Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH Open Access

An LSTM based cross-site scripting attack detection scheme for Cloud Computing environments

Xiaolong Li[1*], Tingting Wang[1], Wei Zhang[1], Xu Niu[1], Tingyu Zhang[1], Tengteng Zhao[1], Yongji Wang[2] and Yufei Wang[2]

Abstract
Cloud Computing plays a pivotal role in facilitating the Internet of Things (IoT) and its diverse applications. Users frequently access and store data on remote servers in Cloud Computing environments through web browsers. Consequently, attackers may exploit vulnerabilities in web browsing to embed malicious code into web pages, enabling them to launch attacks on remote servers in Cloud Computing environments. Due to its complexity, prevalence, and significant impact, XSS has consistently been recognized as one of the top ten web security vulnerabilities by OWASP. The existing XSS detection technology requires optimization: manual feature extraction is time-consuming and heavily reliant on domain knowledge, while the current confusion technology and complex code logic contribute to a decline in the identification of XSS attacks. This paper proposes a character-level bidirectional long-term and short-term memory network model based on a multi-attention mechanism. The bidirectional long-term and short-term memory network ensures the association of current features with preceding and subsequent text, while the multi-attention mechanism extracts additional features from different feature subspaces to enhance the understanding of text semantics. Experimental results demonstrate the effectiveness of the proposed model for XSS detection, with an F1 score of 98.71%.

Keywords Network security, XSS detection, Bidirectional long-term and short-term memory network, Multi-head Attention mechanism

Introduction

The issue of XSS attacks has gained significant attention due to the rising number of vulnerabilities and the potential security risks they pose. The "China Internet Cyber Security Report" [1] by the CNCERT reveals that XSS vulnerabilities accounted for nearly half of the 1,700 high-risk vulnerabilities discovered in Internet financial websites in 2018. The Open Web Application Security Project (OWASP) [2] also highlights the prevalence of XSS attacks, with two-thirds of applications being vulnerable to such attacks.XSS attacks pose a particularly significant threat to Cloud Computing resources as they can lead to various security issues, including data theft, unauthorized access, and service disruption. To mitigate these risks, it is crucial to develop effective security mechanisms for detecting and preventing XSS attacks in Cloud Computing environments.

While research on XSS attack detection has made progress, its technology is facing severe challenges [3–8]. XSS attacks are increasingly complex and adaptive, utilizing obfuscation techniques to evade detection [9, 10]. Traditional rule-based and signature-based approaches

*Correspondence:
Xiaolong Li
lxl-777333@163.com
[1] Beijing Institute of Control and Electronics Technology, Muxidi North Street, 100038 Beijing, China
[2] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 100876 Beijing, China

have limitations in effectively identifying and mitigating these evolving attack vectors.To address these challenges, researchers and practitioners have turned to machine learning algorithms to improve the accuracy and efficiency of XSS attack detection. However, traditional machine learning models also encounter difficulties in the field of XSS detection, including their inability to effectively handle evolving XSS scripts over time and the requirement for extensive preprocessing efforts to capture the complex patterns present in XSS attacks [11, 12].

Therefore, this study aims to overcome the limitations of traditional machine learning models by proposing a novel approach that utilizes a character-level bidirectional long-term and short-term memory network model based on a multi-attention mechanism. This approach is designed to capture the contextual information and semantic meaning of textual data, thereby enhancing feature extraction and detection of obfuscation techniques. The contributions of this article can be summarized as follows:

- Automated feature extraction: This study proposes an improved network model structure to automate the process of feature extraction, thereby enhancing the efficiency and effectiveness of XSS detection.
- Detection of complex obfuscation techniques: Recognizing the increasing complexity of obfuscation techniques employed in XSS attacks, this article introduces advanced mechanisms to detect and decipher these techniques, resulting in improved accuracy in XSS detection.
- Optimization of semantic features: This article emphasizes the importance of further exploration and refinement of semantic features in XSS detection. By leveraging a multi-attention mechanism, the proposed model extracts distinctive characteristics from different feature subspaces, enabling a better understanding of textual context and enhancing the detection of XSS attacks.

The rest of the paper is structured as follows: Section "Preparatory knowledge" provides an introduction to the preparatory knowledge, including LSTM and attention mechanisms. Section "The model of detection" presents the model structure and algorithms. Section "Experimental results and analysis" describes the experimental setup and analysis. Section "Related work" concludes the paper with a summary of the main findings and contributions.

Preparatory knowledge

LSTM

Long Short Term Memory (LSTM) is an optimized network compared to traditional recurrent neural networks (RNNs) [13]. Traditional RNNs suffer from the problems of gradient explosion and vanishing, which limit their effectiveness in learning long text sequences. However, LSTM addresses these issues by utilizing time sequence information to retain valuable information in the text for long-term memory. This ensures long-distance dependencies between features, thereby optimizing the detection effect of the recurrent neural network [13–16].

Compared to the traditional recurrent neural network, LSTM still calculates the hidden state ht based on the hidden state calculated from the current sequence point and the previous sequence point. The difference is that in order to avoid issues like gradient explosion, LSTM introduces input gate it, forget gate ft, output gate ot, and an internal memory unit ct. The input gate structure mainly controls the degree of influence of the current state value on the memory unit, while the forget gate structure controls the degree to which information in the previous state's memory unit is forgotten. The output gate structure controls the degree to which the current output state is affected by the current memory unit. When LSTM calculates the state of each unit, it passes the state through multiple gate structures, ensuring that errors from the last moment are propagated forward and weight modifications from the previous moment are within a reasonable range. This way, issues like gradient vanishing are avoided. The classic LSTM update calculation formula is as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{1}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3}$$

$$\widetilde{c_t} = \mathrm{Tanh}(W_c x_t + U_c h_{t-1}) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot (\widetilde{c_t}) \tag{5}$$

$$h_t = o_t \odot \mathrm{Tanh}(c_t) \tag{6}$$

Among them, i_t is obtained by performing a linear transformation on the input x_t and the previous layer's hidden state h_{t-1}, followed by passing it through the activation function σ. The input gate i_t is a vector that controls the amount of information flowing through each dimension of the node. The parameters W_i, U_i, and b_i are learned and adjusted during training. The forget gate and the output gate follow a similar process. This network structure collectively controls the influence of the previous memory unit on the current state through three components: the input gate, the forget gate, and the activation function.

Attention mechanism

The attention mechanism is used to retain the converted intermediate vector of the input sequence and train a model that can selectively learn from the input text, ultimately producing the final model result [17]. The attention mechanism enhances the learning ability in sequence data by weighting and transforming the sequence. In simple terms, the attention mechanism acts as a mechanism for assisting judgment, helping the model ignore irrelevant information and make more accurate decisions [18, 19].

Attention can be categorized into two types based on the direction of focus: conscious focused attention from top to bottom and unconscious attention from bottom to top, known as saliency-based attention. Saliency-based attention is primarily driven by external stimuli and does not require active intervention, while focused attention is based on specific computational tasks or restricted conditions. The attention mechanism primarily addresses the problem of information overload, effectively allocating resources more reasonably [20–22].

The traditional attention mechanism model is a dot product model, where weights are calculated through the query matrix and the key matrix, and the weight values are transformed into a weight distribution using an activation function. The weighted matrix is obtained by applying this weight distribution to the value matrix [23].

In practical applications, when performing a set of queries, the attention function is calculated simultaneously, with the values packed into a matrix Q, and the keys and values packed into K and V, respectively. The calculation formula for the dot product attention mechanism is as follows [17]:

$$Z = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{7}$$

The scale factor is $\sqrt{d_k}$.

The model of detection

This chapter focuses on the principles underlying model optimization and the characteristics of the optimized model. The complete detection process involves the following steps: First, a large amount of raw data is collected locally through web crawling and log collection and parsing. Next, data processing is performed on the original dataset to obtain a standardized dataset. This standardized dataset is then input into the detection module. Finally, the detection module is trained and used for judgment.

The structure of the model

Based on the current research status, most deep learning studies on XSS are based on traditional network models. To better capture the code's timing, existing research methods utilize LSTM or bidirectional LSTM to model XSS attacks [24]. However, these methods may extract features that contain a significant amount of irrelevant information, increasing the computational complexity of the model and affecting the learning of important features. Additionally, for complex and obfuscated code, a complex process is required for deobfuscation.

To address these issues, this paper proposes a new cross-site script detection model called the Character-level Bidirectional LSTM with Multi-Head Attention (CMABLSTM) model. This model combines the advantages of bidirectional LSTM and the multi-head attention mechanism. The network model aims to improve detection effectiveness through a series of optimizations of the traditional bidirectional LSTM network. The specific features of the model are as follows:

To enhance the understanding of XSS attacks, the CMABLSTM model dissects XSS attacks at the character level. This approach circumvents the limitations of word knowledge and mitigates the impact of incorrect semantic comprehension resulting from improper word segmentation. Consequently, it effectively addresses the issue of poor detection caused by malicious obfuscation of sample data. Character-level representation can be described as the progression from a single character to word representation and then to the entire sentence. This process treats each word individually and maps it to character embedding.

The bidirectional long short-term memory (LSTM) network, building upon the classic LSTM, incorporates the processing of both past and future dependencies. This allows for a more comprehensive understanding of relevant information within the XSS payload and facilitates the extraction of comprehensive abstract features. When the feature matrix is fed into the bidirectional LSTM network model, the feature sequence undergoes multi-head attention to emphasize important features while diminishing irrelevant ones. The final feature vector is then processed by a Softmax classifier to achieve the desired classification effect.

As depicted in Fig. 1, the model structure comprises the following seven main components:

(1) Input layer

The input layer receives a string of character sequences extracted from the traffic flow, preserving the text content's inherent characteristics.

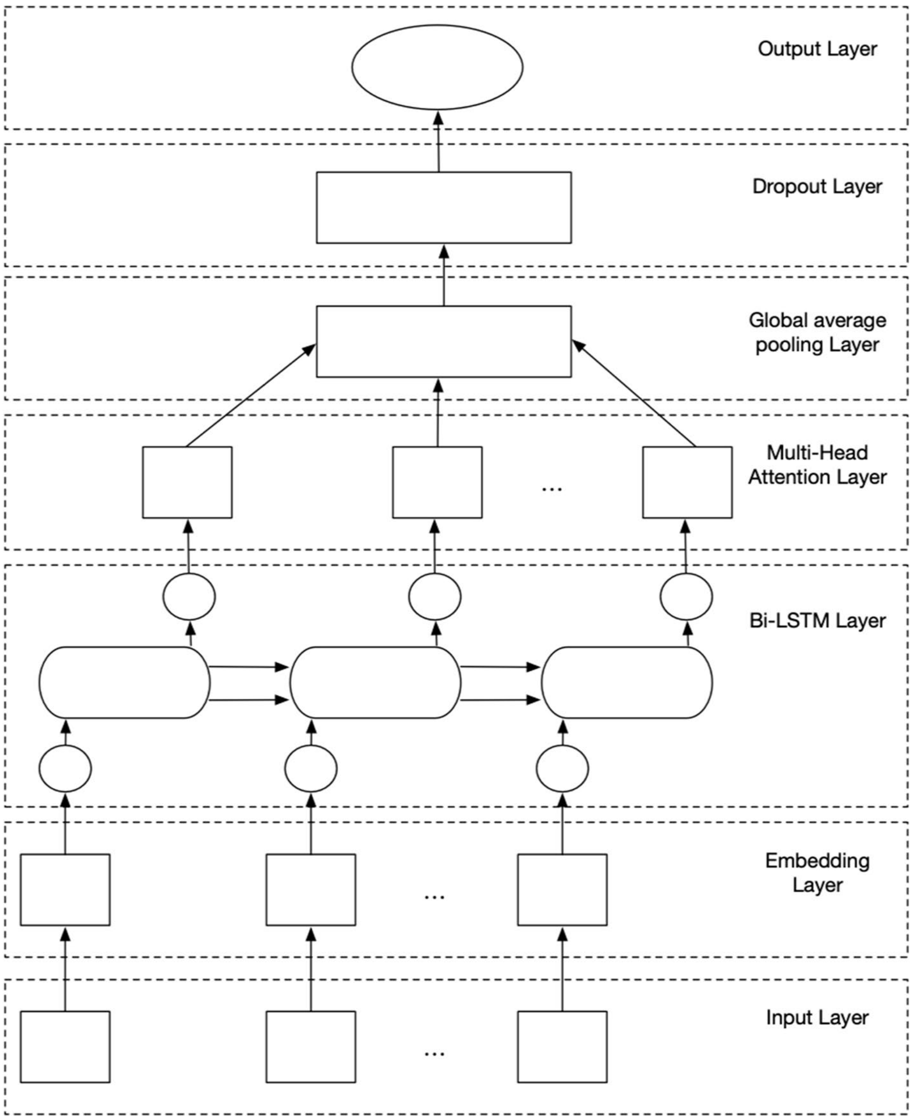

Fig. 1 CMABLSTM model structure diagram

(2) Embedding layer

The input layer parses the data flow into a feature vector, which serves as the input for the layer. The code, after undergoing word segmentation, is converted into a vector. Since the neural network requires a fixed input length, handling the data dimension properly is crucial. Currently, the approach is to truncate sequences that exceed the data dimension and pad sequences that are too short with -1, ensuring that all data has the same length.

(3) Bi-LSTM layer

In this layer, a bidirectional long short-term memory (LSTM) network is employed for deep feature learning. This network effectively captures the temporal aspects of the data and addresses the issues of gradient explosion and vanishing gradients during long sequence training [25]. By utilizing this network in this layer, the learning capability for complex XSS payloads is enhanced. The processing in this layer can be briefly described as follows:

For a given traffic flow T, the word vector $V = \{v_1, v_2, \ldots, v_j\}$ represents the processed input of the word embedding layer. Initially, the forward Long-Short Term Memory (LSTM) algorithm reads the elements of the sequence V from left to right (from v_1 to v_j) and passes through the hidden layer to obtain the positive hidden layer state vector $H = (h_1, h_2, \ldots, h_j)$. Conversely, the LSTM algorithm in the reverse direction reads the sequence V from right to left and obtains the reverse hidden layer state vector $F = (f_1, f_2, \ldots, f_j)$. The hidden layer state $B = [H, F]$ of the network layer is then obtained by combining the forward hidden layer state H and the reverse hidden layer state F. As a result, the bidirectional LSTM enables the output at the current moment to be influenced not only by the preceding state but also by the future state, while also considering deep semantic features.

(4) Multi-head attention layer

The single-head attention mechanism enhances the ability to learn important information within a sentence, thereby capturing long-distance dependencies between words [26]. On the other hand, multi-head attention not only strengthens the capacity to learn such dependencies but also improves understanding of the syntax and semantic structure of sentences.

The multi-head attention mechanism enables the model to focus on different subsets of data and calculate attention values from multiple dimensions. This improves the model's ability to learn feature information from various locations. The output of the upper network is divided into n parts, or heads, and each part is multiplied by a weight matrix W^i to form input vectors W^iX, which are then denoted as Q^i, K^i, and V^i. The attention weight matrix is calculated using the formula shown in (3-1).

$$z^i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \tag{8}$$

The individual results z^i are combined into Z_i, and then the n heads Z_i are concatenated to form Z^C, i.e., $Z^C = \text{concat}(Z_1, \ldots, Z_n)$, which is then multiplied by the weight matrix W_o to obtain the final output $Z = Z^C W_o$.

(5) Global average pooling layer

Global average pooling, introduced by Lin et al. [27], is a technique that effectively replaces fully connected layers. This module computes the average of all feature matrices and uses softmax for classification. The fully connected layer tends to overfit, which affects the generalization of the entire network. Global average pooling combines the effects of the convolutional layer and the fully connected layer by calculating the average value of the feature matrix and associating it with the classification result.

(6) Dropout layer

The generalization layer is an effective technology that prevents neurons from mutually adapting [28]. After pooling the weighted attention vector using global average pooling, a regularization method is applied to improve the network's generalization ability and reduce the risk of overfitting.

(7) Output layer

The output of the network layer, which is the feature vector, is then fed into the Softmax function for classification. The final classification result is obtained by calculating the classification probability p using the following formula:

$$p = \text{softmax}(w_s Z + b_s) \tag{9}$$

Among them, b_s and w_s are the bias and weight of the function.

$$y = \text{argmax}(p) \tag{10}$$

Among them, y is the label predicted by the model, $p \in (0, 1)$.

Algorithm design

The algorithm design in this section focuses on describing the specific operation process of the model. The data collection module collects the original dataset from the data source. After processing the original dataset, the training set (TrS) and test set (TeS) required by the network input layer are obtained. The parameter k in line 7 of the algorithm means that the number of Multi-Head Attention layer's heads are k. The detection model algorithm of CMABLSTM is designed as follows:

```
Input: training set $TrS$, test set $TeS$, $epochs$ and the
       parameters of each network layer $P$.
Output: The result $Y$ is obtained by the Softmax classifier.
1  $i \leftarrow 0;\ j \leftarrow 0;\ TeSize \leftarrow TeS.length();$ while $i < epochs$
   do
2      $output1 \leftarrow Input(TrS);$
3      $output2 \leftarrow BiLSTM(output1);$
4      $output3 \leftarrow Attention((output2, output2, output2), k);$
5      $output4 \leftarrow GAP(output3);$//GAP is the Global
         Average Pooling layer
6      $Y \leftarrow Softmax(output4);$
7      $P \leftarrow Update(P);$
8      $i++;$
9  end
10 while $j < TeSize$ do
11     $Y \leftarrow CMABLSTM(TeS[j]);$
12     $j++;$
13 end
```

Algorithm 1 Detection model algorithm of CMABLSTM.

Experimental results and analysis

Based on the existing experimental environment and data, and following the principles and algorithm steps described in Chapter 2, several comparative experiments have been conducted. This chapter analyzes and compares the experimental results to demonstrate the effectiveness of the CMABLSTM-based XSS classification detection method proposed in this paper.

Experimental environment

This experiment was conducted using the Ubuntu Server 16.04 LTS operating system, with an i7 quad-core CPU, 16GB of memory, and a GTX1080TI graphics card. Python 3.7 was used as the programming language, and the deep learning algorithm library utilized the GPU version of TensorFlow 1.14.

Experimental data

The datasets used for cross-site scripting (XSS) and normal traffic data were primarily collected through crawlers and honeypot logs. The resulting XSS dataset comprised 31,407 normal data samples and 74,063 attack data samples, totaling 105,470 data samples. In the experiment, the malicious and normal samples were combined, and the training set and test set were randomly selected with a ratio of 8:2. The positive samples represented the XSS dataset, labeled as 1, while the negative samples represented the normal dataset, labeled as 0. The data distribution is shown in Table 1.

Experimental evaluation indicators

In the evaluation of experimental results, the selection of evaluation metrics ultimately affects the algorithm's effectiveness in a real environment. This paper analyzed the experimental results from three perspectives: Precision, Recall, and F1 Score. In this experiment, the normal data was considered as the negative sample, and the malicious sample data was considered as the positive sample. The aforementioned three metrics were used to assess the model's performance from a data statistics perspective.

Precision represents the actual ratio of samples correctly classified as positive by the system, and the formula is shown in (11).

$$p = \frac{TP}{TP + FP} \tag{11}$$

Recall represents the proportion of positive samples correctly identified by the system, indicating the extent of positive sample coverage. The formula is shown in (12).

Table 1 Dataset distribution table

Label	Category	Training set	Test set	Total
1	XSS	59250	14813	74063
0	Normal	25126	6281	31407

For TP, FP and FN in formulas (11) and (12), respectively, expressed as:

True Positive (TP): The number of positive samples that are correctly predicted as positive.

$$R = \frac{TP}{TP + FN} \tag{12}$$

F1 score is a comprehensive evaluation index, which is the harmonic average of precision rate and recall rate. The formula is shown in (13).

False Positive (FP): The number of negative sample data that is incorrectly predicted as positive.

False Negative (FN): The number of positive samples that are incorrectly predicted as negative.

$$F1 = \frac{2PR}{P + R} \tag{13}$$

Experimental data processing

In the data processing module, the main task is to standardize the original data and obtain data with the required standard input format for the network model. The specific steps are as follows:

(1) Clean the collected data

Firstly, the invalid data in the dataset is filtered out. Then, the remaining data is truncated and concatenated, which involves removing the domain name information from the URL and concatenating the URL and POST BODY. Next, the data is denoised, primarily focusing on removing duplicate and incomplete data. This process yields valid raw data. An example of simple XSS data is as follows:

%3Djavascript%3Aalert%28/412/%29

(2) Perform word segmentation processing on the cleaned data

The data in the existing dataset is segmented at the character level to preserve data characteristics as much as possible, thereby improving the recognition of obfuscated codes and facilitating feature representation. The sample segmentation results are shown below:

Original data : clickTAG%3D j a v a s c r i p t%3Aalert %28/412/%29

The result after segmentation : ’ c ’ , ’ l ’ , ’ i ’ , ’ c ’ , ’k ’ , ’T ’ , ’A’ , ’G’ , ’% ’ , ’ 3 ’ , ’D’ , ’ j ’ , ’ a ’ , ’v’ , ’ a ’ , ’ s ’ , ’ c ’ , ’ r ’ , ’ i ’ , ’p ’ , ’ t ’ , ’% ’ , ’ 3 ’ , ’A’ , ’ a ’ , ’ l ’ , ’ e ’ , ’ r ’ , ’ t ’ , ’% ’ , ’ 2 ’ , ’ 8 ’ , ’/ ’ , ’ 4 ’ , ’ 1 ’ , ’ 2 ’ , ’/ ’ , ’% ’ , ’ 2 ’ , ’9 ’

(3) Vectorize the segmented data

Each character is converted into numerical data and then transformed into vectors. In this process, consecutive characters are treated as a sequence and passed through a bidirectional long short-term memory (LSTM) network to obtain a vector that connects each character. Since the code length is not fixed, the input dimensions of the network model need to be uniform. Therefore, when converting the code into numerical data, vectors exceeding the dimension are truncated, and vectors with insufficient length are filled with -1 to ensure that all vectors have the same length.

Experimental result

The purpose of this chapter is to analyze the detection results of the CMABLSTM model on the existing test set. In this experiment, the model's performance was evaluated using a vector dimension of 128, the Adam optimizer, a positive sample test set size of 14,813, and a negative sample test set size of 6,281. Throughout the experiment, leveraging theoretical knowledge related to XSS and deep learning, the network model was designed and continuously tuned to obtain the final model structure.

CMABLSTM is a network model that combines the Bi-LSTM model with a multi-head attention mechanism at the character level. In this experiment, the Adam optimization method was used with the two exponential decay rates set to (beta1=0.9, beta2=0.98), and the default learning rate set to 1.0. The activation function applied the Softmax function for model classification. To prevent overfitting, a regularization layer was connected after each network model to improve the model's generalization ability. Additionally, to better capture features, global average pooling technology was employed to replace the fully connected layer. This involved using average pooling to condense the multi-dimensional feature vectors from the last layer into single points, which were then combined to form the final feature vector. Finally, a Softmax calculation was performed for classification judgment.

(1) Multiple-head attention mechanism (MHAM) layer evaluation experiment

It was observed during the experiment that the MHAM layer significantly influenced the detection results when the number of heads was set either too large or too small. In this chapter, the default number of internal hidden nodes was set to 128, and the performance of the MHAM network structure with different numbers of heads (which must be divisible by the vector dimension) was compared. The experiment settings included [1, 2, 4, 8, 16, 32]. The results are presented in Table 2. From Table 2, it was evident that the model performed the best when the number of heads was set to 4. As the number of heads increased beyond 4, the performance of the network model gradually decreased. Consequently, the number of heads in this layer of the network model was set to 4.

(2) Vector dimension comparison experiment

The experiment revealed that the vector dimension of the network model's embedding layer had a direct impact on the detection results when set too large or small. This chapter involves experiments carried out to compare vector dimensions [64, 128, 256] under default network structures. Figure 2 displays the experimental results.

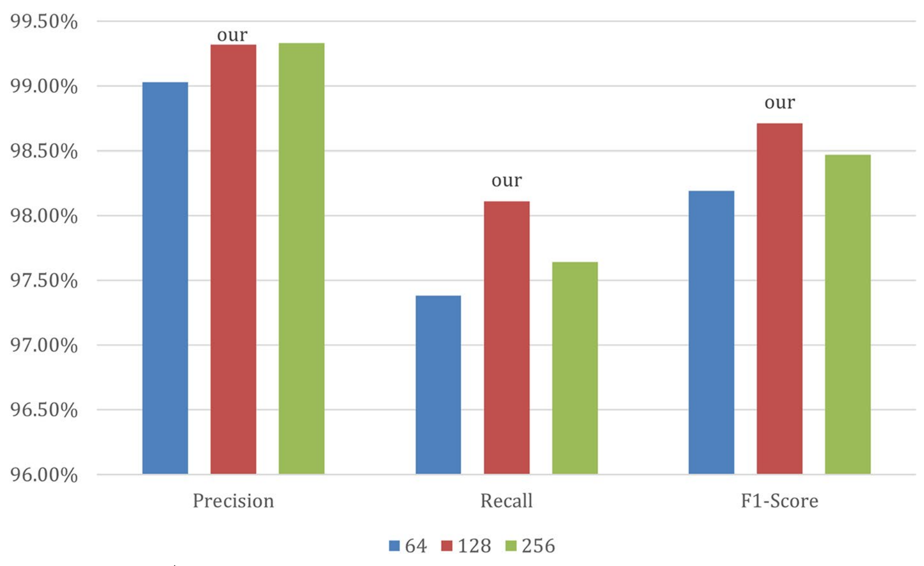

Fig. 2 Experiments comparing vector dimensions

The most optimal experimental results occurred when the vector dimension was set to 128.

(3) Evaluation experiment of global average pooling layer

Due to the complexity of the network model in this chapter, in order to prevent severe overfitting, a global average pooling layer was added to the model for experiments. This chapter carried out comparative experiments to evaluate the importance of adding a global average pooling layer to the model, and the experimental results are shown in Fig. 3.

In actual datasets, there are often imbalanced distributions of positive and negative samples, where the ratio of positive and negative samples can vary in different test sets. The ROC curve remains unchanged even when the dataset distribution changes. To better demonstrate the effectiveness of model training, this section presents the experimental results as ROC curves. Figure 4 shows the ROC curve graphs (the ROC curve of the test dataset is enlarged in the upper left corner for clearer visualization). From the ROC curve graph, we can observe that the area under the curve is close to 1, indicating good prediction performance with a high accuracy rate.

Comparative experimental methods and evaluation

To validate the effectiveness and advantages of the CMA-BLSTM model, this article conducted several sets of comparative experiments, comparing the results with traditional machine learning algorithms and deep learning algorithms.

(1) Traditional machine learning

This chapter compares various deep learning model methods, including LSTM, Bi-LSTM, the joint network structure of Bi-LSTM and Attention (ABLSTM), the joint network structure of Bi-LSTM and Multi-Head Attention (MABLSTM), and the network structure proposed in this article. Both character-level feature extraction and word-2vec feature extraction methods were used in the experiment. The experimental results are presented in Table 3, demonstrating the detection performance of each model based on three indicators.

(2) Deep learning model

This chapter compares the deep learning model methods. The models involved are LSTM, Bi-LSTM, the joint network structure of Bi-LSTM and Attention (ABLSTM), the joint network structure of Bi-LSTM and Multi-Head Attention (MABLSTM), and The network structure in this article. The character-level feature extraction method and word2vec feature extraction method were used in the experiment. The experimental

Table 2 Dataset distribution table

Number of heads	Precision	Recall	F1-score
1	98.99%	97.81%	98.39%
2	99.17%	98.02%	98.59%
4	99.32%	98.11%	98.71%
8	99.33%	98.03%	98.67%
16	99.08%	97.94%	98.50%
32	98.72%	97.38%	98.04%

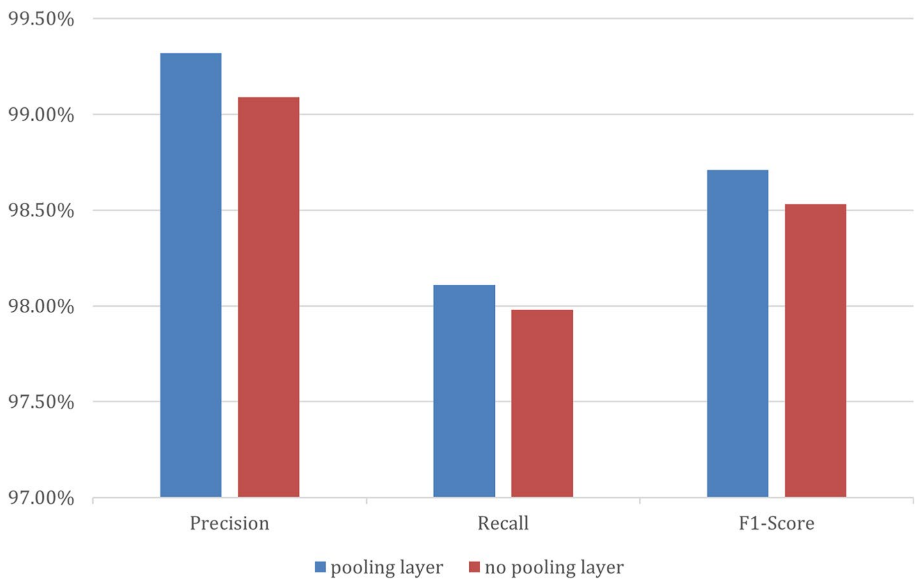

Fig. 3 Experiment evaluating global pooling layers

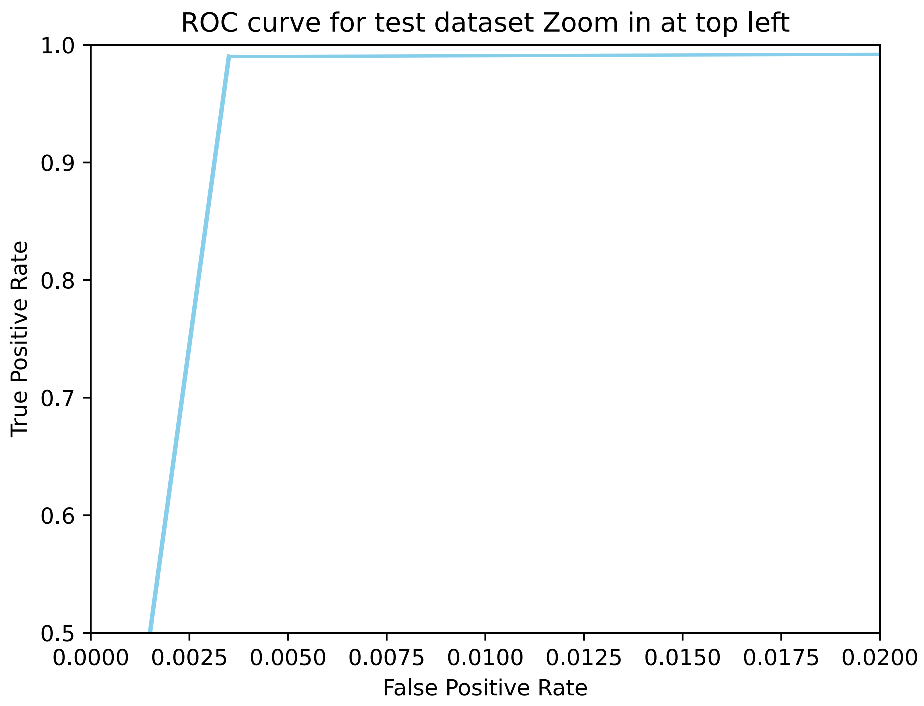

Fig. 4 The ROC curve of the test data set is enlarged in the upper left corner

Table 3 Demonstration of ental results

Model name	Precision	Recall	F1-score
CMABLSTM	99.32%	98.11%	98.71%
XGBoost	94.92%	93.40%	94.15%
SVM	94.33%	90.26%	92.30%

results are shown in Table 4, showing the detection effect of each model from three indicators.

Visualizing the data in Table 4, Fig. 5 clearly shows that the accuracy, recall rate, and F1 score of the proposed method are higher than those of the other models (MABLSTM, ABLSTM, LSTM, BLSTM). Thus, the method proposed in this article outperforms other deep learning models in terms of evaluation results.

Comparing the aforementioned models, this method achieves an F1 score of 99.48%, surpassing the evaluation results of the other models. In summary, the advantage of the model presented in this paper lies in its utilization of bidirectional LSTM to effectively capture long-term dependencies in text information. Additionally, the adoption of a character vector processing mechanism facilitates the retention of content related to confusing information. Lastly, through the employment of the multihead attention mechanism, the model can effectively focus on syntactic features from multiple dimensions. The experiments demonstrate that this method significantly enhances the detection capability of XSS attacks.

Table 4 Demonstration of experimental results

Model name	Precision	Recall	F1-score
CMABLSTM	99.32%	98.11%	98.71%
MABLSTM	99.02%	97.99%	98.50%
ABLSTM	98.90%	97.03%	97.95%
LSTM	97.31%	96.08%	96.69%
BLSTM	98.36%	96.21%	97.27%

Related work

XSS attack detection methods

Gulit Habibi et al. [29] employed SVM with n-gram features for the detection of XSS attacks, effectively improving the accuracy of traditional machine learning models in XSS attack detection tasks. However, this study still fails to address the limitation of traditional machine learning models in handling evolving XSS scripts over time. Additionally, it requires considerable preprocessing work. Jitendra Kumar et al. [30] utilized Convolutional Neural Network (CNN) as a viable approach for detecting XSS attacks. Their research has shown promising results in detecting XSS attacks in web applications. However, due to inherent limitations of CNN, this approach faces challenges in capturing long-term dependencies and sequential patterns in XSS scripts. Fang et al. [31] developed an XSS detection model based on long short-term memory networks

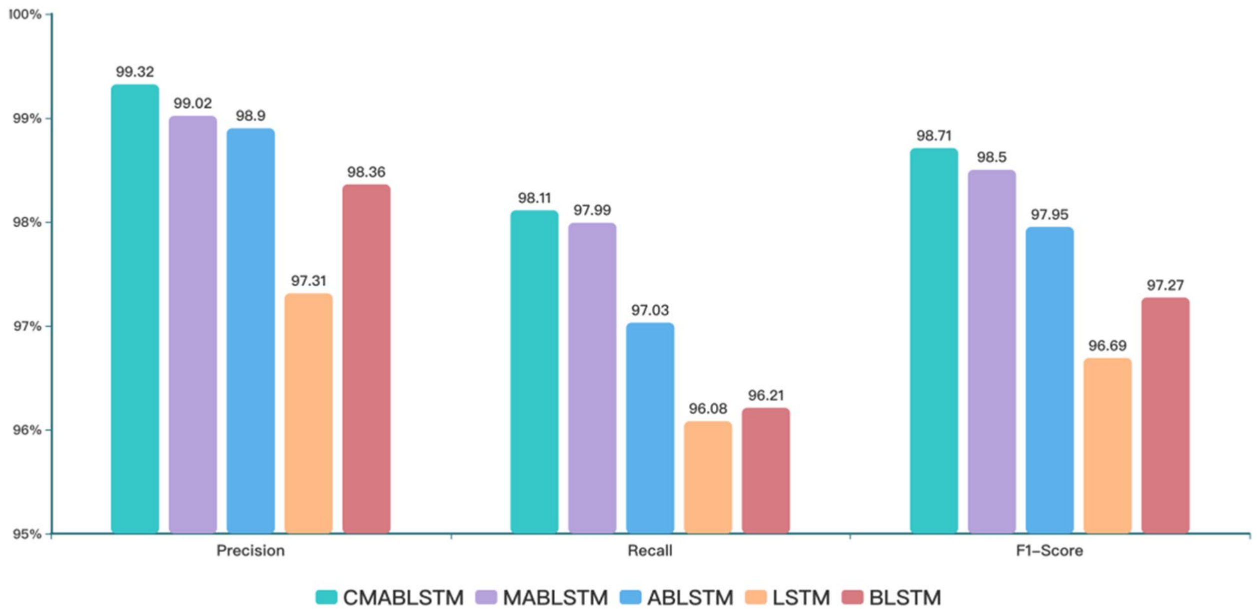

Fig. 5 Comparative experiment cart of deep learning model

(LSTMs) applied to text sequences. Their experimental results demonstrated that leveraging LSTM can effectively improve the effectiveness of XSS detection. However, the analysis revealed that the model was prone to significant confusion in XSS detection, leading to less-than-ideal results.

In summary, while previous studies have made some progress in XSS attack detection, they still have limitations. This study extends the existing work by utilizing a character-level bidirectional long short-term memory network model with multiple attention mechanisms to address the limitations of traditional machine learning models in XSS attack detection. It also optimizes the extraction of semantic features and the detection of obfuscation techniques, resulting in improved accuracy and efficiency in XSS attack detection.

Advanced applications of LSTM

To provide advanced applications of LSTM technology in related fields, we highlight three notable examples. Qi [32] proposes a recommendation model called PPCM for location-based social networks. It addresses the challenge of acquiring preferences from sparse users by considering group influence and privacy protection using LSH. The model captures long- and short-term dependencies using LSTM with an attention mechanism and temporal sliding window. By leveraging POI categories, it efficiently mines user interests despite limited data. Evaluation on real check-in datasets from New York City and Tokyo demonstrates improved recommendation performance compared to other models. Liu [33] focuses on the collection of user check-in data in location-based social networks and proposes an improved model called ITGCN for successive point-of-interest (POI) recommendation. ITGCN captures dynamic user and POI representations while considering high-order connectivity through a self-attention aggregator. The experimental results show that ITGCN outperforms existing methods, providing valuable insights for travel enterprises' management systems and future planning. Liu [34] addresses the challenge of controlling the greenhouse climate and proposes a greenhouse climate prediction model. The model, called GCP-lstm, focuses on six climatic factors that affect crop growth and uses long short-term memory (LSTM) to capture the nonlinear climate changes. A 5-minute time sliding window is employed to consider the short-term impact on future trends. The model also exhibits robustness in handling abnormal data from sensors. The method is evaluated on datasets of three vegetables, tomato, cucumber, and pepper, showing better performance compared to other models.

Drawing inspiration from these advanced applications of LSTM, our work on XSS attack detection utilizing a LSTM model with multiple attention mechanisms extends the capabilities of traditional machine learning models. By incorporating LSTM's ability to capture long-term dependencies, our model aims to improve accuracy and efficiency in detecting XSS attacks. We demonstrate how leveraging LSTM and attention mechanisms can optimize the extraction of semantic features and detection of obfuscation techniques, thereby enhancing the performance of XSS attack detection.

Conclusion

This paper presents an enhanced network model structure for detecting XSS malicious attacks, with the objective of bolstering the protection of Cloud Computing against such attacks, which can lead to significant confusion and involve intricate logic. The salient features of this network structure are as follows: it initiates with character-level features to effectively preserve text details and prevent the compromise of the semantic structure due to improper text segmentation. Moreover, the structure employs bidirectional LSTM to capture the temporal context of the text, ensuring that the current features are associated not only with the preceding text but also with the subsequent text. Additionally, the multi-head attention mechanism is employed to leverage different feature subspaces, enabling the model to extract more distinctive characteristics and enhance its comprehension of the textual context. Through experimental analysis conducted in this study, it has been demonstrated that the proposed CMABLSTM model surpasses other similar detection techniques in terms of XSS detection, achieving an impressive F1 score of 98.71%. This innovative approach to XSS detection plays a pivotal role in fortifying Cloud Computing against XSS attacks. By implementing this model, Cloud Computing systems can enhance their ability to safeguard themselves and mitigate the risks posed by XSS attacks, thereby ensuring the security and integrity of the overall Cloud Computing infrastructure. However, this manuscript only uses the CMABLSTM model to detect XSS vulnerability attacks. In the future, our research will focus on investigating the suitability of this model for diverse web vulnerability detection and vulnerability mining tasks within the Cloud Computing environment. Specifically, we will explore its applicability in areas such as buffer overflow, SQL injection, and cross-site request forgery.

Acknowledgements
We sincerely thank the editors and reviewers for their valuable comments on this paper.

Authors' contributions
All authors were involved in the research for this paper. Xiaolong Li and Tina Wang led the entire work. Wei Zhang carried out the detailed research, including background study and review compilation. Niu Hui and Ting-Yu Zhang were responsible for the modeling and experimental analysis part of the whole paper. Zhao Fujitou and Wang Yongji were responsible for writing the thesis as well as the experimental evaluation. Yufei Wang was responsible for the correction and layout of the thesis. All authors read and approved the final draft.

Funding
Not applicable.

Availability of data and materials
All code used to support this work is available from the authors upon request.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 29 May 2023 Accepted: 5 July 2023
Published online: 08 August 2023

References

1. Center, N.I.E. China Internet Cyber Security Report. https://www.cert.org.cn/publish/main/upload/File/2018annual.pdf. 2021-10-03
2. OWASP. OWASP Top Ten. https://owasp.org/www-project-top-ten/. 2021-10-29
3. Bhardwaj A, Chandok SS, Bagnawar A, Mishra S, Uplaonkar D (2022) Detection of cyber attacks: XSS, sqli, phishing attacks and detecting intrusion using machine learning algorithms. 2022 IEEE Global Conference on Computing. Power and Communication Technologies (GlobConPT), IEEE, pp 1–6
4. Perumal S, et al (2021) Stacking ensemble-based XSS attack detection strategy using classification algorithms. In: 2021 6th International Conference on Communication and Electronics Systems (ICCES), IEEE, pp 897–901
5. Habibi G, Surantha N (2020) XSS attack detection with machine learning and n-gram methods. In: 2020 International Conference on Information Management and Technology (ICIMTech), IEEE, pp 516–520
6. Luo J, Xu G (2021) XSS attack detection methods based on xlnet and gru. 2021 4th International Conference on Robotics. Control and Automation Engineering (RCAE), IEEE, pp 171–175
7. Lei L, Chen M, He C, Li D (2020) XSS detection technology based on LSTM-attention. 2020 5th International Conference on Control. Robotics and Cybernetics (CRC), IEEE, pp 175–180
8. Jingyu Z, Hongchao H, Shumin H, Huanruo L (2021) A XSS attack detection method based on subsequence matching algorithm. In: 2021 IEEE International Conference on Artificial Intelligence and Industrial Design (AIID), IEEE, pp 83–86
9. Hadpawat T, Vaya D (2017) Analysis of prevention of XSS attacks at client side. Int J Comput Appl 173(10):1–4
10. Santithanmanan K (2022) The detection method for XSS attacks on nfv by using machine learning models. In: 2022 International Conference on Decision Aid Sciences and Applications (DASA), IEEE, pp 620–623
11. Chui KT, Gupta AK (2022) Analysis of machine learning based XSS attack detection techniques. Cyber Secur Insights Mag Insights2Techinfo 1:7–10
12. Birje MN, Challagidad PS, Goudar R, Tapale MT (2017) Cloud computing review: concepts, technology, challenges and security. Int J Cloud Comput 6(1):32–57
13. Yu Y, Si X, Hu C, Zhang J (2019) A review of recurrent neural networks: LSTM cells and network architectures. Neural Comput 31(7):1235–1270
14. Landi F, Baraldi L, Cornia M, Cucchiara R (2021) Working memory connections for LSTM. Neural Netw 144:334–341
15. Liu Z, Zhou W, Li H (2019) Ab-LSTM: Attention-based bidirectional LSTM model for scene text detection. ACM Trans Multimed Comput Commun Appl (TOMM) 15(4):1–23
16. Chen T, Wang Z, Li G, Lin L (2018) Recurrent attentional reinforcement learning for multi-label image recognition. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32. AAAI Press
17. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. Advances in neural information processing systems 30:5998–6008
18. Hubballi N, Singh Y, Garg D (2023) XSSmitigate: Deep packet inspection based XSS attack quarantine in software defined networks. In: 2023 IEEE International Conference on Consumer Electronics (ICCE), IEEE, pp 1–6
19. Brauwers G, Frasincar F (2023) A general survey on attention mechanisms in deep learning. IEEE Transactions on Knowledge and Data Engineering 35:3279–98

20. Wang J, Liu L (2020) A multi-attention deep neural network model base on embedding and matrix factorization for recommendation. Int J Cogn Comput Eng 1:70–77
21. Zhao X, Sun K, Gong S, Wu X (2023) Rf-biLSTM neural network incorporating attention mechanism for online ride-hailing demand forecasting. Symmetry 15(3):670
22. Ye J, Wang H, Li M, Wang N (2021) Iot-based wearable sensors and bidirectional lstm network for action recognition of aerobics athletes. Journal of Healthcare Engineering 2021(Article ID 9601420)
23. Augustyniak Ł, Kajdanowicz T, Kazienko P (2019) Aspect detection using word and char embeddings with (bi) LSTM and crf. In: 2019 IEEE second international conference on artificial intelligence and knowledge engineering (AIKE), IEEE, pp 43–50
24. Qiqin C, Liang W (2020) Application research of biLSTM in cross-site scripting detection. J Front Comput Sci Technol 14(8):1338
25. Sunny MAI, Maswood MMS, Alharbi AG (2020) Deep learning-based stock price prediction using lstm and bi-directional LSTM model. In: 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), IEEE, pp 87–92
26. Mnih V, Heess N, Graves A, et al (2014) Recurrent models of visual attention. Advances in neural information processing systems 27:2204–2212
27. Lin M, Chen Q, Yan S (2013) Network in network. arXiv preprint arXiv:1312.4400
28. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580
29. Habibi G, Surantha N (2020) XSS attack detection with machine learning and n-gram methods. In: 2020 International Conference on Information Management and Technology (ICIMTech), pp 516–520. https://doi.org/10.1109/ICIMTech50083.2020.9210946
30. Kumar J, Santhanavijayan A, Rajendran B (2022) Cross site scripting attacks classification using convolutional neural network. In: 2022 International Conference on Computer Communication and Informatics (ICCCI), pp 1–6. https://doi.org/10.1109/ICCCI54379.2022.9740836
31. Fang Y, Li Y, Liu L, Huang C (2018) Deepxss: Cross site scripting detection based on deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence. Springer, pp. 47–51
32. Qi L, Liu Y, Zhang Y, Xu X, Bilal M, Song H (2022) Privacy-aware point-of-interest category recommendation in internet of things. IEEE Internet Things J 9(21):21,398–21,408. https://doi.org/10.1109/JIOT.2022.3181136
33. Liu Y, Wu H, Rezaee K, Khosravi MR, Khalaf OI, Khan AA, Ramesh D, Qi L (2023) Interaction-enhanced and time-aware graph convolutional network for successive point-of-interest recommendation in traveling enterprises. IEEE Trans Ind Inform 19(1):635–643. https://doi.org/10.1109/TII.2022.3200067
34. Liu Y, Li D, Wan S, Wang F, Dou W, Xu X, Li S, Ma R, Qi L (2022) A long short-term memory-based model for greenhouse climate prediction. Int J Intell Syst 37(1):135–151

Publisher's Note