be a backend program to scan web forms before submitting them to the server; to check for possible XSS inputs using the generated machine learning model.

3.7. The data used to bring out the results

The extraction of payloads data were from online repositories, the building of web crawlers for payloads data extraction from logged XSS websites, and manual curation of the dataset The sources of data include: portswigger XSS cheatsheet, owasp cheatsheets for XSS attack and Github repository. The training and test dataset were gotten from these three websites. The raw payloads data was saved in txt format while the raw bening data was saved in csv format. The model's training was done using the Google colab platform, and the codes were prepared in Jupyter notebook.

3.8. Proposed system architecture

The proposed system's structure is founded on a three-tier layered architecture. It comprises the presentation layer, logic layer, and the data layer, with all layers contributing to the system's total workability.

a. Presentation tier: this where the web application of the system runs. It provides an interfacing link for the users to interact with the system. From this layer, the user can register themselves and access other functions of the system.
b. Logic tier: this layer is regarded as the most relevant layer because all the registered functionalities in the logic tier are carried out here. It also comprises the application modules (the filtering program, XSS model API, cloud data create, read, update, and delete CRUD operations).
c. Data tier: the data tier where data is stored and retrieved from. It includes the database (user data, cloud storage, and API event logs).

4. RESULTS AND DISCUSSION

4.1. Model evaluation

After the extraction and gathering of XSS payloads, the dataset for training and testing was done through downloads, the building of web crawlers, and curation. Ten thousand six hundred datasets for payloads data and five thousand datasets for benign data were extracted. The algorithms (DBN, LSTM, and MLP) were used individually to train XSS input, detection models. The best performing model was to be picked for use in the web application based on the testing dataset evaluation. The performance metrics used were: accuracy, ROC-AUC, precision, recall, and F1-score. The modeling; was based on two features:

a. TFIDF: Term frequency and inverse document frequency, computed as a bag-of-word model with each tokenized query as words and their weighted frequencies as vectors.
b. Word embedding: The word embedding model used was FastText by Facebook research.

Model parameters used are discussed below

a. Multi-layer perceptron: a two dense layer with 512 units of outputs and RELU as activation function was used. Also, the number of Epochs was three at 500 batch size each at training. A third dense layer with the neural network's final output was added with Sigmoid as an activation function. Two dropouts at 0.2 were applied for the first two layers consecutively. The loss function used Binary Cross-Entropy, which was optimized using Adams optimization.
b. LSTM: an embedding layer with 128 dimensions and the equal number of features as the dataset was used with the sequence model. 0.2 Dropout was also throughout the network, and the final output layer was activated using Sigmoid. The loss function used binary cross-entropy, which was optimized using Adams Optimization. The number of Epochs was three at 500 batch size for each training or backpropagation.

c. Deep belief network: restricted Boltzmann machine (RBM) was used as first layer activation to extract extra features. After these layers were done, the extra 3 dense layers and activation functions used in MLP were also applied here. The loss function used binary cross-entropy, which was optimized using Adams optimization. the number of Epochs was three at 500 batch size for each training.
d. Ensemble: hard vote ensemble method was used for ensembling in this work. Calculated as the average of all three models for each prediction given.

4.2. Model validation

Model validation are discussed as:

– Training set: 70% Testing set: 30%
– This split percentage was used for both TFIDF and Word Embedding.
– Performance metrics used were: accuracy, ROC-AUC, precision, recall, and F1-Score.
– The model's training was done using the Google Colab platform, and the codes were prepared in Jupyter notebook. The dataset's visualization report is shown in Figure 3, 10600 for attack data and 5000 for benign data. Table 1 presents the metric evaluations using TFIDF as features for the four deep learning models being utilized for this study, and Table 2 presents the metric evaluations using Word Embedding as features for the four deep learning models, respectively.

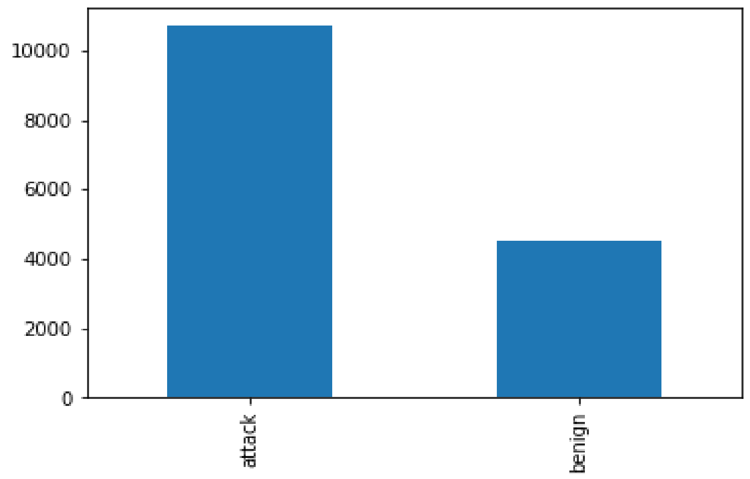


Figure 3. Visualisation report for the dataset

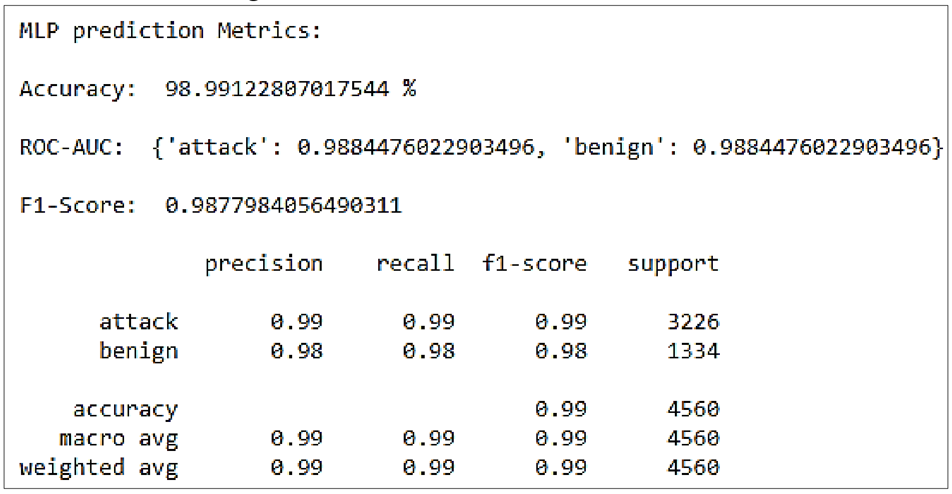
Table 1: Metric evaluations using TFIDF as features

Model	Accuracy	Roc-Auc	Precision	Recall	F1 Score
MLP	98.99%	{'attack':0.98, 'benign':0.98}	{'attack':0.99, 'benign':0.98}	{'attack':0.99, 'benign':0.98}	0.98
LSTM	70.87%	{'attack': 0.5, 'benign': 0.5}	{'attack':1.00, 'benign':0.00}	{'attack':0.71, 'benign':0.00}	0.41
DBN	72.85%	{'attack':0.53, 'benign':0.53}	{'attack':1.00, 'benign':0.07}	{'attack':0.72, 'benign':1.00}	0.48
ENSEMBLE	72.85%	{'attack':0.53, 'benign':0.53}	{'attack':1.00, 'benign':0.07}	{'attack':0.72, 'benign':1.00}	0.48

Table 2. Metric evaluations using word embedding as features

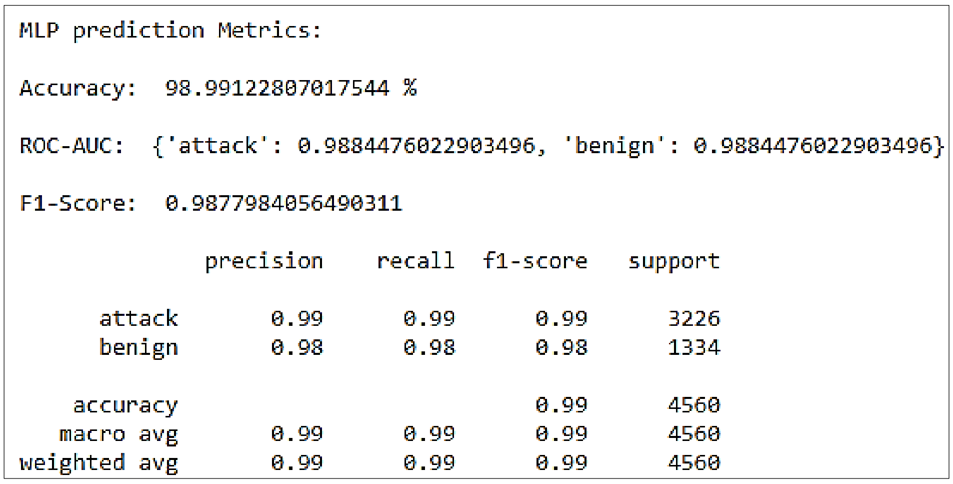
Model	Accuracy	Roc-Auc	Precision	Recall	F1 Score
MLP	99.47%	{'attack':0.99, 'benign':0.99}	{'attack':1.00, 'benign':0.98}	{'attack':0.99, 'benign':1.00}	0.99
LSTM	70.87%	{'attack': 0.5, 'benign': 0.5}	{'attack':1.00, 'benign':0.00}	{'attack':0.71, 'benign':0.00}	0.41
DBN	98.26%	{'attack':0.97, 'benign':0.97}	{'attack':1.00, 'benign':0.94}	{'attack':0.98, 'benign':1.00}	0.97
ENSEMBLE	98.20%	{'attack':0.96, 'benign':0.96}	{'attack':1.00, 'benign':0.94}	{'attack':0.98, 'benign':1.00}	0.97

Figure 4 and Figure 5 shows the classification report on the jupyter notebook for the MLP model using TFIDF and word embeddings.



```
MLP prediction Metrics:

Accuracy:  98.99122807017544 %

ROC-AUC:  {'attack': 0.9884476022903496, 'benign': 0.9884476022903496}

F1-Score:  0.9877984056490311

              precision    recall  f1-score   support

      attack       0.99      0.99      0.99      3226
      benign       0.98      0.98      0.98      1334

    accuracy                           0.99      4560
   macro avg       0.99      0.99      0.99      4560
weighted avg       0.99      0.99      0.99      4560
```

Figure 4. Classification report for the MLP using TFIDF



```
MLP prediction Metrics:

Accuracy:  98.99122807017544 %

ROC-AUC:  {'attack': 0.9884476022903496, 'benign': 0.9884476022903496}

F1-Score:  0.9877984056490311

              precision    recall  f1-score   support

      attack       0.99      0.99      0.99      3226
      benign       0.98      0.98      0.98      1334

    accuracy                           0.99      4560
   macro avg       0.99      0.99      0.99      4560
weighted avg       0.99      0.99      0.99      4560
```

Figure 5. Classification report for the MLP using TFIDF

4.3. The experiments that works on the system

The malicious scripts were injected on the web application through the input forms. The deep learning model was integrated at the back end of the web application to filter the inputs against malicious xss queries. An API that has a trained deep learning model was built. The API call was done from the frontend using Javascript.

```
with open('mlp_model.pkl', 'rb') as file:
        mlp_model = pickle.load(file)
@app.route("/", methods=["POST","GET"])
def root():
 if request.method == "POST":
                user_input = request.form['user_query']
                prediction = mlp_model.predict(user_input)
 return jsonify({'prediction':prediction})
 else:
 return jsonify({'prediction':false})
```

This shows the javascript code which filters the users input against the malicious queries and the web application is depicted in Figure 6.

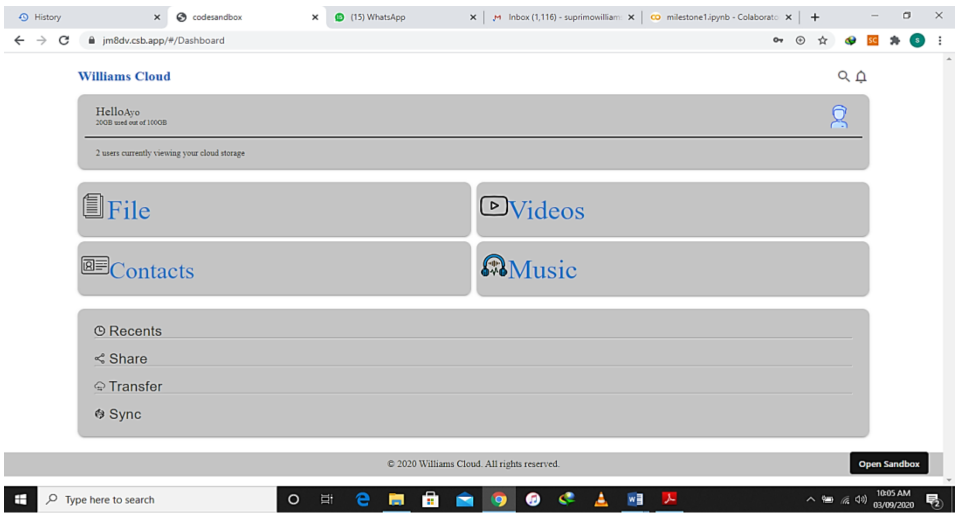


Figure 6. The web application

4.4. Discussion

LSTM performed the poorest using both TFIDF and Word Embedding. LSTM had 0 precision rates for benign queries and 1 for attack queries, because it was just labeling test set as attacks the test set is only for evaluating the model. Combined with its poor recall, it would lead to very high false positives in practical scenarios indicated by an F1-score less than 0.5. Wrong data belling for LSTM, hence it's poor performance. Using F1-score as the metric, MLP performed best for both TFIDF and Word Embedding features. Word Embedding features performed slightly better TFIDF using the MLP model. Also, MLP made a very high accuracy of 99.4% on the test set and had a recall and precision for both attack and benign queries above 0.9, which should reduce false positives practical scenario as indicated by F1-score above 0.5 and very slightly close to 1 at 0.993. A brief description of what the performance Metrics implies. Performance metrics used were: Accuracy, ROC-AUC, Precision, Recall, and F1-Score.

a. Accuracy: this measures how many correct predictions each model got in percentage. However, the easiest method of measuring performance is if the data is imbalanced or there is bias in classification, then an accuracy metric won't effectively be able to tell.
b. For example; if there are 90 normal queries and 10 XSS attack queries, and the model classified all set as normal; it would achieve a 90% accuracy, but in this case, the model has not been able to effectively identify attack query due to possible bias or inability to handle data imbalance. Hence accuracy is not always the best validation metric.
c. Precision: following the example above, the precision for normal query prediction would be a ratio of 0.9; however, for attack queries, it would be a precision ratio of 0. Precision, unlike accuracy, actually measures how well the model did for each of the classes and not an aggregation of all classes; which tend to lead to less false positives and identification bias in the model.
d. Recall: still following the example, recall measures how many attack queries were predicted as attacks and vice versa for the normal queries. There were 90 normal queries, and all 90 were predicted correctly, so the recall ratio for the normal query would be 1. There were ten attack queries for attack query, and none were predicted correctly so that recall would be 0. This metric is also known as sensitivity.
e. F1-score: F1-score combines precision and recall using harmonic mean to generate a metric for evaluation. Usually, a metric or ratio above 0.5 and closer to 1 means the model has better precision and recall, which means it tends to be more robust to data bias and false positives and vice versa.
f. ROC-AUC: this is a combination of receiver operating characteristics and area under the curve, which is used to measure how well a model can distinguish between classes using random samples at different thresholds. A ratio of above 0.5 and close to 1 indicates better performance, and vice versa indicates poor performance. It measures sensitivity, specificity, and false-positive rates of the model, for its final evaluation.

In addition to the explanation above regarding the metric generated by each model, MLP outperformed all other models in all metrics. It is precise, and recall values for predicting attacks are very promising and show it is not prone to false positives or bias from random samples as indicated by the high ROC-AUC score of 0.99. Previous records' predominant success and network flow LSTM indicates that it is best suited for sequence problems like speech, thereby requiring larger data. At the same time, DBN adds an extra layer of feature extraction using Restricted Boltzmann machine which is to be fed to any other network layer, so it is not a system of layers like CNN, MLP. hence its performance is subjected to another network parameters and the extraction from restricted Boltzmann machine activation.

MLP, on the other hand, performs well due to its simplistic network layer and less need for a larger dataset as opposed to other "more-advanced" deep learning models. So it trains and fits the data size much faster and effectively than "more-advanced" algorithms that require a larger set of activation functions and parameter activation. Also, the No free lunch theorem concept, the parameters used for MLP here, as discussed in the previous mail above, could have been more optimal for this data. The parameters used for the others could be less effective. For example, when using a softmax loss function during training, the deep learning models' accuracy metric drastically reduced below 30%; however, when changed to binary cross-entropy, there was a significant boost. So the subjective parameters of the model also indicate poor or high performance.

5. CONCLUSION

In this work, a model was developed to detect XSS attacks using the MLP deep learning model, following a comparative review of its output compared to three other deep learning models, Ensemble, LSTM, and DBN. Research on the project led to the creation of Williams Cloud. This web-based proof-of-concept framework incorporated an MLP deep learning model to detect and handle XSS attack scripts injected into a web application. The results of machine learning training on four algorithms showed that MLP performed best in detecting XSS attacks based on the evaluation metrics. The MLP model achieved 98.99 percent using TFIDF as a feature and 99.47 percent using word embedding as a feature. This work contributes immensely to the knowledge that can also be further developed and adopted to counter and prevent other web-based attacks. Also, results obtained from users' evaluation will be made available for further research.

ACKNOWLEDGMENT

The authors recognize the sponsorship and support from Covenant University via the Centre for research, innovation, and discovery (CUCRID). There are no conflicts of interest from this study.

REFERENCES

[1] J. Fonseca, N. Seixas, M. Vieira and H. Madeira, "Analysis of Field Data on Web Security Vulnerabilities," in *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 2, pp. 89-100, March-April 2014, doi: 10.1109/TDSC.2013.37.

[2] X. Guo, S. Jin and Y. Zhang, "XSS Vulnerability Detection Using Optimized Attack Vector Repertory," *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2015, pp. 29-36, doi: 10.1109/CyberC.2015.50.

[3] P. Mell and T. Grance, "The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology," *Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Gaithersburg*, pp. 97-101, 2012.

[4] I. Odun-Ayo, V. Geteloma, I. Eweoya, and R. Ahuja, "Virtualization, Containerization, Composition, and Orchestration of Cloud Computing Services," in *Computational Science and Its Applications – ICCSA 2019*, 2019, pp. 402-417, doi: 10.1007/978-3-030-24305-0_30.

[5] I. Odun-Ayo, C. Okereke, and H. Orovwode, "Cloud computing and internet of things: Issues and developments," *The World Congress on Engineering 2018*, London, 2018, p. 2235.

[6] W. A. R. W. M. Isa, A. I. H. Suhaimi, N. Noordin, A. F. Harun, J. Ismail, and R. A. Teh, "Cloud computing adoption reference model," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 1, pp. 395-400, 2019, doi: 10.11591/ijeecs.v16.i1.pp395-400.

[7] S. A. Hameed, A. Nirabi, M. H. Habaebi, and A. Haddad, "Application of mobile cloud computing in emergency health care," *Bulletin of Electrical Engineering and Informatics*, vol. 8, no. 3, pp. 1088-1095, 2019, doi: 10.11591/eei.v8i3.1498.

[8] I. Odun-Ayo, V. Geteloma, S. Misra, R. Ahuja, and R. Damasevicius, "Systematic Mapping Study of Utility-Driven Platforms for Clouds," *Lecture Notes in Electrical Engineering*, vol. 605, no. 2020, pp. 762-774, 2020, doi: 10.1007/978-3-030-30577-2_68.

[9] J. Majidpour and H. Hasanzadeh, "Application of deep learning to enhance the accuracy of intrusion detection in modern computer networks," *Bulletin of Electrical Engineering and Informatic*, vol. 9, no. 3, pp. 1137–1148, 2020, doi: 10.11591/eei.v9i3.1724.

[10] Y. Zhou, M. Han, L. Liu, J. S. He and Y. Wang, "Deep learning approach for cyberattack detection," *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 262-267, doi: 10.1109/INFCOMW.2018.8407032.

[11] C. Wang, S. Dong, X. Zhao, G. Papanastasiou, H. Zhang and G. Yang, "SaliencyGAN: Deep Learning Semisupervised Salient Object Detection in the Fog of IoT," in *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2667-2676, April 2020, doi: 10.1109/TII.2019.2945362.

[12] M. Mohammadi, A. Al-Fuqaha, S. Sorour and M. Guizani, "Deep Learning for IoT Big Data and Streaming Analytics: A Survey," in *IEEE Communications Surveys & Tutorial*s, vol. 20, no. 4, pp. 2923-2960, Fourthquarter 2018, doi: 10.1109/COMST.2018.2844341.

[13] M. Akour, H. Al Sghaier, and O. Al Qasem, "The effectiveness of using deep learning algorithms in predicting students achievements," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 1, pp. 387-393, 2020, doi: 10.11591/ijeecs.v19.i1.pp387-393.

[14] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018, pp. 1-10, doi: 10.1145/3230833.3230856.

[15] L. K. Shar, L. C. Briand and H. B. K. Tan, "Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning," in *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 6, pp. 688-707, 1 Nov.-Dec. 2015, doi: 10.1109/TDSC.2014.2373377.

[16] M. N. Khalid, H. Farooq, M. Iqbal, M. T. Alam, and K. Rasheed, "Predicting Web Vulnerabilities in Web Applications Based on Machine Learning," *Intelligent Technologies and Application,* Springer Singapore vol. 932, pp. 473-484, 2019, doi: 10.1007/978-981-13-6052-7_41.

[17] R Shailendra, K. Pradip and J. Hyu, "XSSClassifier: An Efficient XSS Attack Detection Approach Based on Machine Learning Classifier on SNSs," *Journal of Information Processing Systems*, vol. 13 no. 4, pp. 1014-1028, 2017, doi: 10.3745/JIPS.03.0079.

[18] R. Wang, X. Jia, Q. Li and S. Zhang, "Machine Learning Based Cross-Site Scripting Detection in Online Social Network," *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, 2014, pp. 823-826, doi: 10.1109/HPCC.2014.137.

[19] Y. Fang, Y. Li, L. Liu, and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, pp. 47-51, 2018, doi: 10.1145/3194452.3194469.

[20] S. Goswami, N. Hoque, D. K. Bhattacharyya, and J. Kalita, "An unsupervised method for detection of XSS attack," *International Journal of Network Security*, vol. 19, no. 5, pp. 761-775, 2017, doi: 10.6633/IJNS.201709.19(5).14.

[21] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," no. Cd, 2014, [Online]. Available: http://arxiv.org/abs/1402.1128.

[22] A. M. Abdel-Zaher and A. M. Eldeib, "Breast cancer classification using deep belief networks," *Expert Systems with Applications*, vol. 46, pp. 139-144, 2016, doi: 10.1016/j.eswa.2015.10.015.

[23] U. Narayanan, V. Paul, and S. Joseph, "A novel approach to big data analysis using deep belief network for the detection of android malware," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 16, no. 3, pp. 1447-1454, 2019, doi: 10.11591/ijeecs.v16.i3.pp1447-1454.

[24] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, p. 102419, 2020, doi: 10.1016/j.jisa.2019.102419.

[25] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014, doi: 10.1017/atsip.2013.9.

[26] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," *Briefings in bioinformatics*, vol. 18, no. 5, pp. 851-869, 2017, doi: 10.1093/bib/bbw068.

[27] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27-48, 2016, doi: 10.1016/j.neucom.2015.09.116.

[28] M. Zulqarnain, R. Ghazali, Y. M. M. Hassim, and M. Rehan, "A comparative review on deep learning models for text classification," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, no. 1, pp. 325-335, 2020, doi: 10.11591/ijeecs.v19.i1.pp325-335.

[29] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427-437, 2009, doi: 10.1016/j.ipm.2009.03.002.

[30] D. Satybaldina and G. Kalymova, "Deep learning bas[n," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 21, no. 1, pp. 398-405, 2021, doi: 10.11591/ijeecs.v21.i1.pp398-405.

BIOGRAPHIES OF AUTHORS

Isaac A. Odun-Ayo was born in Ilesha, Nigeria, in 1962. He received the B.S, M.S and Ph.D degrees in Computer Science from the University of Benin, Benin City, Nigeria. Between 2010 and 2013 he was a faculty and Director Information and Communication Technology at the National Defence College, Abuja, Nigeria. He joined the faculty of Covenant University, Ota, Nigeria as a Senior Lecturer in October 2016. He is the author of one book and more than 40 journal and conference articles in Cloud Computing. His research interest include cloud computing, human resource management, e-governance and software engineering. Dr. Odun-Ayo is a recipient of the National Productivity Order of Merit Award, Nigeria, to contribute to computing. He is a member of the Nigeria Computer Society (NCS), Computer Professionals of Nigeria (CPN), International Association of Engineers (IAENG), Institute of Electrical and Electronics Engineers (IEEE) and Member Information Science Institute (ISI).

Marion Adebiyi received a BSc. degree in computer science from University of Ilorin, Kwara State, Nigeria in 2000. Her MSc. and Ph. D degree in computer science, bioinformatics Option from Covenant University, Ota, Nigeria in 2008 and 2014 respectively. She is a Senior lecturer in Computer and Information Sciences Department of Covenant University, and on deployment to Landmark University, Omu-Aran, Kwara State, Nigeria in 2018, and currently a PDF researcher at Durban University of Technology, South Africa. She authors two books, over ten chapter in book and more than 40 articles, and 25 conferences and workshops. Dr. Adebiyi heads the H3Africa projects coordinating Division. The Entomology and Data Management Division of Covenant University Bioinformatics Research (CUBRe) group and her research interests include bioinformatics, high throughput data analytics, genomics, proteomics, transcriptomics, homology modeling, and Organism's inter-pathway analysis.

Toro-Abasi Williams received a BSc. degree in computer science from Afe Babalola University, Ado-Ekiti, Ekiti State, Nigeria in 2016. He received a MSc. Degree from the Department of Computer and Information Sciences, Covenant University, Nigeria in 2020. He takes tutorials for several courses at the postgraduate level. He has a passion for academics and research in computer science. Williams has some cloud computing publications. His research interests include cloud computing, mobile computing, artificial intelligence, machine learning, and software engineering.

Oladapo A. Alagbe is a research assistant and a postgraduate student at the Department of Computer and Information Sciences, Covenant University, focusing on Management and Information Systems. He received a B.Tech in Computer Science from Ladoke Akintola University of Technology (LAUTECH), Ogbomosho. He is passionate about Mobile, Fog and Cloud Computing, Cyber Security, and how they affect organizations.