Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.040121

ARTICLE

Injections Attacks Efficient and Secure Techniques Based on Bidirectional Long Short Time Memory Model

Abdulgbar A. R. Farea[1], Gehad Abdullah Amran[2,*], Ebraheem Farea[3], Amerah Alabrah[4,*], Ahmed A. Abdulraheem[5], Muhammad Mursil[6] and Mohammed A. A. Al-qaness[7]

[1]School of Big Data & Software Engineering, Chongqing University, Chongqing, 401331, China

[2]Department of Management Science Engineering, Dalian University of Technology, Dalian, 116024, China

[3]Software College, Northeastern University, Shenyang, 110169, China

[4]Department of Information Systems, College of Computer and Information Science, King Saud University, Riyadh, 11543, Saudi Arabia

[5]Department of Management Science and Engineering, South China University of Technology, Guangzhou, 510641, China

[6]Department of Computer Engineering and Mathematics, University of Rovira i Virgili, Tarragona, Spain

[7]College of Physics and Electronic Information Engineering, Zhejiang Normal University, Jinhua, 321004, China

*Corresponding Authors: Gehad Abdullah Amran. Email: jehad.westran@gmail.com; Amerah Alabrah. Email: aalobrah@ksu.edu.sa

Received: 06 March 2023 Accepted: 13 June 2023 Published: 08 October 2023

ABSTRACT

E-commerce, online ticketing, online banking, and other web-based applications that handle sensitive data, such as passwords, payment information, and financial information, are widely used. Various web developers may have varying levels of understanding when it comes to securing an online application. Structured Query language SQL injection and cross-site scripting are the two vulnerabilities defined by the Open Web Application Security Project (OWASP) for its 2017 Top Ten List Cross Site Scripting (XSS). An attacker can exploit these two flaws and launch malicious web-based actions as a result of these flaws. Many published articles focused on these attacks' binary classification. This article described a novel deep-learning approach for detecting SQL injection and XSS attacks. The datasets for SQL injection and XSS payloads are combined into a single dataset. The dataset is labeled manually into three labels, each representing a kind of attack. This work implements some pre-processing algorithms, including Porter stemming, one-hot encoding, and the word-embedding method to convert a word's text into a vector. Our model used bidirectional long short-term memory (BiLSTM) to extract features automatically, train, and test the payload dataset. The payloads were classified into three types by BiLSTM: XSS, SQL injection attacks, and normal. The outcomes demonstrated excellent performance in classifying payloads into XSS attacks, injection attacks, and non-malicious payloads. BiLSTM's high performance was demonstrated by its accuracy of 99.26%.

KEYWORDS

Web security; SQL injection; XSS; deep learning; RNN; LSTM; BiLSTM

1 Introduction

In recent years, the demand for web applications has rapidly increased. Using the web may significantly reduce enterprises' distribution costs for information, products, and services [1]. Vulnerability trends show that applications have a large number of vulnerabilities. In addition, SQL injection and cross-site scripting have been widely documented application vulnerabilities. SQL injections are more dangerous attacks because they can affect vital databases for any company. The exposure must be repaired at the code level, while the developer or programmer must take remedial action from a response perspective [2]. As long as web applications are not very secure, several well-designed injections and malicious scripts can be performed on the database and the victim browser. Intruders may exploit this situation and do malicious actions such as malware distribution, cookie theft, and session hijacking to steal users' credentials. This sensitive data could be transferred to any third party, including a hacker's or intruder's server [3].

A SQL injection attack is a type of attack that targets web applications and any applications dealing with the database. The attacker can exploit the weakness of user input filtering and inject illegal SQL queries or malicious payloads, which execute on the database as a legal query. SQL Injection Attacks can destroy the target web application's confidentiality, integrity, and availability. The attacker can gain any sensitive information from the database or even destroy the database [4]. On the other hand, Web applications can be vulnerable to Cross-Site Scripting (XSS) attacks, in which an attacker takes control of a user's browser and executes malicious Hyper Text Markup Language (HTML)/JavaScript code in order to steal the user's credentials. This can be done in a number of ways, like the theft of cookies, the hijacking of a user's session, the distribution of malware, or a redirect to a malicious page [5].

According to the similarities between the way of working of these two attacks, which lead us to work on these two attacks. Both SQL injection and XSS are code injection attacks that use the same mechanism to attack the website [6]. The intruder can inject SQL or XSS payloads on website user inputs or Uniform Resource Locators (URLs). The intruder can perform malicious actions on the database, damaging the attacked data or even stealing website cookies. SQL injection and XSS attacks can be used to serve these purposes. A web application consists of server-side, client-side, and database servers. An XSS attack is a malicious script executed on the client browser, whereas an SQL injection attack is a malicious SQL query executed on the database server. According to OWASP (Open Web Application Security Project), SQL injection and cross-site scripting (XSS) are ranked in OWASP's top ten 2017 web application security risks and vulnerabilities [7].

Many articles have been published in the area of web application attacks as well as in the field of web application security. However, most studies still have drawbacks and weak points, such as performing multiclassification attacks, which can specify the types of detected attacks and the correct classification rate. Some current work uses binary classification (normal and malicious payloads), which deals with all types of attacks as a single attack without any differentiation between them [8–11]. It is worth mentioning that some research focused only on XSS [8,12], and other studies focused only on SQL injection attacks [9]. The most relevant work is by Abaimov et al. [6], who built (CODDLE) a convolutional deep neural network model to detect SQL injection and XSS attacks, but he used a separate dataset for each attack and obtained less than our results.

Our key contributions to this research are as follows:

- We propose a new methodology based on the BiLSTM recurrent neural network for detecting and multi-classify SQL injection and XSS attacks.

- The proposed model utilized textual data containing SQL injection and XSS payloads to classify them into three classes, which are XSS, SQL injection, and normal payloads. To achieve high performance in detection, the model utilized Recurrent Neural Networks (RNNs), which consider the sequence of sentences or texts, resulting in excellent results in all evaluation metrics, including accuracy, precision, recall, and F1 score.

Following the structure of the paper, the rest of the work is organized as follows: Section 2 presents relevant studies that connect anomaly detection techniques for web application attacks with deep learning. Section 3 introduces the methodology of the proposed model. Model implementation and experimental setup are explained in detail in Section 4. Section 5 shows the results of the proposed model with further details and discussion. In Section 6, the conclusions are summed up, and ideas for further studies are suggested.

2 Related Works

Recently much work has been carried out for the discrimination of different kinds of Cross-Site Scripting (XSS) and SQL injection attacks [10,12].

2.1 SQL Injection

Xie et al. [13] described in their study entitled Elastic-Pooling, conventional neural network (CNN)-based (EP-CNN), a deep learning model used for detecting SQL injection attacks in web applications based on weblogs. EP-CNN model Used the Word2vec method to convert the original query to a vector. Then, an elastic pooling layer is added to three layers of convolution kernels after the convolution layers. Padding in convolution must be employed to keep the input and output dimensions equal when using several convolution layers of different sizes. The outside of this layer passed on other convolution kernel layers without trimming the data. This model achieved an accuracy of 98.7% on the test set. Chen et al. [11] proposed using word embedding and CNN Multilayer Perceptron (MLP) algorithms to prevent SQL injection attacks as a novel approach. The HTTP requests are denoised and decoded, then Word2Vec produces word embeddings of these decoded characters, trains an MLP, CNN model, and then utilizes the classifier to identify fraudulent requests. Both models successfully detect SQL injection attacks. MLP is 98.5% accurate, whereas CNN is 98.2% accurate. In their study, Hasan et al. [14] created a heuristic algorithm based on machine learning that was trained on a limited amount of data. They also developed a Graphical User Interface (GUI) application for five models. The Ensemble Boosted Trees model had the most accurate results, with an accuracy rate of 93.8%. However, the researchers suggest adding more infected statements to the dataset to improve the algorithm's accuracy. Abdalla et al. [15] aimed to prevent SQL Injection Attacks (SQLIA) with an adaptive model that relies on runtime validation to detect such attacks. However, the model's accuracy was restricted to 86.6%, and no machine-learning mechanisms were implemented. The investigation used a dataset containing 4201 entries.

2.2 Cross-Site Scripting (XSS)

In 2018, DeepXSS [8] used the RNN LSTM algorithm and the Word2vec technique to detect XSS attacks. The proposed method maps each XSS payload to a feature vector using the Word2vec CBOW model. The LSTM technique is then used to train and test XSS payload datasets. The DeepXSS model performed well, with an F1 score accuracy of 98.7%, precision of 99.57%, and recall of 97.9%. DeepXSS can be enhanced to detect more web app attacks. Sharma et al. [16] emphasized the importance of feature set extraction in detecting web-based attacks. They propose an approach

to extract feature sets that can significantly improve results when used with a machine learning-based intrusion detection model. The authors conducted an experiment using the CSIC HTTP 2010 dataset and the Weka tool, which involved three steps. Firstly, the data was pre-processed with a python script. Secondly, features were extracted from the dataset based on specific keywords before being fed into Weka for data modeling. Lastly, the data was fed into three Machine learnings (ML) models, J48, OneR, and Naïve Bayes, in Weka, with J48 producing the best results compared to other classifiers. These findings have implications for developing effective intrusion detection systems for web-based attacks. Kaur et al. [17] developed a machine-learning model to detect malicious attack vectors before a victim's browser processes them. To identify blind XSS and stored XSS attacks, they utilized the Linear Support Vector classification algorithm. The authors gathered features by examining attackers' JavaScript events and scripts on the website. The experiment was carried out on Mutillidae, a free website that is vulnerable to attacks, using a linearly separable dataset. The model achieved a high detection accuracy rate of 95.4%, with a recall value of 0.951 and a false positive rate of 0.111.

2.3 SQL Injection and XSS Attacks

In 2017, Liang et al. [18] proposed a novel deep learning approach for detecting unusual requests by entailing the unsupervised training of two RNNs. With a sophisticated recurrent unit (Gated Recurrent Unit (GRU) or LSTM unit) for learning the typical request patterns utilizing only typical requests, followed by supervised learning of a neural network classifier that uses the outcome of RNNs as input to differentiate between abnormal and legal requests. The model used normal requests to familiarize the RNNs (LSTM and GRU) with legitimate request patterns. The first RNN looks at URL path structure, while the second looks at query parameter structure. In the final step, an MLP model was trained on the output of prior models to distinguish between normal and abnormal URL occurrence probability sequences. The models' accuracy on the CSIC dataset is 97.8% for GRU and 98.4% for LSTM. GRU and LSTM models achieve 98.5 percent accuracy on the WAF logs dataset. This URL attack classification model did not classify each URL assault according to type. In their study, Tang et al. [9] analyzed the textual content of URLs and developed eight distinctive features. Furthermore, they employed the Payloads dataset and utilized ASCII code to map character sequences into a numerical matrix. The dataset was subsequently trained and tested using LSTM and MLP models with appropriate hyperparameters. Results indicated an accuracy of 99.67% and 97.68% for LSTM and MLP, respectively. Zhang et al. [19] proposed a method called Adversarial Perturbation for Model Stealing Attack (APMSA) to protect Deep Learning models deployed in the cloud from being stolen by attackers. The method adds noise to the input queries to hide the internal information of the model and prevent attackers from reverse-engineering a substitute model. The limitations of this paper are that the proposed method may not be effective if the attacker conceals the query sample to look like a normal benign query, and the detection techniques may fail to capture this malicious behavior. Additionally, the proposed method requires the Deep Learning model to be processed before deployment, which may reduce the availability and utility of the model.

It should be noted that distinct datasets and feature extraction methods were employed for both models. Gong et al. [20] employed model uncertainty to estimate the deep learning model's prediction accuracy. This model consists of two parts. The first part is a CNN model, which takes weblogs as inputs and extracts their features. The second is the Bayesian model. which is used as a classifier to classify each weblog containing a URL, response code, user agent, and source address into a web attack or normal log. This model achieved an accuracy of 98.38%, a precision of 99.84%, and a recall of 94.77% as its performance metrics. Mo et al. [21] presented an intrusion detection system based on Bi-LSTM (BL-IDS) model, which uses LSTMs and bidirectional recurrent neural

networks to detect web attacks. The Word2vec toolbox converted the text into a word vector using the word embedding NLP technique (Skip-gram model). The CSIC 2010 HTTP dataset was used. The Bi-LSTM model was used to classify HTTP requests as legal or illegal for ten epochs of batch training methods. Almost all the models proposed to detect web application attacks classify attacks as having a binary classification (normal and malicious payloads). However, the proposed model is a multi-classification model that classifies each web attack according to type. Abramov et al. [6] built (CODDLE) convolutional Deep Neural Network model to detect SQL injection and XSS attacks. Although CODDLE's best performance was up to 94% accuracy, 99% precision, and 93% recall value, it used a separate dataset where each attack dataset was trained and tested separately by a binary classification model. Our model combined the two datasets into a single dataset, trained and tested the dataset with a multi-classification model, and achieved high-performance metrics values. The related works are summarized below in Table 1.

Table 1: Existing prior studies about cross-site scripting (XSS) and SQL injection attacks

	Problem statement	Proposed model	Results	Drawbacks
[11]	Word embedding and CNN MLP algorithms were used to propose a new SQLIA prevention method. Denoise and decode HTTP requests, use Word2vec to create word embeddings of the decoded characters, train a CNN, MLP model, and use the classifier to identify fraudulent requests.	User's HTTP request dataset. Word embedding created a word vector from the HTTP request. CBOW creates word embedding. MLP and Convolutional Neural Networks were used to train and test the dataset (CNN).	Both models detected SQL injection attacks. CNN has 98.2% accuracy and MLP 98.5.	It detects only one type of web attack, which is SQLIA.
[22]	This article suggests an Elastic-Pooling CNN-based (EP-CNN) model to detect SQL injection attacks in weblog-based web applications.	Model proposed Word2vec converted the query to Vector. Three convolution kernels follow. After elastic pooling. Other convolution kernel layers passed outside this layer. This method produces a two-dimensional matrix without data trimming.	It detected new SQL injection attacks by matching irregular characteristics with 99.93% accuracy in the training set. Test set accuracy is 98.7%.	The needing to enhance this model to become multi-classification and cover other web attacks such as XSS attacks.
[8]	DeepXSS is a deep learning approach for detecting XSS attacks based on the RNN LSTM algorithm and Word2vec technique.	The Word2vec Continuous Bag of Words (CBOW) model mapped XSS payloads to feature vectors to convert input texts into numeric vectors. LSTM is used to train and test XSS payload datasets.	The proposed model performed well with an F1 accuracy score of 98.7%, a precision rate of 99.5%, and a recall rate of 97.9%.	The model is not generalized to detect more web application attacks.
[18]	They provide a deep learning method for identifying unusual requests. This method involves unsupervised RNN training (RNNs). The complicated recurrent unit (LSTM or GRU unit) to learn regular request patterns using only normal requests, followed by supervised training of a neural network classifier that uses RNN output to discriminate anomalous and legal requests.	Tokenization precedes URL request. Word embedding then assigned real numbers to each word. After that, two RNN models—LSTM and GRU—were trained using typical requests to familiarize them with legitimate request patterns. The first RNN analyzes URL route structures, whereas the second analyzes query parameter structures. Finally, an MLP model trained on the preceding models' probability sequences of URLs with the output label to distinguish between normal and anomalous URL occurrence probability sequences.	This model achieves 97.8% GRU and 98.4% LSTM accuracy on the CSIC dataset. GRU and LSTM have 98.5% and 98.3% accuracy on WAF logs, respectively.	Do not classify each URL attack based on type.

(Continued)

Table 1 (continued)

	Problem statement	Proposed model	Results	Drawbacks
[6]	Build Convolutional Deep Neural Network model to detect SQL injection and XSS attacks.	SQLI/XSS symbols were encoded as command/symbol values using a customized pre-processing method. The first value is a simple numeric label, while the second is commend/symbols. Dataset payloads came from GitHub. Payloads are then converted into pairs (value and category). The CNN model trained and tested this dataset.	The CNN model achieves up to 94% accuracy, 99% precision, and a 93% recall value as the best performance.	It used every attack as a separated dataset and did not group them into a single dataset.
[22]	BL-IDS analyzes HTTP requests to detect Web threats using LSTMs and Bi-LSTMs.	Word embedding NLP used the word2vec toolkit to convert text into word vectors (Skip-gram model). Used CSIC 2010 HTTP dataset. The Bi-LSTM model classified HTTP requests as normal or abnormal for ten batch training epochs.	The model performs well with 98% accuracy and 99% precision.	It did not classify each HTTP attack to its type.
[8]	A neural network-based SQL injection detection method is presented in this paper. They get accurate user URL access log data from the ISP to prove their strategy is real, effective, and feasible. After that, we statistically analyze normal and SQL injection data. Based on statistical findings, we train an MLP model with eight features. Using the Payloads dataset, this study detected SQLI with an LSTM model. And compare it to the MLP model's output.	Based on statistical findings, they analyze URL text and create eight features. They used Payloads instead. By mapping characters with American Standard Code for Information Interchange (ASCII) code. An LSTM model is trained on this dataset with the right hyperparameters. Both models' outcomes were compared.	The MLP model's precision and accuracy are 99.55 and 99.86 percent after training. LSTM has a precision of 99.85% and an accuracy of 98.69%. MLP and LSTM accuracy tested at 99.67% and 97.68%, respectively.	Different datasets and feature extraction methods were used for both models. Compared to other studies, the dataset is small. Only SQLi attacks are detected.

3 Methodology

In this section, we are going to illustrate our work and the proposed methodology. Our model starts by fitting on the textual datasets for both attack payloads: SQL injection and XSS. Then, we combine these two datasets into a single dataset formed from three classes. After that, label each class with a specific symbol. Different machine learning and deep learning algorithms will be used to multi-classify the attacks' payloads. Different pre-processing techniques will be used to clean the data and convert the dataset from its textual form into a numeric form that can be easily manipulated by machine or deep learning algorithms.

In this research, we propose an intelligent web attack classification. Fig. 1 depicts the methodology used in this paper to detect multi-class attacks.

As shown in Fig. 1, the methodology started by collecting the experimental dataset and constructing a multi-classification model that can detect two types of injection attacks and normal payloads. This model consists of various steps of pre-processing the textual dataset, encoding the payloads, and then converting the textual dataset into a numeric matrix using the principle of word embedding. The numeric dataset is split into a training dataset and a testing dataset. Finally, our model used the

BiLSTM algorithm to classify the payloads into three groups. The following steps illustrate each phase used in our methodology.

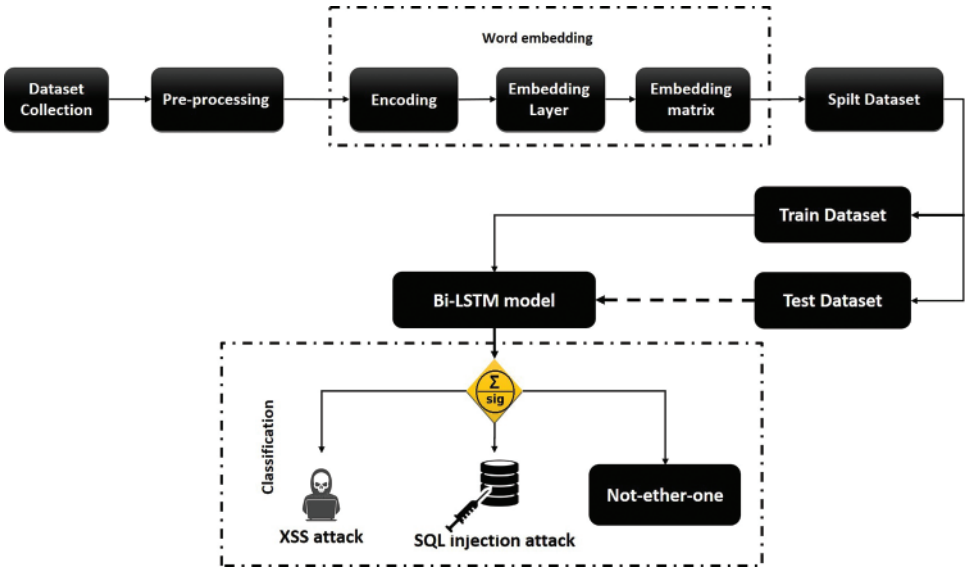

Figure 1: Flow graph of proposed approach

3.1 Dataset Collection

Resources. XSS payloads have been collected from XSS_dataset [23] with 5282 normal payloads and 7368 abnormal payloads. At the same time, SQL injection payloads have been collected from the SQL injection dataset [24] and SQL-injection-payload-list in GitHub [25], including 3005 normal payloads and 1822 abnormal payloads.

3.2 Data Pre-Processing

First, we merged the two datasets into a single dataset. Then, we labeled the payloads into three classes: 1 for XSS attacks, 2 for SQL injection attacks, and 0 for not-either-one payloads. Following that, we cleaned the payloads of special characters like “, ’, ” $, %, &, @, and so on. Finally, we used stemming text normalization to normalize each payload word.

For example, the following SQL injection and XSS payloads

‘and 1 in (select min(name) from sysobjects where xtype = ‘U’ and name > ‘.’)–

<label onpointerdown = alert (1) > XSS </label>

Will be after pre-processing as follows:

and 1 in selecting min name from sysobjects where xtype U and name

label onpointerdown alert 1 XSS label

3.3 Word Embedding

Word embedding is a Natural Language Processing (NLP) technique that converts Natural Language text into a vector representing the text. Basically, word embedding maps a word to a vector using a dictionary [26].

In this article, first, we used one-hot encoding to convert every word into a number based on vocabulary size. This number represents the index of the word in the encoding matrix. Because word embedding inputs should be the same size, we pad the inputs with zeros (padding = 40). Finally, we used the TensorFlow embedding layer to perform word embedding and generate the embedding matrix.

3.3.1 Encoding

One-hot encoding is a way to change categorical variables into a format that deep learning algorithms can use to make better predictions. In the proposed model, we used one hot encoding with word embeddings. One hot encoding will convert the text into a sparse matrix with a lot of zeros, and only one value will indicate the location of this word in the predefined dictionary of words. We used word embedding with one-hot encoding to overcome the sparse matrix from the one-hot process into a dense matrix representing feature representation.

3.3.2 Embedding Layer

A layer that can only be utilized as the initial layer of a model is known as an embedding layer. The layer converts positive integers (indices) into dense vectors of a predetermined size by multiplying them together. Word embeddings may be learned from text data and re-used in different projects at different times. They may also be trained as part of the process of fitting a neural network to text input. The embedding of a word in the learned vector space is referred to. The vector space location of a word is determined by the words surrounding it when it is employed.

Keras has an embedding layer that can be used for neural networks that work with textual data because each word has a unique number, and the input data needs to be encoded as an integer. This means that a number represents each word. A layer called the embedding layer is set up with random weights. It will learn an embedding for all of the words in the training dataset.

In our model, we set up an embedding layer as the flowing, input_dim = 5000, representing the vocabulary size in the text data, which means that each word will be encoded by a number from 0 to 4999 in the embedding matrix. output_dim = 40, which means each word will be placed in a vector space with 40 dimensions. input_length = 100, which represents the length of input sequences.

3.3.3 Embedding Matrix

An embedding matrix is an idea that tries to solve this problem of how to show relationships. First, we choose a dimension of meaning. This can be a little bit random. Let us say we decide that all meaning can be mapped to a three-dimensional space that is not real. Theoretically, that would mean that each word would be a single point in a 3D space, and three numbers could describe the position of each word in that space (x, y, z). But in reality, meaning is too complex to fit well into three dimensions. Usually, we use something like 300 dimensions, and all words map to some point in this 300-dimensional hyperspace and are defined by 300 numbers. The 300 numbers that tell us what a word means are called the "embedding" for that word.

3.4 Dataset Splitting

There are a total of 17,478 samples in our dataset collection, which represents the attack's payloads. The total number of samples in the trainset reached 13,982, with about 3,496 samples used as a test set.

3.5 Long Short-Term Memory (LSTM)

RNNs are artificial neural networks that perform well with sequential inputs. RNN is designed to connect events from the previous state to the current state by storing environmental data in an inner state [18]. RNNs achieved excellent results in dealing with most sequential data problems because they consider the sequence of the sentences or texts. So, it is a powerful tool for dealing with time series information that contains correlations between data points near each other in the sequence [27].

LSTM is an algorithm for solving the weaknesses of the RNN algorithm, which are vanishing and exploding gradient problems. The LSTM layer consists of memory blocks, which are recurrently connected blocks, as shown in Fig. 2. Each one contains recurrently connected memory cells and three multiplicative units (input, output, and forget gates) that act as continuous analogs for the cells' write, read, and reset operations [28]. A memory cell is a unit in LSTM networks that stores the state or the value. LSTM replaces nodes in hidden layers with one or more memory cells called memory blocks. Activation functions are used in the LSTM architecture instead of gates. The output from the previous layer is stored in gates, and functions determine whether the output will be used as input for the next hidden layer [29].

$$i_t = \sigma\left(W_i X_t + W_i h_{t-1} + b_i\right) \tag{1}$$

$$f_t = \sigma\left(W_f X_t + W_f h_{t-1} + b_f\right) \tag{2}$$

$$o_t = \sigma\left(W_o X_t + W_o h_{t-1} + b_o\right) \tag{3}$$

$$\widetilde{c}_t = \sigma\left(W_c X_t + W_c h_{t-1} + b_c\right) \tag{4}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \widetilde{c}_t \tag{5}$$

$$h_t = o_t \otimes \tanh(c_t) \tag{6}$$

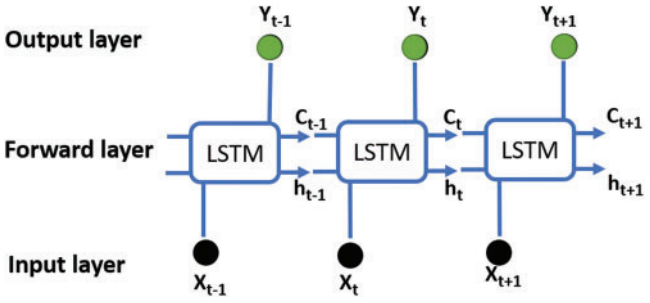

Figure 2: LSTM layer with time series

The equation above and Fig. 3 illustrate the structure of the LSTM layer, where i_t, f_t, o_t are the input gate, forget gate, and output gate, respectively. Whereas c_t, c̃_t is the new state and candidate states of the memory cell, respectively, ct is the current state. The weight matrices are W_i W_c W_f W_o, and the biases are b_i b_f b_o. Sigmoid and hyperbolic tangent functions are identified as $\sigma()$ and tanh().

3.6 Bidirectional Long Short-Term Memory (BiLSTM)

The main idea of BiLSTM is that each training sequence is presented forward and backward to two independent recurrent networks coupled to the same output layer [28].

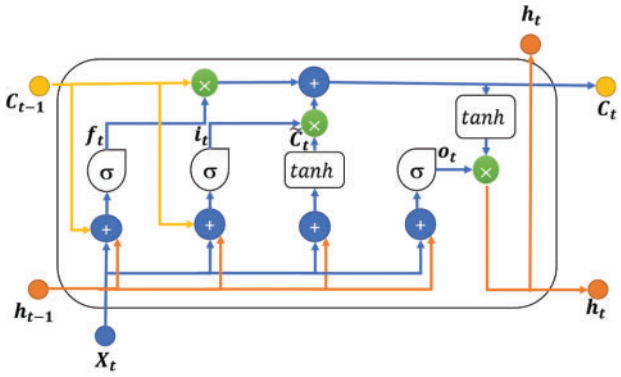

Figure 3: LSTM architecture

BiLSTM is designed to access both sentence directions (preceding and succeeding). Because it combines forward and backward LSTM layers, as illustrated in Fig. 4, The networks are trained over time using the backpropagation principle [30].

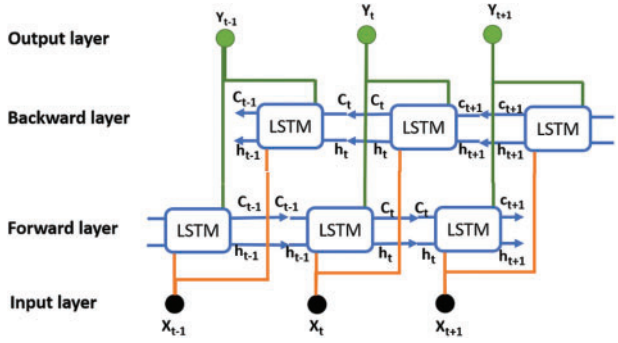

Figure 4: BiLSTM model

4 Model Implementation and Experimental Setup

This section will go over the specifics of how this model will be fully implemented, with an explanation of the implemented model and the hyperparameter of the implemented model. Also, we explained the evaluation matrices used to evaluate our model. Moreover, the optimizer and loss functions are applied in this model.

The experimental setup was conducted on a computer with an Intel(R) Xeon(R) CPU @ 2.30 GHz processor. The GPU was NVIDIA_SMI Tesla K80 with 12 GB RAM. Python 3 was used as a programming language with Keras and the TensorFlow 2.6.0 framework.

4.1 Dataset Preparing

Before passing the data into the model, the data must be well prepared and cleaned. Several steps are used for preparing and pre-processing the dataset to produce cleaned data. First, we collected the dataset for both attacks separately. Then we combined the datasets into a single dataset. After that, we labeled the data into three groups: 0 for valid payloads, 1 for XSS attacks, and 2 for SQL injection attacks. The final step is skipping unwanted characters and cleaning the data using a Porter stemmer. Fig. 5 illustrates the sequence of these steps.

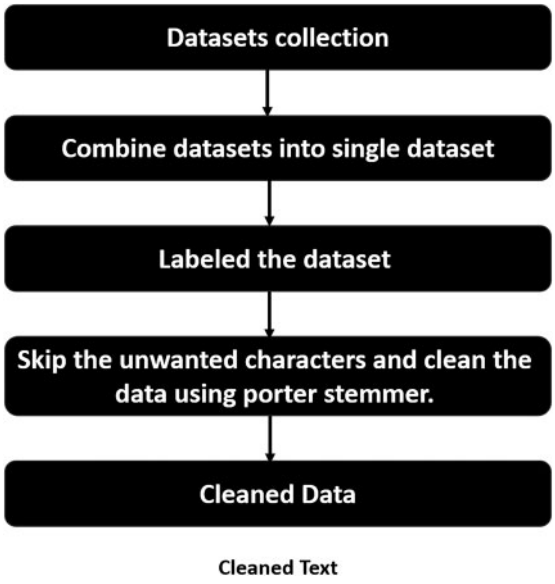

Figure 5: Dataset preparing process

4.2 The Implementation of the Proposed Model

In our proposed model, we used a sequential model. We started this model by declaring the embedding layer with a parameter of 40 as embedding vector features and 100 as sentence length. This is followed by a bidirectional LSTM layer with 100 neurons in the forward and 100 neurons in the backward layers. Then drop out the layer to prevent overfitting. Finally, our model ends with a dense layer with three output neurons. Fig. 6 illustrates the flow graph of the implementation of the proposed model. Moreover, Fig. 7 shows the sequential summary of the proposed model.

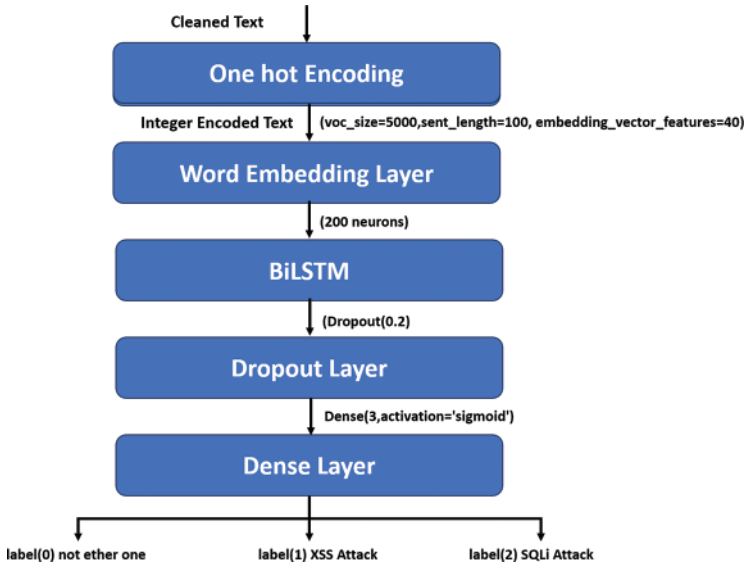

Figure 6: Flow graph of the implementation of the proposed model

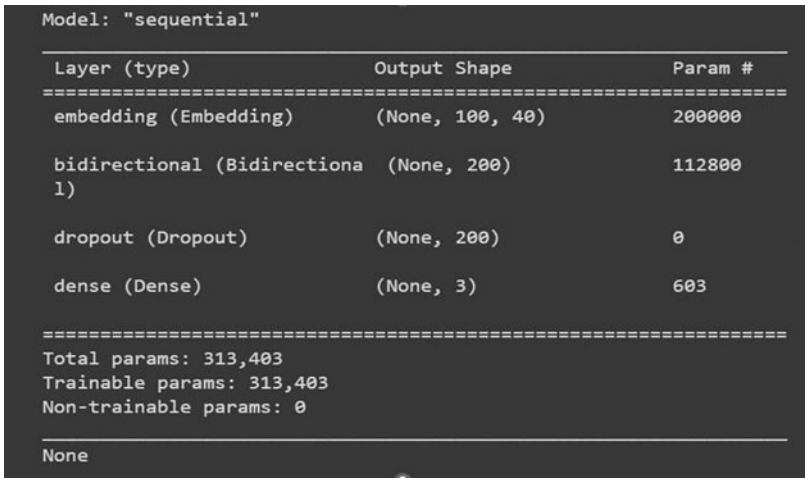

Figure 7: Sequential summary of the proposed model

4.3 Model Hyperparameters

In this phase, the hyperparameters were set. First, 40 features were determined to be the model's input, which means that the input size was set to 40 input layers. Thus, the LSTM hidden layers adopted one layer with 100 neurons for each direction, one layer as the forward layer and one as the backward layer of the BiLSTM scheme, and the output layer had three nodes. Each node represents a particular class. Thirty epochs and 128 batches with a learning rate of 0.01 were used to train the neural network. We have also selected Sigmoid as the activation function and utilised sparse_categorical_crossentropy as our loss function. Finally, Adam's optimizer was selected to update network weights iteratively based on training data. Moreover, it has many benefits over classical stochastic gradient descent. It is straightforward to implement, computationally efficient, has little memory requirements, and is well suited for large problems in terms of data and parameters. These hyperparameters are illustrated in Table 2.

Table 2: Hyperparameter of the model

Hyper parameter	Description	Setting
Vocabulary size	The maximum range indicators of a particular word in the victor	5000
Sentence length	The maximum length of the sentence	100
Embedding vector features	No. of features to be extracted by embedding layer	40
Input size	No. of the neurons at the input layer	40
Hidden layer	No. of the hidden layers in the model	1 (BiLSTM)
Hidden size	No. of the neurons at the hidden layer	$100 * 2$
output size	No. of the neurons at the output layer	3
Epochs	No. of epochs	30
batch size	Batch size	128

4.4 Evaluation Metrics

Accuracy, recall, F1 score, and precision are used for the evaluation phase. The four possible combinations are denoted by the symbols True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) in the model outputs for the test set data and the actual labels of the data. For the training and test sets, we assessed the performance of a neural network model on these four variables.

$$\text{Accuracy} = \frac{TN + TP}{TP + FP + TN + FN} \tag{7}$$

$$\text{Recall} = \frac{TP}{FN + TP} \tag{8}$$

$$\text{Precision} = \frac{TP}{FP + TP} \tag{9}$$

$$\text{F1} - \text{score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{10}$$

5 Results and Discussion

The total payloads in our dataset are 17,478 samples. The number of samples in the train set reached 13,982, and about 3,496 were used as a test set. As shown in Table 3, the dataset was trained and tested using various models. Random Forest with StratifiedKFold with n_splits = 5, random_state = 100, and RandomizedSearchCV with n_jobs = −1, n_iter = 20, verbose = 2, Logistic Regression with tol = 1e−4 and, Support Vector Machine (SVM) with kernel = "rbf" and MLP. We found that our model (the BiLSTM) did an excellent job of classifying the payloads into three classes (e.g., cross-site scripting (XSS) attacks, injection attacks, and non-malicious payloads). A high level of performance was achieved at 99.2% in terms of all evaluation metrics (e.g., accuracy, recall, precision, and F1 score).

Table 3: Different models comparison based on performance

Algorithm	Accuracy	Precision	Recall	F1 score
Random forest	97.88%	97.90%	97.88%	97.89%
SVM	88.22%	88.76%	88.22%	88.36%
Logistic regression	71.02%	73.49%	71.02%	70.57%
MLP	90.99%	91.49%	90.99%	91.15%
Our model	99.26%	99.26%	99.25%	99.24%

5.1 Confusion Matrix

Essentially, the confusion matrix is a cross table that records how many occurrences occurred between two raters, together with their true/actual classification and their expected classification [31].

The confusion matrix in Fig. 8 showed that the BiLSTM model did great in the three class detections. In class 1 (Not-ether-one), the overall samples were 1658; 1642 were classified correctly, whereas 16 were classified incorrectly. For class 2, that stands for XSS attacks. It was 14,766 samples.

This model classified 1473 as an XSS attack, and only one sample was classified as an SQL injection attack. The third class is SQL injection attacks. Of the overall samples (364), 355 were classified correctly, whereas nine were classified incorrectly.

		Predicted			
	Classes	**Not-ether-one**	**XSS**	**SQL injection**	**Total**
Actual	**Not-ether-one**	**1642**	**2**	**14**	**1658**
	XSS	**0**	**1474**	**2**	**1476**
	SQL injection	**8**	**1**	**355**	**364**
	Total	**1650**	**1477**	**371**	**3496**

Figure 8: Confusion metrix

The BiLSTM reached a loss of 0.03 in validation loss and 0.0020 in training loss at the 30th epoch. On the other hand, the accuracy at the 30th epoch reached 0.9926 in terms of validation accuracy and 0.9970 in training accuracy, as illustrated in Figs. 9 and 10.

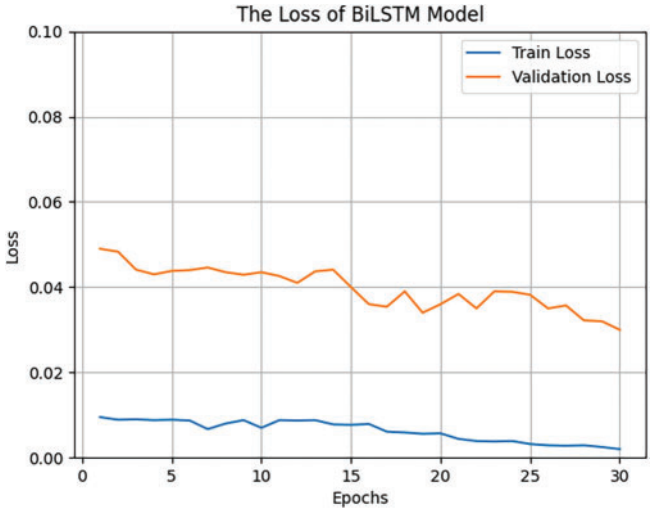

Figure 9: Loss curve

As shown in Table 4, the model work separately and detects XSS attacks with a precision of 0.9952, a recall of 0.9903, and an F1 score of 0.9927. The results were the same for a Not-ether-one payloads class. SQL injection attacks are also detected with a 0.9595 precision, a 0.9753 recall, and a 0.9673 F1 score.

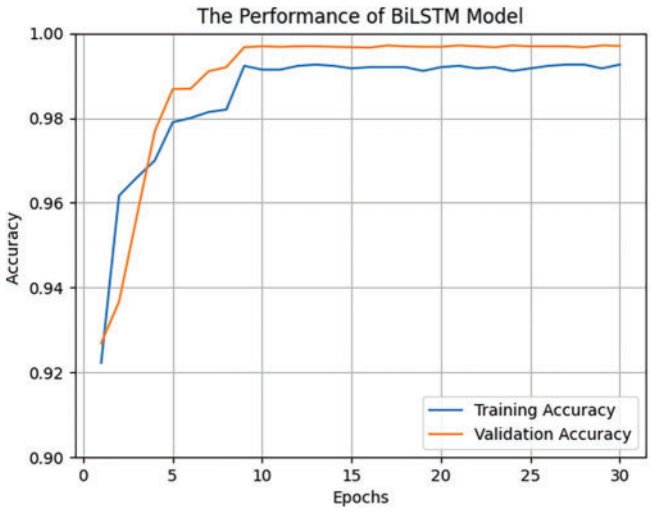

Figure 10: Accuracy curve

Table 4: BiLSTM performance of each class separately

Class	Precision	Recall	F1 score
XSS attacks	0.9952	0.9903	0.9927
SQL injection attacks	0.9595	0.9753	0.9673
Not-ether-one payloads	0.9952	0.9903	0.9927

5.2 Binary Classification for XSS and SQL Injection Attacks

For performing binary classifying on XSS and SQL injection attacks with the same model, we labeled both XSS and SQL injection payloads in the dataset as 1. In contrast, the normal payloads were labeled as 0. Binary_crossentropy was utilized as a loss function with an output layer of two neurons. The results shown in Table 5 point out that our model can also classify these two attacks as having malicious or non-malicious payloads.

Table 5: Binary classification *vs.* multi-classification

Classes	Accuracy	Precision	Recall	F1 score
Binary classification	99.17%	99.18%	99.16%	99.17%
Multi-classification	99.26%	99.26%	99.25%	99.24%

6 Conclusion and Future Work

This paper develops a method based on deep learning to classify SQL injection and Cross-Site scripting attacks. This model used the BiLSTM recurrent neural network principle for training the payload dataset. The suggested model demonstrated that BiLSTM is extremely useful for detecting web application attacks such as XSS and SQL injection with high accuracy and efficiency. The

results obtained in this study reached 99.260%, 99.261%, 99.259%, and 99.248% in terms of accuracy, precision, recall, and F1 score, respectively. For future work, we may extend this research to detect more attacks, such as phishing sites and Distributed Denial-of-Service (DDoS) Attacks. Furthermore, applying oversampling techniques to resolve the imbalanced dataset is highly suggested.

Acknowledgement: We gratefully thank King Saud University for Supporting Researchers Project Number (RSP2023R476), King Saud University, Riyadh, Saudi Arabia.

Funding Statement: This work was funded by Researchers Supporting Project Number (RSP2023R476), King Saud University, Riyadh, Saudi Arabia.

Author Contributions: Abdulgbar A. R. Farea, Gehad Abdullah Amran contributed equally as co first authors. Study conception and design: Abdulgbar A. R. Farea, Gehad Abdullah Amran; data collection: Abdulgbar A. R. Farea, Gehad Abdullah Amran, Ebraheem Farea, Amerah Alabrah, Ahmed A. Abdulraheem, Muhammad Mursil and Mohammed A. A. Al-qaness; analysis and interpretation of results: Abdulgbar A. R. Farea, Gehad Abdullah Amran, Ebraheem Farea, Amerah Alabrah, Ahmed A. Abdulraheem, Muhammad Mursil and Mohammed A. A. Al-qaness; draft manuscript preparation: Abdulgbar A. R. Farea, Gehad Abdullah Amran. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. K. Baranwal, "Approaches to detect SQL injection and XSS in web applications," *EECE 571b, Term Survey Paper*, 2012.

[2] R. Johari and P. Sharma, "A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection," in *Int. Conf. on Communication Systems and Network Technologies, CSNT*, Rajkot, Gujarat, India, pp. 453–458, 2012.

[3] S. K. Mahmoud, M. Alfonse, M. I. Roushdy and A. B. M. Salem, "A comparative analysis of cross site scripting (XSS) detecting and defensive techniques," in *2017 Eighth Int. Conf. on Intelligent Computing and Information Systems (ICICIS)*, Cairo, Egypt, pp. 36–42, 2017.

[4] J. Hasan, A. M. Zeki, A. Alharam and N. Al-Mashhur, "Evaluation of SQL injection prevention methods," in *2019 8th Int. Conf. on Modeling Simulation and Applied Optimization, ICMSAO*, Manama, Bahrain, pp. 1–6, 2019.

[5] S. Gupta and B. B. Gupta, "Cross-site scripting (XSS) attacks and defense mechanisms: Classification and state-of-the-art," *International Journal of Systems Assurance Engineering and Management*, vol. 8, pp. 512–530, 2017.

[6] S. Abaimov and G. Bianchi, "CODDLE: Code-injection detection with deep learning," *IEEE Access*, vol. 7, pp. 128617–128627, 2019.

[7] "OWASP Top 10 2017," OWASP Foundation Oracle Healthcare Data Repository Secure Development Guide, vol. 8, no. 3, 2017. https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

[8] Y. Fang, Y. Li, L. Liu and C. Huang, "DeepXSS: Cross site scripting detection based on deep learning," in *Proc. of the 2018 Int. Conf. on Computing and Artificial Intelligence*, Chengdu, China, pp. 47–51, 2018.

[9] P. Tang, W. Qiu, Z. Huang, H. Lian and G. Liu, "Detection of SQL injection based on artificial neural network," *Knowledge-Based Systems*, vol. 190, pp. 105528, 2020.
[10] M. T. Muslihi and D. Alghazzawi, "Detecting SQL injection on web application using deep learning techniques: A systematic literature review," in *2020 Third Int. Conf. on Vocational Education and Electrical Engineering (ICVEE)*, Surabaya, Indonesia, pp. 1–6, 2020.
[11] D. Chen, Q. Yan, C. Wu and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *Journal of Physics: Conference Series*, Changsha, China, vol. 1757, no. 1, pp. 012055, 2021.
[12] V. S. Stency and N. Mohanasundaram, "A study on XSS attacks: Intelligent detection methods," *Journal of Physics: Conference Series*, vol. 1767, no. 1, pp. 12047, 2021.
[13] X. Xie, C. Ren, Y. Fu, J. Xu and J. Guo, "SQL injection detection for web applications based on elastic-pooling CNN," *IEEE Access*, vol. 7, pp. 151475–151481, 2019.
[14] M. Hasan, Z. Balbahaith and M. Tarique, "Detection of SQL injection attacks: A machine learning approach," in *2019 Int. Conf. on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, IEEE, pp. 1–6, 2019.
[15] H. Abdalla, E. Elsamani, A. Abdallah and R. Elhabob, "An efficient model to detect and prevent SQL injection attack," *Journal of Karary University for Engineering and Science*, pp. 1858–8034, 2022.
[16] S. Sharma, P. Zavarsky and S. Butakov, "Machine learning based intrusion detection system for web-based attacks," in *2020 IEEE 6th Int. Conf. on Big Data Security on Cloud (BigDataSecurity), IEEE Int. Conf. on High Performance and Smart Computing,(HPSC) and IEEE Int. Conf. on Intelligent Data and Security (IDS)*, Baltimore, MD, USA, IEEE, pp. 227–230, 2020.
[17] G. Kaur, Y. Malik, H. Samuel and F. Jaafar, "Detecting blind cross-site scripting attacks using machine learning," in *Proc. of the 2018 Int. Conf. on Signal Processing and Machine Learning*, Shanghai, China, pp. 22–25, 2018.
[18] J. Liang, W. Zhao and W. Ye, "Anomaly-based web attack detection: A deep learning approach," in *Proc. of the 2017 VI Int. Conf. on Network, Communication and Computing*, Kunming, China, pp. 80–85, 2017.
[19] J. L. Zhang, S. Peng, Y. S. Gao, Z. Zhang and Q. H. Hong, "APMSA: Adversarial perturbation against model stealing attacks," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1667–1679, 2023.
[20] X. Gong, Y. Zhou, Y. Bi, M. He, S. Sheng *et al.,* "Estimating web attack detection via model uncertainty from inaccurate annotation," in *6th IEEE Int. Conf. on Cyber Security and Cloud Computing, CSCloud 2019 and 5th IEEE Int. Conf. on Edge Computing and Scalable Cloud, EdgeCom 2019*, Paris, France, pp. 53–58, 2019.
[21] X. Mo, P. Chen, J. Wang and C. Wang, "Security and privacy in new computing environments," in *SPNCE: Int. Conf. on Security and Privacy in New Computing Environments*, Tianjin, China, vol. 284, pp. 96–104, 2019.
[22] H. Saiyu, J. Long and Y. Yang, "Bl-IDS: Detecting web attacks using Bi-LSTM model based on deep learning," in *Security and Privacy in New Computing Environments: Second EAI Int. Conf., SPNCE 2019*, Tianjin, China, pp. 551–563, 2019.
[23] Kaggle, *Cross Site Scripting XSS Dataset for Deep Learning*. [Online]. Available: https://www.kaggle.com/syedsaqlainhussain/cross-site-scripting-xss-dataset-for-deep-learning
[24] Kaggle, *SQL Injection Dataset*. [Online]. Available: https://www.kaggle.com/syedsaqlainhussain/sql-injection-dataset
[25] *SQL Injection Payload List*. [Online]. Available: https://github.com/payloadbox/sql-injection-payload-list
[26] A. V. Nimkar and D. R. Kubal, "A survey on word embedding techniques and semantic similarity for paraphrase identification," *International Journal of Computational Systems Engineering*, vol. 5, no. 1, pp. 36, 2019.
[27] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[28] S. Zhang, D. Zheng, X. Hu and M. Yang, "Bidirectional long short-term memory networks for relation classification," in *Proc. of the 29th Pacific Asia Conf. on Language, Information and Computation*, Shanghai, China, pp. 73–78, 2015.
[29] S. Hochreiter and J. Schmidhuber, "LSTM can solve hard long time lag problems," *Advances in Neural Information Processing Systems*, vol. 9, pp. 473–479, 1997.
[30] G. Liu and J. Guo, "Bidirectional LSTM with attention mechanism and convolutional layer for text classification," *Neurocomputing*, vol. 337, pp. 325–338, 2019.
[31] M. Grandini, E. Bagli and G. Visani, "Metrics for multi-class classification: An overview," arXiv: 2008.05756, 2020.