Cross-site Scripting Detection with Two-channel Feature Fusion Embedded in Self-Attention Mechanism

Tianle Hu , Chonghai Xu , Shenwen Zhang, Shuangshuang Tao , Luqun Li

PII: S0167-4048(22)00382-0
DOI: https://doi.org/10.1016/j.cose.2022.102990
Reference: COSE 102990

To appear in: *Computers & Security*

Received date: 8 June 2022
Revised date: 16 August 2022
Accepted date: 23 October 2022

Cross-site Scripting Detection with Two-channel Feature Fusion Embedded in Self-Attention Mechanism

Tianle Hu, Chonghai Xu, Shenwen Zhang, Shuangshuang Tao and Luqun Li*

The College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, PR China

ARTICLE INFO

Keywords:
Cross-site scripting
Word2Vec
Feature fusion
Bidirectional Long-Short Term Memory
Self-Attention mechanism

ABSTRACT

In the era of big data, stealing users' private data has become one of the main targets of network hackers. In recent years, cross-site scripting (XSS) attacks to obtain users' privacy data have been one of the main web attack methods of network hackers. Traditional antivirus software cannot identify such cross-site scripting attacks. To identify cross-site scripting attacks quickly and accurately, we proposed a cross-site scripting detection model (C-BLA) with two-channel multi-scale feature fusion embedded in a self-attention mechanism. The model first maps cross-site scripting payloads into spatial vectors by data preprocessing using Word2Vec. Then the two-channel network performs feature extraction on the data. Channel I: extract local features of cross-site scripting payloads at different scales by designing parallel one-dimensional convolutional layers with different convolutional kernel sizes; Channel II: extract semantic information of cross-site scripting payloads from two directions of positive and negative order using a bidirectional Long-Short Term Memory network, and then embed the self-attention mechanism to strengthen the semantic information features. Experiments show that the proposed model achieves a precision rate of 99.8% and a recall rate of 99.1% for cross-site scripting detection, which is a certain improvement in detection rate compared with a single deep learning model and traditional machine learning methods. The two-channel feature fusion of this model better solves the cross-site scripting detection problem.

1. Introduction

Cross-site scripting (XSS) attack is a malicious attack in which an attacker exploits a web application vulnerability and induces the victim to visit a web page injected with a malicious script, resulting in an attack trigger that steals user information. These scripts are generally written in client-side scripting languages such as JavaScript and HTML. In general, XSS is classified into three categories based on its characteristics and utilization practices [1]: reflected, stored, and DOM-based. Among them, reflected and stored XSS are server-side vulnerabilities, while DOM-based XSS belongs to a client-side vulnerability. According to data published by the Open Web Application Security Project (OWASP) [2], XSS attacks respectively ranked third and seventh in global web application security risks in 2013 and 2017. In the latest data published in 2021 [3], XSS is classified as a security risk in the "Injection" category, ranking third. The details of the statistics are shown in Table 1. These data show that XSS attacks are still a serious threat to user security and privacy.

With the development of technology, various variants of cross-site scripting attacks have emerged, and various cross-site scripting attacks are commonplace. Sina Weibo, a famous Chinese social network, was attacked by a cross-site scripting worm, in which the personal information of more than 30,000 Weibo users was stolen in just one hour. Renren, which has more than 40 million registered users, was also exposed to a cross-site scripting worm. Amazon's e-book library, KindleLibrary, also has a cross-site scripting vulnerability that can be hidden in the metadata of Kindle e-books, such as titles, which can be used by hackers to steal cookies from the victim's Amazon account, causing considerable damage to the user.

In summary, cross-site scripting attacks come in a variety of forms and structures and are therefore very easy for hackers to exploit. Attackers usually encode the script code multiple times, such as URL encoding, Base64 encoding, etc. This leads to traditional detection methods not being able to effectively read the maliciously obfuscated source code, which in turn leads to redundant or even invalid features extracted in subsequent stages, and detection efficiency needs to be improved.

Deep learning techniques have developed rapidly in recent years, demonstrating a powerful ability to detect unknown samples. In this paper, we propose a cross-site scripting detection model C-BLA with two-channel feature fusion embedded in the Self-Attention Mechanism. The main contributions of us are as follows:

- For the characteristics of cross-site scripting payloads, the pre-processed data are word-sorted by designing a set of regular expressions. With the help of the Word2Vec tool, suitable dimensions are selected to map the disambiguated payload to a high-dimensional vector space to characterize the relationship between words with word vectors.
- With the feature extraction capability of convolutional neural networks for text sequence data, three parallel one-dimensional convolutional layers with different convolutional kernel sizes are set to extract local features of cross-site scripting payloads at different scales and perform multi-scale feature fusion.
- The semantic information features of cross-site scripting payloads are extracted from both positive and neg-

*Corresponding author
liluqun@gmail.com (L. Li)

Table 1
Top 10 OWASP security vulnerabilities from 2013 to 2021.

Level	2013 OWASP Top 10	2017 OWASP Top 10	2021 OWASP Top 10
A01	Injection	Injection	Broken Access Control
A02	Broken Authentication and Session Management	Broken Authentication	Cryptographic Failures
A03	Cross-Site Scripting (XSS)	Sensitive Data Exposure	Injection (including XSS)
A04	Insecure Direct Object References	XML External Entities (XXE)	Insecure Design
A05	Security Misconfiguration	Broken Access ControlSecurity Misconfiguration	Cryptographic Failures
A06	Sensitive Data Exposure	Security Misconfiguration	Security Misconfiguration
A07	Missing Function Level Access Control	Cross-Site Scripting (XSS)	Identification and Authentication Failures
A08	Cross-Site Request Forgery (CSRF)	Insecure Deserialization	Software and Data Integrity Failures
A09	Using Components with Known Vulnerabilities	Using Components with Known Vulnerabilities	Security Logging and Monitoring Failures
A10	Unvalidated Redirects and Forwards	Insufficient Logging & Monitoring	Server-Side Request Forgery

ative directions with the help of the temporal feature extraction capability of bidirectional Long-Short Term Memory (Bi-LSTM) network for sequence data. In order to solve the long-range semantic dependency problem faced by Bi-LSTM in extracting semantic information of payload sequences, the embedded Self-Attention Mechanism is proposed to strengthen the extracted semantic information features.

- The features extracted from the CNN channel fused with multi-scale and the Bi-LSTM channel embedded with Self-Attention Mechanism combine the advantages of both to achieve better detection results.

2. Related Work

Currently, the detection methods for XSS attacks in network security can be broadly classified into the following categories: static analysis, dynamic analysis, and machine learning methods.

2.1. Detection methods based on static analysis

Wassermann et al. [4] proposed a static detection method for finding direct solutions to weak input validation or lack of input validation, which is better for detecting XSS vulnerabilities in server-side code but cannot detect DOM-based XSS. Gupta et al. [5] proposed CSSXC, an inspection framework deployed in cloud environments, which cleans up XSS vulnerabilities in a context-sensitive manner. Mohammadi et al. [6] proposed a unit test-based approach for detecting XSS vulnerabilities caused by incorrect coding.

Static analysis methods can effectively reduce the rate of leakage by querying all paths in the source code, but they cannot analyze dynamic code and code that has applied decompiling techniques such as code mangling.

2.2. Detection methods based on dynamic analysis

Parameshwaran et al. [7] designed a dynamic taint analysis platform DexterJS for detecting DOM-based XSS vulnerabilities using source-to-source rewrite execution at the character level. Wang et al. [8] proposed a detection framework based on client-side taint tracking, TT-XSS, which can dynamically analyze the source-to-receiver data flow and can effectively detect DOM-based XSS.

Dynamic analysis methods do not require access to the source code of web pages, but require high time overhead and may yield a high false positive rate.

2.3. Detection methods based on machine learning

Kascheev et al. [9] used various machine learning methods such as SVM, decision tree, naive Bayes classifier, and logistic regression for XSS detection, where the decision tree model achieved the best results, obtaining a recall rate of 93.70% and an accuracy rate of 98.81%. Rathore et al. [10] proposed to detect XSS vulnerabilities in social networks based on three features, URLs, HTML tags, and SNSs, using ten machine learning algorithms such as Random Forest, Logistic Regression, and Naive Bayes. Among these models, the Random Forest model achieved an accuracy rate of 97.2% and a recall rate of 97.1%.

Traditional machine learning-based detection methods rely on manually extracted features and have certain limitations.

2.4. Detection methods based on deep learning

In recent years, with the development of deep learning, many researchers in the field of network security have adopted this approach for cross-site scripting detection.

Fang et al. [11] proposed an XSS detection method called DeepXSS, which constructs a neural network consisting of LSTM, Dropout layer, Softmax layer. The experiment achieved 99.5% accuracy and 97.9% recall, which is an improvement over both ADTree and AdaBoost models. Lei et al. [12] proposed an improved model adding an attention mechanism. This model first learns context-sensitive features using LSTM, and then adds an attention mechanism to further extract effective features, which can identify XSS attacks more effectively, but the generalization ability of the model needs to be improved. Liu et al. [13] conducted comparative experiments using deep learning models such as CNN, LSTM, TextRCNN, as well as machine learning model XGBoost for the XSS classification problem. The experimental results showed that TextRCNN achieved the highest accuracy rate of 93.95% while LSTM achieved the lowest loss value of 0.056 as well as a shorter detection time.

By analyzing the current state of research at home and abroad, it is clear that the following gaps exist in current cross-site scripting detection research: cross-site scripting is still an active and highly damaging class of attacks, and existing detection methods cannot effectively target new payloads that are increasingly variable and flexible, and the false positive rate needs to be reduced. Traditional methods rely on manually extracted features, which are highly subjective, while deep learning models can automatically learn abstract features that are highly relevant to the classification of cross-site scripting, as well as have strong learning capabilities for new and unknown samples. Therefore, this paper chooses to use a deep learning model for cross-site scripting detection.

3. Architecture and Components of C-BLA

In this section, we propose a detection model call C-BLA which embeds the Self-Attention Mechanism and fuses features. The model is set up with dual channels for feature extraction. The core of the model is the fusion of local features of the XSS extracted from the 1D convolutional layer and the semantic information features of the XSS enhanced by the Self-Attention Mechanism. As is shown in Figure 1, the model consists of the following components:

(1) Input Layer: input the pre-processed XSS payloads;
(2) Embedding Layer: map XSS payloads to vector space by Word2Vec after word separation; the relationship between words is characterized by word vectors;
(3) Feature extraction channel I: contain ① Conv1D layer with a convolutional kernel size of 2; ② Conv1D layer with a convolutional kernel size of 4; ③ Conv1D layer with a convolutional kernel size of 6, three in parallel;
(4) Feature Fusion Layer for Channel I: fuse XSS local features extracted by three parallel Conv1D layers;
(5) Feature extraction channel II: contain ① Bi-LSTM; ② Self-Attention Mechanism;
(6) Dual-channel Feature Fusion Layer: fuse features extracted from channel I and channel II;
(7) Flatten Layer: one-dimensionalize multi-dimensional input features;
(8) Dropout Layer: prevent overfitting;
(9) Output Layer: consist of a Dense layer with softmax activation function and classify malicious XSS and normal XSS.

Due to the limitation of convolution kernel size, the one-dimensional convolution layer only considers the relationship between the features extracted within the range of convolution kernels when performing the convolution operation and does not fully consider the global information. Self-Attention Mechanism acting on Bi-LSTM can solve the long-range semantic dependency problem while fully extracting contextual information. Therefore, we combine CNN, Bi-LSTM, and Self-Attention Mechanism to extract features from XSS at different levels, aiming to obtain scientifically comprehensive scripting features.

3.1. Multi-scale feature extraction Channel I

In recent years, convolutional neural network (CNN) have shown remarkable performance in areas such as computer vision. At the same time, CNN is also applicable to the field of natural language processing. Among them, one-dimensional convolution (Conv1D) is mostly used to process text sequences, and text features of different scales can be extracted by designing convolutional kernels of different sizes. Convolutional kernels[14] perform convolutional operations on each part of the text sequence data to extract features and adopt weight sharing to effectively reduce the number of updated weights required and thus reduce the computational complexity of the model.

In the embedding layer of the model, the cross-site scripting payloads after preprocessing and word splitting are represented as vectors of dimension d. The maximum length of the payload sequence is set to T. That is to say, the part exceeding the length T is directly truncated and the part less than the length T is padded. Then the complete payload sequence can be transformed into a word embedding matrix with fixed length T.

The one-dimensional convolutional layer performs convolutional operations on the cross-site scripting payload sequences processed by the embedding layer to extract locally relevant features.We note that the number of neurons contained in the convolution layer is n and the payload word embedding matrix is X_i. Then the feature z_i extracted after the i^{th} neuron corresponds to the convolution kernel $K_i \in \mathbb{R}^{t\times d}$ with the word vector matrix for the convolution operation is shown in formula (1).

$$z_i = ReLU\left(X_i \bigotimes K_i + b_i\right) \tag{1}$$

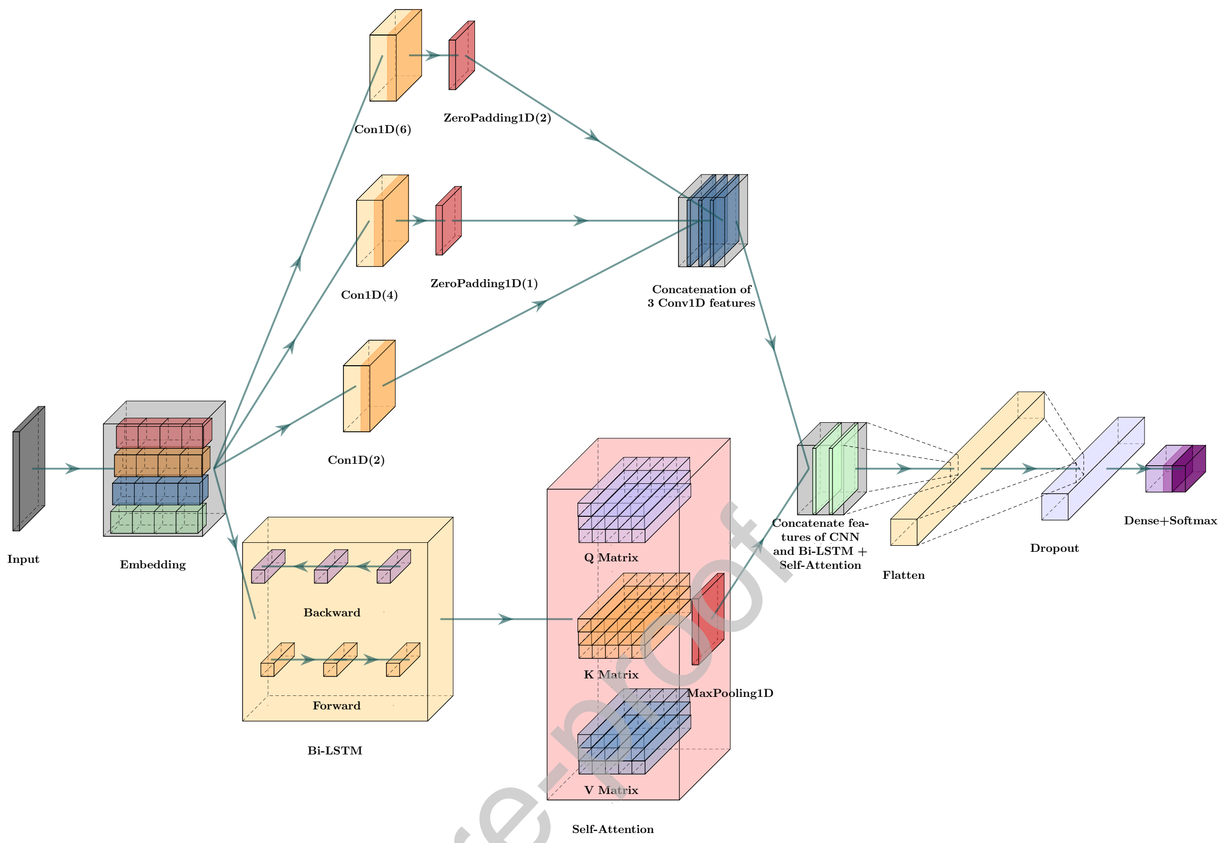
In the formula, $\bigotimes$ is the convolution operator, K_i is the weight matrix, b_i is the bias item, and $ReLU$ denotes the nonlinear activation function. The output C_{out} of the final Conv1D layer is the result of joining the output of n neurons.

The convolutional kernels have different sizes, different receptive fields, and thus resulting in obtaining different local features. Smaller convolutional kernels correspond to smaller perceptual fields, with stronger representation of geometric detail information and weaker representation of semantic information. While larger convolutional kernels correspond to larger perceptual fields, weaker representation of geometric detail information, and stronger representation of semantic information. Therefore, setting up three parallel Conv1D layers with different sizes of convolutional kernels can simultaneously extract local features of cross-site scripting payloads at different scales. Combining the representational power of geometric detail information and the representational power of semantic information and fusing the multi-scale local features helps to improve the classification effect.

3.2. Feature extraction Channel II with embedded Self-Attention Mechanism

3.2.1. Bi-LSTM

The convolutional layer loses much of the previous information as the sliding window moves from the head to the tail when dealing with longer cross-site scripting payloads. Relying only on local features extracted by CNN for



Model.pdf

Fig. 1: Structure of C-BLA.

classification is not sufficient. Recurrent Neural Networks (RNN) receive not only the input of the current moment but also the output of the previous moment at each time step, and have the ability to process the information of the current moment using the past information. However, RNN has obvious shortcomings in dealing with long-range semantic dependency problems, which can easily lead to gradient disappearance or gradient explosion phenomenon due to the long back propagation path.

The Long-Short Term Memory (LSTM) network is improved to address the above problems and solves the long-range semantic dependency problem by setting up memory units. Figure 2 shows the LSTM unit and the architecture of the Bi-LSTM network [15], which mainly consists of an input gate, forget gate, and output gate, and the combination of the three completes the inflow, state update, and outflow of information.

The forget gate in the LSTM cell receives two inputs: the hidden state h_{t-1} of the output of the previous layer and the input X_t of the current time step. We train to obtain a gate function f_t. Formula (2) gives the calculation for this function:

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

The input gate in the LSTM unit receives two inputs: the hidden state h_{t-1} of the output of the previous layer and the input X_t of the current time step. We train to obtain a gate function i_t. At the same time, the integration of information from the previous moment and the current moment can be accomplished by training a small neural network with an activation function *tanh*. Formulas (3) to (4) give the relevant calculation:

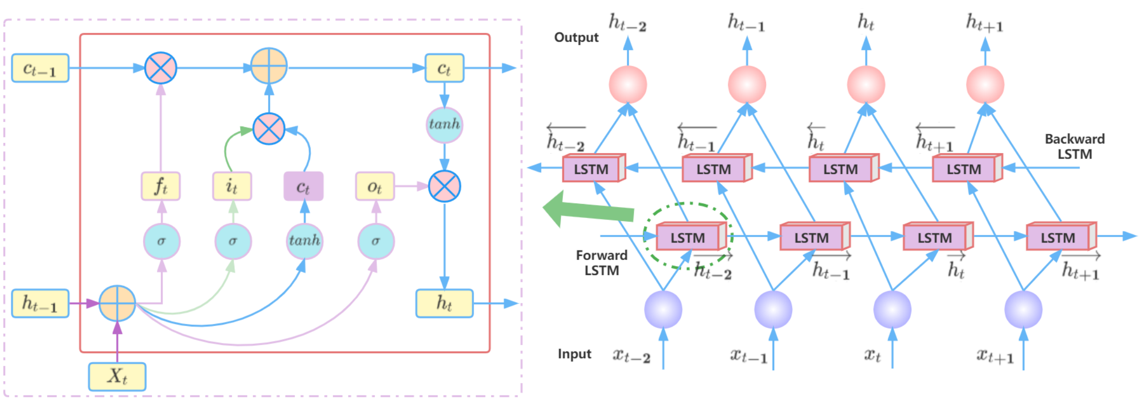
$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, X_t\right] + b_i\right) \tag{3}$$

$$\widetilde{C}_t = tanh\left(W_c \cdot \left[h_{t-1}, X_t\right] + b_c\right) \tag{4}$$

The output f_t of the forget gate is multiplied with the global information c_{t-1} output of the previous layer to indicate information selection. This helps forget a part of the global information. The new information can be extracted with reservations by multiplying the output i_t of the input gate function with the integrated information $\widetilde{C}_t$. Formula (5) gives the relevant calculation:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{5}$$

The output gate in the LSTM cell receives two inputs: the hidden state h_{t-1} of the output of the previous layer and the input x_t of the current time step. We train to obtain a gate



2.jpg

Fig. 2: Structure of the LSTM Unit and Bi-LSTM.

function o_t. Formula (6) gives the calculation for the output gate:

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, X_t\right] + b_o\right) \tag{6}$$

The global information c_t of the current time step is multiplied with the output o_t of the output gate function by a $tanh$ activation function operation to obtain the hidden state h_t of the current moment where the global information has an impact on the next LSTM cell. Formula (7) gives the relevant calculation:

$$h_t = o_t * tanh(C_t) \tag{7}$$

In the formulas (2) to (7) above, W_f represents the weight matrix of the forget gate, W_i represents the weight matrix of the input gate, W_o represents the weight matrix of the output gate. b_f,b_i,b_o,b_c are bias vectors. $*$ represents an element-by-element multiplication calculation. σ represents the activation function which is called sigmoid. $\widetilde{C}_t$ represents the state of the memory cell at time t.

One-way LSTM can only capture sequence information in a forward-to-backward order, which means it can only handle the one-way dependence of above on below. Bi-LSTM, on the other hand, fully considers contextual semantic relationships and learns script features comprehensively by forward and backward propagation, which can fully extract features that are highly relevant to script classification. The hidden layer state H of the Bi-LSTM is obtained by splicing the hidden layer state component $\overrightarrow{h_t}$ obtained by forwarding propagation with the hidden layer state component $\overleftarrow{h_t}$ obtained by backward propagation.

3.2.2. Self-Attention Mechanism

Each part of the cross-site scripting payloads does not contribute equally to the representation of the overall script semantics, and thus needs to be assigned reasonable weights to them. The attention mechanism [16] is a mechanism for extracting important information from a large amount of information by simulating the way the human brain processes information overload, which relies less on external information and focuses more on capturing the internal relevance of data or features. The attention mechanism can directly calculate the dependencies between words in a text sequence without considering the distance between them.

The use of the Self-Attention Mechanism to dynamically adjust the weights of the hidden layer states H of the Bi-LSTM can serve to strengthen the semantic information features extracted by the Bi-LSTM and make the model focus more on the features with high contribution to the classification. Formula (8) gives the calculation for the Self-Attention Mechanism [17]:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{8}$$

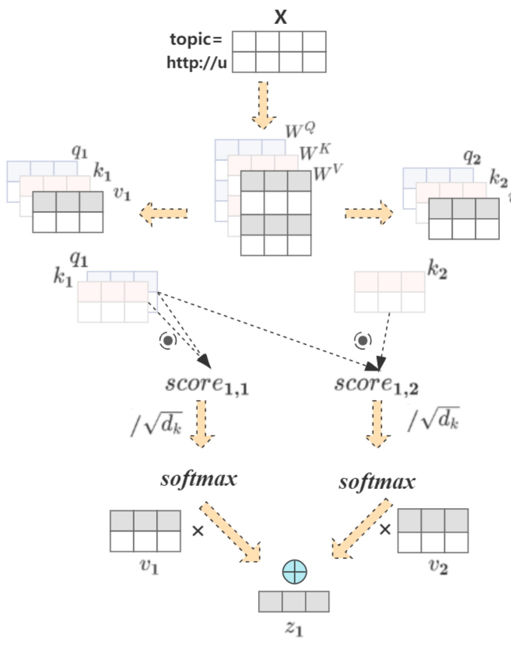
In the formula above, Q, K, and V denote the query matrix, key matrix, and value matrix of the payloads, respectively. d_k denotes the vector dimension. Figure 3 shows a schematic diagram of the internal operation of the Self-Attention Mechanism.

According to the query matrix W_Q, key matrix W_K, and value matrix W_V, three corresponding vectors q_i, k_i, and v_i are generated for each input word vector x_i. The query vector q_i is dot-producted with the key vector k_i to obtain the $score_i$ of the current word. This score indicates the magnitude of the contribution of other words when encoding the current word. Formula (9) gives the relevant calculation:

$$score_i = q_i \cdot k_i \tag{9}$$

The score $score_i$ is divided by the square root of the vector dimension $\sqrt{d_k}$ and normalized using the $softmax$ function. This decouples the steepness of the distribution of the $softmax$ function from the vector dimension, which in turn promotes the gradient to remain stable during training.

For each word, the contribution score of any word to the current word encoding can be obtained by such a computational procedure [18]. The $softmax$ score is multiplied with



3.jpg

Fig. 3: Internal operation diagram of Self-Attention Mechanism.

the corresponding value vector v_i and the resulting vector is weighted and summed to obtain the final output z for the current word. Formula (10) gives the relevant calculation:

$$z = \sum softmax(\frac{score_i}{\sqrt{d_k}})v_i \tag{10}$$

Embedding the Self-Attention Mechanism layer after the Bi-LSTM layer that extracts the semantic features of the cross-site scripting payloads can strengthen the semantic information features. This means that dynamically assigning weights to them helps improve the classification results.

3.2.3. Dual-channel feature fusion

After the feature extraction channel I, we can obtain the multi-scale cross-site scripting payload features extracted by three parallel Conv1D layers. After the feature extraction channel II, we can obtain the semantic features of the cross-site scripting payload reinforced by the Self-Attention Mechanism. The dual-channel feature fusion layer of the C-BLA model fuses the two parts of features to obtain a more scientific and comprehensive feature representation, which makes the classifier have a better classification effect.

4. Cross-site scripting bypass strategy

4.1. Allowlist bypass strategy

Allowlist means that only trusted characters are allowed to enter and all others are rejected. Usually allowlist-based bypass strategies are more secure. In the face of strong cross-site scripting defenses, HTML tags, attributes, JavaScript code, etc. can be used to construct exploits.

The construction of cross-site scripting payload is possible using HTML tags and attributes.

To take `<a href="javascript:alert(1)">xss</a>` as an example. In this script, the href attribute of the `<a></a>` tag comes with a JavaScript special event that can cause the attack to be triggered.

For the case `<img src="xxx">`, you can start with a single quote `"` and right-tipped brackets `>` paired with the preceding symbols to close the tag before constructing it. A typical example is `<img src=""><img src=x onerror=alert(1)>"`.

Assuming the quotes are allowlisted and filtered, they can also be bypassed using the HTML parsing priority. Here we take `<title><img src="</title><img src=x onerror=alert (1)>"></title>` as an example. This case takes advantage of the fact that `<title></title>` tags have a higher parsing priority than `<img>`, and browsers prioritize the former when parsing, which causes the attack to be triggered. Another typical example is `<img src="x" onerror="alert(1)">` which fails in the case of allowlist filtering of alphabetic brackets. At this point, we can use HTML escaped encoding to encode `alert(1)` as `&#97;&#108;&#101;&#114;&#116;&#40;&#49;&#41;` to bypass it.

In addition, JavaScript can be easily injected with cross-site scripting due to its flexible coding style. Here we take the *constructor* in JavaScript as an example. If the current allowlist only allows the characters `[]$='\()+` and the numbers `0-9`, you can use *Function* and string morphing to bypass it. We take `Object.constructor("alert(1)")()` as another example. It can be morphed into `"..."["\163\165\142\163\164\162"]["\143\157\156\163\164\162\165\143\164\157\162"]("\141\154\145\162\164\50\61\51")()`. This leads to the triggering of the attack.

4.2. Blocklist bypass strategy

Blocklist means that the specified characters are not allowed to be entered, and the rest of the non-restricted characters are allowed to pass.

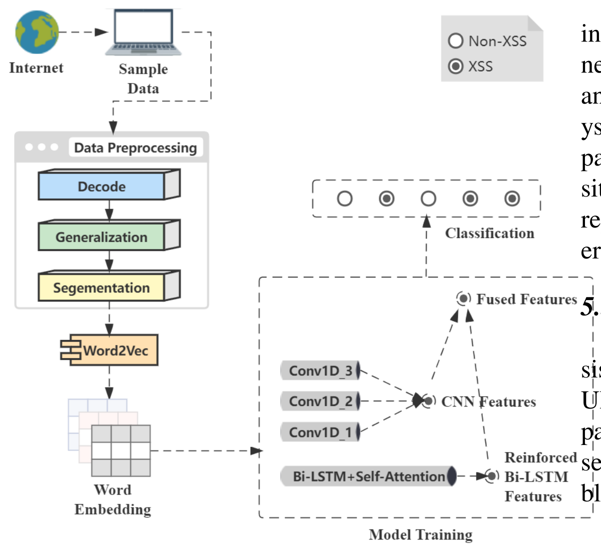
In rich text environments such as email, space, and editors, you can directly enter various types of tag symbols, take case bypass, uncommon non-blocklisted tags, uncompleted opening tags, and trigger events for bypass. For example, writing the original `<script>` as `<sCript>` or `<SCRIPT>` takes advantage of capitals and lower case letters bypassing. For example, `<script src=x?` utilizes an incomplete tag for bypassing. Another example is to add uncommon trigger events such as *ontouchstart* to non-blocklisted tags such as `<a>`, `<img>`, etc. for bypassing purposes. In addition, flexible nesting such as `onl<script>oad` can be taken to bypass.

5. Experiment Design

The experimental flow is shown in Figure 4, which mainly contains the steps of data set acquisition, data pre-processing, and model training.

5.1. Experiment dataset

Currently, there is a scarcity of publicly available datasets within the cybersecurity domain. Meanwhile, in the real



4.jpg

Fig. 4: Experiment flow chart.

problem, the number of cross-site attack scripting is much smaller than the number of normal cross-site request scripting. This means that the dataset corresponding to the problem is generally not a balanced dataset, and there are far more legitimate data than malicious data.

To ensure that the experimental data is true and valid, we collected 31,407 malicious cross-site scripting from site http://www.xssed.com/ as a positive sample and 74063 normal cross-site scripting from site https://curlie.org/en as a negative sample. The composition of the dataset is shown in Table 2, where No-XSS denotes normal cross-site scripting data and XSS denotes malicious cross-site scripting data.

Table 2
Dataset composition.

Type	Dataset	Label
XSS	31407	1
Non-XSS	74063	0

5.2. Experiment environment

The experiment was conducted on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 16G RAM, and NVIDIA GeForce GTX 1050.

5.3. Data preprocessing

5.3.1. Data cleaning

Generally, attackers perform multiple obfuscated encoding of cross-site scripting in order to bypass detection systems. Therefore, decoding operations such as URL decoding, Base64 decoding, and HTML decoding are required for the raw data. Table 3 gives sample examples of several decoding methods.

At the same time, since the cross-site scripting contains information involving allergy, to protect data security, it is necessary to delete the host and path information in the URL, and only retain the cross-site script payload part. After analysis, the numbers, domain name, and path information in the payload do not affect distinguishing whether it is a cross-site attack script or not. Therefore, the URL is uniformly replaced with `http://u`, and all numbers are uniformly generalized to `0`, as shown in Table 4.

5.3.2. Tokenization

After analysis, cross-site scripting payloads generally consist of HTML tags, attributes, events, JavaScript functions, URL links, etc. In order to facilitate feature extraction, the payload needs to be sub-phrased. A set of regularized word separation rules is experimentally designed as shown in Table 5.

After regularization of the scripting, we can obtain the pre-processed data as shown in Table 6.

5.3.3. Word list construction

The word list was constructed according to the TF-IDF algorithm [19], and the frequency of occurrence of all tokens in the positive sample was counted. The TF-IDF values of each word are calculated and ranked in descending order, and the top 3000 words with the highest values are taken to build the word list. The word list sets the number for each word in turn, from smallest to largest, the smaller the number of the first word, the greater the relevance to the cross-site scripting. The words that are not in the word list are replaced with "UNK" in the script sequence, while the words in the word list remain unchanged.

5.3.4. Vectorization

Neural networks are unable to directly process the garbled data after hybrid coding. The original data needs to be decoded, normalized, and split into words, and the split data needs to be mapped and projected into vector space. In this paper, we use Word2Vec [20], a tool introduced by Google in 2013, for word vector training, which contains both CBOW and Skip-gram models. The CBOW model moves the sliding window once per training to obtain a training sample, i.e., to predict the current word based on the context. The word vectors trained by this model are adjusted equally with their surrounding words and are relatively poorly trained for rare words. The Skip-gram model moves the sliding window once per training to obtain multiple training samples, i.e., to predict the context of the current word based on its context. The word vector trained by this model has to be adjusted several times, and it works better for the training of rare words and smaller corpus. It can be seen that although the training time of the Skip-gram model is longer than that of the CBOW model, the training results are relatively better. Therefore, the experiment uses the Skip-gram model to train the XSS word vector. High-frequency word sampling and negative sampling [21] are also used for optimization to avoid overfitting and slow gradient descent due to the large spatial dimension of the word vector. This can improve the quality

Table 3
Decoding examples.

Type	Source code	After Decoding
URL	%3Cscript%3Ealert(1)%3C%2Fscript%3E	<script>alert(1)</script >
Base64	PGltZyBzcmM9eCBvbmVycm9yPWFsZXJ0KDEpPg==	<img src=x onerror=alert(1) >
HTML	&#60;img src=x onerror=alert(1)&#62;	<img src=x onerror=alert(1) >
Unicode	<script>\u0061\u006c\u0065\u0072\u0074(9);</script>	<script>alert(9)</script>

Table 4
Data generalization examples.

Type	Source code	After generalization
http://	topic= http://xxx.com/xss.js<script>alert(document.cookie)	topic= http://u<script>alert(document.cookie)
0-9	<sCRipt>alert(String.fromCharCode(11,22,33))</SCriPt>	<script>alert(String.fromCharCode(0,0,0))</script>

Table 5
Regularized word segmentation rules.

Regular Expression	Splitting Object
(?x)[\w\.]+?\(	Function
"\w+?"	Contents contained in double quotes
'\w+?'	Contents contained in single quotes
http://\w	URL Links
</\w+>	Termination Tags
<\w+>	Start Tags
\w+=	Window Activities

of the trained word vectors while reducing the computational burden. The Skip-gram model framework is shown in Figure 5.

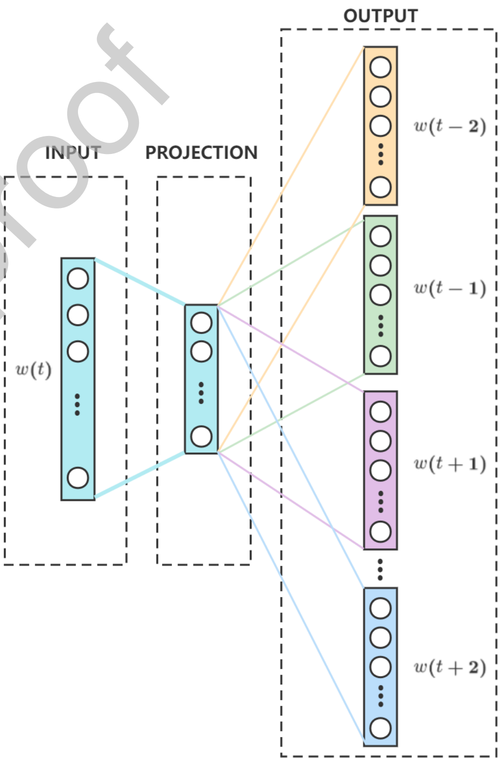
The Skip-gram model consists of three parts: the input layer, the hidden layer, and the output layer. The central word entering the input layer is transformed by the projection of the hidden layer to output its context window in the output layer. The hidden layer weights are obtained by learning the training data from the Skip-gram model to obtain the final word vector representation. Figure 6 shows the context window for Skip-gram model training.

We take the XSS positive sample "topic=http://u<script> alert(document.cookie)</script>" as an example. Suppose "topic=" is selected as the input word, and the sliding window size is defined as 2, which means two words are selected from the left or right side of the current input word. The words contained in the context window at this point (including the current input word) are ['topic=', 'http://u', '<script>']. Suppose "http://u" is selected as the input word, and the sliding window size is defined as 2. The words contained in the context window centered on this input word (including the current input word) are ['topic=', 'http://u', '<script>', 'alert('], and the size of the entire context window is 4.

Calculation of the sampling rate is given as formula (11).

$$P\left(w_i\right) = \left(\sqrt{\frac{Z\left(w_i\right)}{sample}}\right) \times \frac{sample}{Z\left(w_i\right)} \tag{11}$$

Calculation of negative sampling probability is given as



5.jpg

Fig. 5: Structure of Skip-gram model.

formula (12):

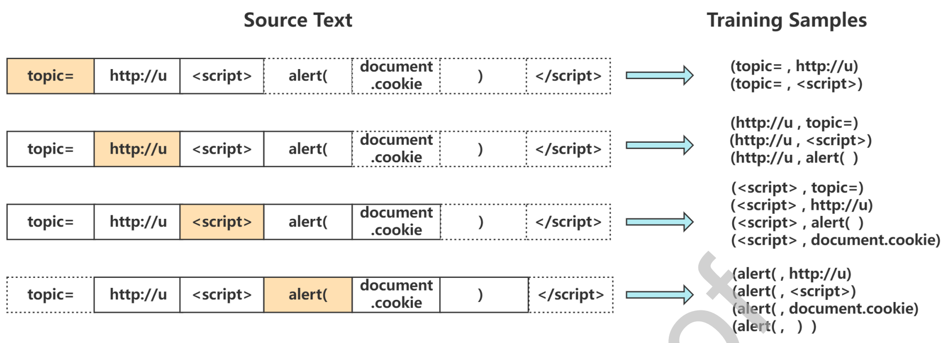
$$P\left(w_i\right) = \frac{f\left(w_i\right)^{\frac{3}{4}}}{\sum_{j=0}^{n}\left(f\left(w_i\right)^{\frac{3}{4}}\right)} \tag{12}$$

The visualization of the high-dimensional word vector using t-SNE is shown in Figure 7.

Two words with similar semantics have a closer distribution in the low-dimensional coordinate system. Some of the words even overlap to represent the word vectors that characterize them are extremely similar. The trained word vectors can characterize the semantics of payload subwords, and the

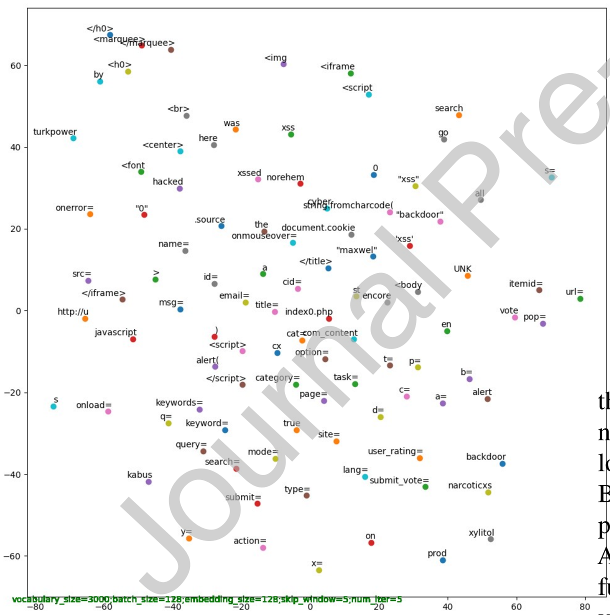
Table 6
Examples after regularized segmentation.

XSS payload	After tokenization
<script>alert(xss)</script>	['<script>', 'alert(', 'xss', ')', '</script>']
oynprodid=><iframe src=http://u></iframe>	['oynprodid=', '>', '<iframe', 'src=', 'http://u', '>', '</iframe>']
msg=><script>alert(0)</script>	['msg=', '>', '<script>', 'alert(', '0', ')', '</script>']
query=<script>javascript alert(document.domain)</script>	['query=', '<script>', 'javascript', 'alert(', 'document.domain', ')', '</script>']



6.jpg

Fig. 6: Context Window.



7.jpg

Fig. 7: High-dimensional word vector visualization (part).

Table 7
Words similar to the word "<script>".

Word	Similarity
</script>	0.5032463073730469
)	0.4145205318927765
alert(	0.30463308095932007
<h0>	0.30144232511520386
by	0.2688441574573517
<marquee>	0.21753743290901184
</h0>	0.20323854684829712
recherche=	0.19963857531547546
id=	0.1927407830953598
searchfor=	0.1906474083662033

words with semantic approximation to the word "<script>" and their similarity are given in Table 7.

5.4. Model construction

The main body of the experimentally constructed model contains two feature extraction channels. Channel I is mainly composed of three Conv1D layers with different sizes of convolutional kernels; channel II is mainly composed of the Self-Attention Mechanism and Bi-LSTM.

The model receives the trained word vectors and feeds them to the dual-channel simultaneously. The convolutional neural network layer extracts the local features of the payload by one-dimensional convolutional operations, and the Bi-LSTM extracts the bi-directional semantic features of the payload and reinforces the semantic features by the Self-Attention Mechanism. The feature fusion layer of the model fuses the above two parts of features and obtains the feature vector for classification by dimensionality reduction. The model sets the Dropout layer to prevent overfitting and uses the Softmax classifier to classify the fused features. The C-BLA model parameters are set as shown in Table 8.

Table 8
Parameter setting.

Parameter	Settings
embedding size	128
epoch	100
batch size	128
CNN kernel size	2-4-6
dropout	0.5
optimizer	Adam
classifier	softmax

6. Experimental Results and Analysis

The experiment deals with a dichotomous classification problem. The classification results are divided into positive and negative categories. The binary classification problem can usually be represented by constructing a confusion matrix as shown in Table 9.

Table 9
Confusion matrix.

Type	Actual XSS	Actual Non-XSS
Predicted XSS	T_P	F_P
Predicted Non-XSS	F_N	T_N

The confusion matrix contains four classes [22]. T_P denotes samples that are actually malicious cross-site scripting and are predicted by the model as malicious scripting. F_P denotes samples that are actually normal cross-site scripting but are predicted by the model as malicious scripting. T_N denotes samples that are actually normal cross-site scripting and are predicted by the model as normal scripting. F_N denotes samples that are actually malicious cross-site scripting but are predicted by the model as normal cross-site scripting.

Based on this confusion matrix, the experimental results were evaluated using Accuracy, Precision, Recall, and F_1 values. The calculation formula is given by equations (13) to (16).

$$Accuracy = \frac{T_P + T_N}{T_P + F_N + F_P + T_N} \tag{13}$$

$$Precision = \frac{T_P}{T_P + F_P} \tag{14}$$

$$Recall = \frac{T_P}{T_P + F_N} \tag{15}$$

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{16}$$

To evaluate the performance of the model proposed, the experiments were conducted using the ten-fold cross-validation method. The experimental results are shown in Table 10.

Table 10
Results of the ten-fold cross-validation experiment.

Fold	Accuracy	Recall	F_1
1	0.993078601	0.992533261	0.995032324
2	0.992414905	0.990120449	0.994562262
3	0.994595619	0.992879215	0.996158253
4	0.993837110	0.992858105	0.995608405
5	0.994371954	0.988530563	0.993827579
6	0.993509718	0.990483849	0.994683357
7	0.992414905	0.992750705	0.99462004
8	0.992414905	0.989584742	0.994562262
9	0.994442381	0.992346283	0.995544701
10	0.994289221	0.987365538	0.992839071

The ten-fold cross-validation method divides the data set into ten subsets, one of which is taken each time as the test set and the rest as the training set. This approach allows each subset to act as a test set for evaluating model performance. Taking the average of ten sets of model evaluation indexes as the performance indexes of the final model, we can obtain the classification accuracy rate of the C-BLA model as 99.4%, the recall rate as 99.1%, and the F_1 value as 99.4%.

6.1. Comparison of different feature extraction channels

To compare the performance of the proposed dual-channel feature fusion model, several groups of comparison models are set up in the experiments. For example, the model with only convolutional feature extraction channel, the model with only LSTM feature extraction channel, etc. The specific experimental comparison results are shown in Table 11.

Table 11
Experimental results of single and double feature extraction channels.

Channel setting		Evaluation metrics		
		Precision	Recall	F_1
CNN channel	CNN-2	95.2%	94.6%	94.9%
	CNN-4	96.1%	95.3%	95.7%
	CNN-6	95.8%	95.2%	95.5%
	CNN-2&4	97.3%	96.9%	97.1%
	CNN-2&6	96.8%	97.4%	97.1%
	CNN-4&6	97.1%	96.5%	96.8%
	CNN-2&4&6	97.7%	97.3%	97.5%
LSTM channel	LSTM	96.4%	96.3%	96.4%
	Bi-LSTM	97.2%	96.7%	97.0%
	Bi-LSTM+SA	**98.2%**	**98.1%**	**98.1%**
Proposed	**C-BLA**	**99.8%**	**99.1%**	**99.4%**

CNN-2, CNN-4, and CNN-6 in Table 11 denote the convolutional neural network layers with convolutional kernel sizes of 2, 4, and 6, respectively. CNN-2&4 denotes the two parallel Conv1D layer with convolutional kernel sizes of 2 and 4. CNN-2&6 denotes the two parallel Conv1D layer with convolutional kernel sizes of 2 and 6. CNN-4&6 denotes the two parallel Conv1D layers with convolutional kernel sizes of 4 and 6. CNN-2&4&6 denotes the three parallel Conv1D layers with convolutional kernel sizes of 2, 4, and 6. Bi-LSTM+SA denotes the Bi-LSTM with embedded Self-Attention Mechanism.

When only a single convolutional feature extraction channel is set, the maximum precision rate is 96.1%, the maximum recall rate is 95.3%, and the maximum F_1 value is 95.7%.

When two or three parallel convolutional feature extraction channels are set, the precision rate is up to 97.7%, the recall rate is up to 97.4%, and the F_1 value is up to 97.5%.

When only LSTM, Bi-LSTM or Bi-LSTM followed by Self-Attention Mechanism is set as the feature extraction channel, the maximum precision rate is 98.2%, the maximum recall rate is 98.1% and the maximum F_1 value is 98.1%.

The analysis shows that three parallel Conv1D layers with convolutional kernel sizes of 2, 4, and 6 can be used as the extracted feature channels to obtain better classification features. At the same time, the Self-Attention Mechanism acting on the Bi-LSTM can also obtain better classification features. The proposed model C-BLA obtained the highest precision rate of 99.8%, the highest recall rate of 99.1%, and the highest 99.4% F_1 value by fusing the features extracted from the above dual channels.

6.2. Comparison with machine learning algorithms

To compare the performance of the proposed model with traditional methods, we selected several representative machine learning algorithms.

Wang et al.[23] selected ADTree and AdaBoost for detecting XSS. Their experiments achieved good detection results at the time. ADTree is a decision tree learning algorithm based on boosting. It outperforms general decision trees in classification accuracy and has a wide range of practical applications. AdaBoost stands for Adaptive Boosting. It works by training several weak classifiers and combining them into a strong classifier with a smaller classification error rate.

Nunan et al.[24] selected SVM and Naive Bayes for detecting XSS. SVM is a classification algorithm that solves for the maximum-margin hyperplane by learning samples. It allows for non-linear classification with high robustness through a kernel approach. Naive Bayes is a classification algorithm based on Bayes' theory and the assumption of conditional independence of features.

G. Habibi et al.[25] selected KNN as one of their classifiers for detecting XSS. KNN is a common classification algorithm that determines the classification of the current sample based on the class of the nearest few samples. The disadvantage of this method is that it is usually computationally intensive.

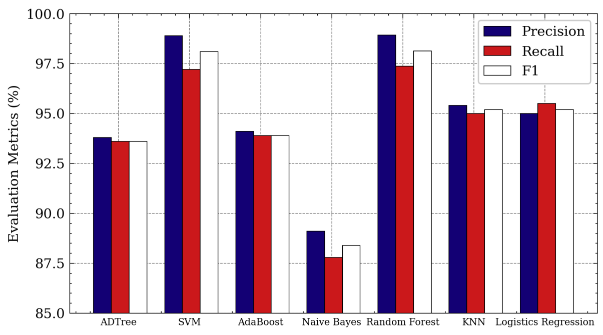
In addition, Random Forest is a classifier that uses multiple trees to train and predict samples. Logistic Regression is one of the most fundamental algorithms in classification problems. It can be used for binary classification as well as for multi-classification.

We conducted comparative experiments using the seven representative machine learning algorithms mentioned above. The results are shown in Table 12.

Table 12
Comparison with machine learning algorithms.

Algorithm		Evaluation metrics		
		Precision	Recall	F_1
Machine learning algorithms	ADTree	93.8%	93.6%	93.6%
	SVM	98.7%	97.2%	97.9%
	AdaBoost	94.1%	93.9%	93.9%
	Naive Bayes	89.1%	87.7%	88.4%
	Random Forest	98.9%	97.3%	98.1%
	Logistic Regression	95.0%	95.5%	95.2%
	KNN	95.4%	95.0%	95.2%
Proposed	C-BLA	**99.8%**	**99.1%**	**99.4%**

The results are shown in Figure 8 for more visual comparison.



8.jpg

Fig. 8: Comparison of the proposed method with machine learning algorithms.

Figure 8 shows that SVM and Random Forest perform better for detection, with Random Forest slightly outperforming the former in three metrics. Among these algorithms, Naive Bayes achieves the worst detection results.

Compared with Random Forest which has the best detection results in all seven machine learning algorithms, our proposed method is 0.9% higher in precision rate, 1.8% higher in recall rate, and 1.3% higher in F_1 value. Due to the ability of deep learning models to learn features automatically, the method can avoid the subjectivity and complexity of traditional manual feature extraction.

6.3. Comparison with deep learning algorithms

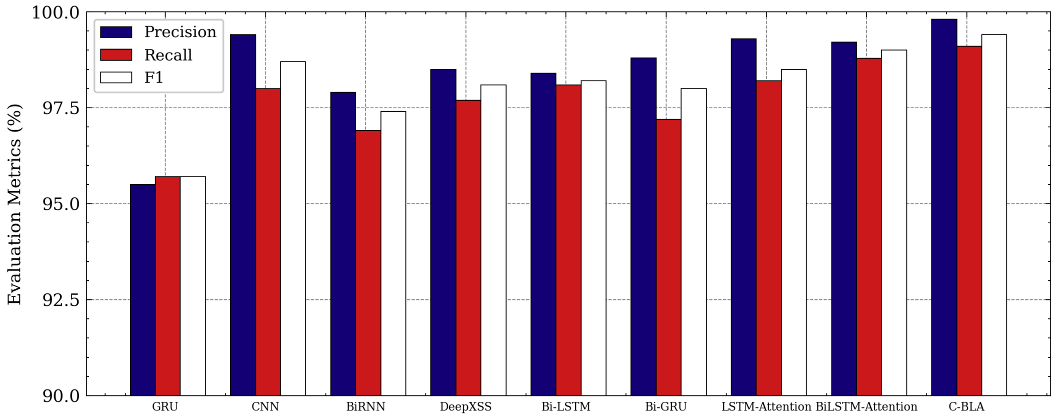
To compare the performance of the proposed model with existing deep learning methods, we selected several representative algorithms.

J. Kumar's team [26] used CNN to classify cross-site scripting which was represented by Unicode. Fang's team [11] used a LSTM-based model called DeepXSS to classify the processed cross-site scripting and compared their results with ADTree and AdaBoost. Farea et al. [27] used Bi-LSTM to detect cross-site scripting and compared their results with several representative detection methods. Lei's team [12] improved LSTM by adding the attention mechanism and achieved better results than the original model. Fang's team [28] proposed a bi-directional RNN-based detection model adding the attention mechanism from third-party in 2020, and experimental results showed that the method outperformed models such as Random Forest, Naive Bayes, and SVM.

To make the results more convincing, we constructed the relevant models based on the methods of the researchers mentioned above and applied them to our dataset for comparative experiments. The detailed experimental results are shown in and Table 13.

The results are shown in Figure 9 for more visual comparison.

Through Figure 9, we find that GRU is relatively poor in terms of detection rate for cross-site scripting and the detection rate of the improved BiRNN is higher than RNN. The LSTM and Bi-LSTM models improved by the atten-



9.jpg

Fig. 9: Comparison of the proposed method with deep learning methods.

Table 13
Comparison with deep learning models.

Model		Evaluation metrics		
		Precision	Recall	F_1
Deep learning models	CNN	99.4%	98.0%	98.7%
	BiRNN	97.9%	96.9%	97.4%
	DeepXSS	98.5%	97.7%	98.1%
	BiLSTM	98.4%	98.1%	98.2%
	GRU	95.6%	95.7%	95.7%
	BiGRU	98.8%	97.2%	98.0%
	LSTM-Attention	99.3%	98.2%	98.5%
	BiLSTM-Attention	99.2%	98.8%	99.0%
Proposed	**C-BLA**	**99.8%**	**99.1%**	**99.4%**

tion mechanism outperformed the original model in terms of detection rate, indicating that the attention mechanism contributes to cross-site scripting classification. CNN improved by Kumar's team has the highest precision rate, but is still 0.4% lower than our model. The best recall rate and F_1 value of these models are from the Bi-LSTM modified by the attention mechanism, which achieves better detection results than the unidirectional LSTM, despite the larger number of parameters due to the inclusion of bidirectional semantic features. Compared to BiLSTM-Attention, our proposed model improves by 0.6% in precision, 0.3% in the recall, and 0.4% in F_1 value. The comparison means that in the face of a large number of cross-site scripting attacks, our model can identify malicious scripts more effectively than a single deep learning model, thus maintaining network security.

To further evaluate our proposed model, we designed several additional comparison experiments. We have split the components of the proposed model and restructured them in a serial and parallel manner.

CNN-LSTM(s) denotes the CNN and LSTM serial architecture, extracting features sequentially. CNN-BiLSTM(s) denotes the CNN and Bi-LSTM serial architecture, extracting features sequentially. CNN-LSTM(p) indicates that both CNN and LSTM are used in parallel to extract features simultaneously. CNN-BiLSTM(p) indicates that both CNN and Bi-LSTM are used in parallel to extract features simultaneously. CNN-BiLSTM+SA(p) presents that both CNN and Bi-LSTM+SA are used in parallel to extract features simultaneously. BiLSTM+SA denotes Bi-LSTM embedded in Self-Attention Mechanism. The detailed experimental results are shown in Table 14.

Table 14
Comparison of serial and parallel model architectures.

Model		Evaluation metrics		
		Precision	Recall	F_1
Serial	CNN-LSTM(s)	99.15%	96.27%	97.69%
	CNN-BiLSTM(s)	99.74%	97.44%	98.58%
Parallel	CNN-LSTM(p)	99.65%	96.79%	98.20%
	CNN-BiLSTM(p)	99.67%	99.04%	99.35%
Proposed	**C-BLA**	**99.8%**	**99.1%**	**99.4%**

Table 14 shows that CNN-BiLSTM outperforms CNN-LSTM in both serial and parallel architectures. CNN-LSTM based on the parallel architecture outperforms the serial architecture in all three metrics. The CNN-BiLSTM based on the parallel architecture is 0.07% lower in precision than the serial architecture, but 1.6% and 0.77% higher in recall and F_1 value, respectively. Our proposed model outperformed the highest precision of these four combined models by 0.06%, also outperformed the highest recall by 0.06%, and outperformed the highest F_1 value by 0.05%. Although the improvement over the four combined models is not significant, the small increase in detection rate for cross-site scripting detection means that a large number of malicious scripts can be identified, thus securing cyberspace.

7. Conclusion

In this paper, we propose a detection model called C-BLA with two-channel feature fusion embedded in the Self-Attention Mechanism, which can detect whether the data submitted by users contain cross-site scripting attack. It is experimentally demonstrated that the model optimizes the detection performance by combining CNN, Bi-LSTM, and Self-Attention Mechanism. On the one hand, Conv1D layers with different convolutional kernel sizes are designed to extract local features of payloads at different scales. On the other hand, the contextual information is fully extracted by forwarding and backward propagation computation of the Bi-LSTM, followed by embedding the Self-Attention Mechanism to strengthen the extracted semantic features of the payloads. Finally, the local features are fused with the reinforced semantic features for classification. Using the ten-fold cross-validation method and comparing traditional machine learning models such as SVM and existing deep learning models such as LSTM, the C-BLA model has improved in precision, recall, and F_1 value.

CRediT authorship contribution statement

Tianle Hu: Conceptualization, Methodology, Software, Writing - original draft. **Chonghai Xu:** Resources, Investigation, Validation.. **Shenwen Zhang:** Data curation.. **Shuangshuang Tao:** Project administration, Supervision.. **Luqun Li:** Formal analysis, Writing - reviewing & editing.

Acknowledgments

This work was supported by Analysis and Simulation of Computer Networks in the Sub-project of Construction of First-class Graduate Education in High-level Local Universities in 2021 (209-AC9103-21-368012452).

References

[1] Miao Liu, Boyu Zhang, Wenbin Chen, and Xunlai Zhang. A survey of exploitation and detection methods of xss vulnerabilities. *IEEE access*, 7:182004–182016, 2019.

[2] OWASP. Owasp top 10-2017,The Ten Most Critical Web Application Security Risks, 2017. https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf, Last accessed on 2021-12-25.

[3] OWASP. Owasp top 10: 2021 web application security risks, 2021. https://owasp.org/www-project-top-ten/, Last accessed on 2022-1-20.

[4] Gary Wassermann and Zhendong Su. Static detection of cross-site scripting vulnerabilities. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 171–180. IEEE, 2008.

[5] Shashank Gupta and BB Gupta. Cssxc: Context-sensitive sanitization framework for web applications against xss vulnerabilities in cloud environments. *Procedia Computer Science*, 85:198–205, 2016.

[6] Mahmoud Mohammadi, Bill Chu, and Heather Richter Lipford. Detecting cross-site scripting vulnerabilities through automated unit testing. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 364–373. IEEE, 2017.

[7] Inian Parameshwaran, Enrico Budianto, Shweta Shinde, Hung Dang, Atul Sadhu, and Prateek Saxena. Dexterjs: robust testing platform for dom-based xss vulnerabilities. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 946–949, 2015.

[8] Ran Wang, Guangquan Xu, Xianjiao Zeng, Xiaohong Li, and Zhiyong Feng. Tt-xss: A novel taint tracking based dynamic detection framework for dom cross-site scripting. *Journal of Parallel and Distributed Computing*, 118:100–106, 2018.

[9] Stanislav Kascheev and Tatyana Olenchikova. The detecting cross-site scripting (xss) using machine learning methods. In *2020 Global Smart Industry Conference (GloSIC)*, pages 265–270. IEEE, 2020.

[10] Shailendra Rathore, Pradip Kumar Sharma, and Jong Hyuk Park. Xss-classifier: an efficient xss attack detection approach based on machine learning classifier on snss. *Journal of Information Processing Systems*, 13(4):1014–1028, 2017.

[11] Yong Fang, Yang Li, Liang Liu, and Cheng Huang. Deepxss: Cross site scripting detection based on deep learning. In *Proceedings of the 2018 international conference on computing and artificial intelligence*, pages 47–51, 2018.

[12] Li Lei, Ming Chen, Chengwan He, and Duojiao Li. Xss detection technology based on lstm-attention. In *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*, pages 175–180. IEEE, 2020.

[13] Kai Liu, Yun Zhou, Qingyong Wang, and Xianqiang Zhu. Vulnerability severity prediction with deep neural network. In *2019 5th International Conference on Big Data and Information Analytics (BigDIA)*, pages 114–119. IEEE, 2019.

[14] Chen Zhang, Renzhong Guo, Xiangyuan Ma, Xi Kuai, and Biao He. W-textcnn: A textcnn model with weighted word embeddings for chinese address pattern classification. *Computers, Environment and Urban Systems*, 95:101819, 2022.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[17] Prashant Srivastava, Saptarshi Bej, Kristina Yordanova, and Olaf Wolkenhauer. Self-attention-based models for the extraction of molecular interactions from biological texts. *Biomolecules*, 11(11):1591, 2021.

[18] P Bhuvaneshwari, A Nagaraja Rao, and Y Harold Robinson. Spam review detection using self attention based cnn and bi-directional lstm. *Multimedia Tools and Applications*, 80(12):18107–18124, 2021.

[19] Apra Mishra and Santosh Vishwakarma. Analysis of tf-idf model and its variant for document retrieval. In *2015 international conference on computational intelligence and communication networks (cicn)*, pages 772–776. IEEE, 2015.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[22] Zhonglin Liu, Yong Fang, Cheng Huang, and Jiaxuan Han. Graphxss: an efficient xss payload detection approach based on graph convolutional network. *Computers & Security*, 114:102597, 2022.

[23] Rui Wang, Xiaoqi Jia, Qinlei Li, and Shengzhi Zhang. Machine learning based cross-site scripting detection in online social network. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, pages 823–826. IEEE, 2014.

[24] Angelo Eduardo Nunan, Eduardo Souto, Eulanda M Dos Santos, and Eduardo Feitosa. Automatic classification of cross-site scripting in web pages using document-based and url-based features. In *2012 IEEE symposium on computers and communications (ISCC)*, pages 000702–000707. IEEE, 2012.

[25] Gulit Habibi and Nico Surantha. Xss attack detection with machine learning and n-gram methods. In *2020 International Conference on Information Management and Technology (ICIMTech)*, pages 516–520. IEEE, 2020.

[26] Jitendra Kumar, A. Santhanavijayan, and Balaji Rajendran. Cross site scripting attacks classification using convolutional neural network. In *2022 International Conference on Computer Communication and In-*

formatics (ICCCI), pages 1–6, 2022.

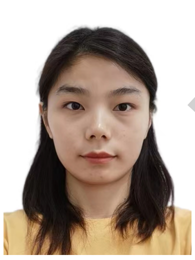
[27] Abdulgbar AR Farea, Chengliang Wang, Ebraheem Farea, and Abdulfattah Ba Alawi. Cross-site scripting (xss) and sql injection attacks multi-classification using bidirectional lstm recurrent neural network. In *2021 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pages 358–363. IEEE, 2021.

[28] Yong Fang, Yijia Xu, Peng Jia, and Cheng Huang. Providing email privacy by preventing webmail from loading malicious xss payloads. *Applied Sciences*, 10(13):4425, 2020.

Tianle Hu is currently pursuing the master degree with the College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China. His current research interests include cyberspace security, machine learning and attack detection.

Chonghai Xu is currently pursuing the master degree with the College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China. His current research interests include computer network attack and defense.

Shenwen Zhang is currently pursuing the master degree with the College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China. His current research interests include network low latency transmission with intelligent algorithms and web security.



Shuangshuang Tao is currently pursuing the master degree with the College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai, China. Her current research interests include network security and traffic classification.

Luqun Li D., Professor, Head of Department of Computer Science, Shanghai Normal University, Director of Wireless Network and Computation Laboratory, "Young Academic Backbone of Shanghai Normal University"; engaged in research and teaching of computer networks, wireless sensor networks, group intelligence algorithms, network GIS, etc. In recent years, he has presided over the research on "Research on wireless sensor network adaptive routing protocol based on dynamic ant colony algorithm", "Research on wireless mobile web service description and discovery algorithm", "Research on wireless sensor network topology control based on DTN". He has also submitted the "Feasibility Study of Tolerable Delay Routing Algorithm for Wireless Sensor Network" with Prof. Sherman (Xuemin) Shen and participated as a core member of the project team in the project of Shanghai Jiaotong University. As a core member of the project team, he participated in the National Natural Science Foundation of China (NSFC) "Research on Transaction Processing Technology in Grid Environment" hosted by Prof. Minglu Li of the Department of Computer Science, and the major research project of Shanghai Science and Technology Commission "Based on Large-scale Logistics Spatial Information".

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:
Luqun Li reports financial support was provided by Shanghai Normal University.

Credit Author Statment

Tianle Hu: Conceptualization, Methodology, Software, Writing - original draft.
Chonghai Xu: Resources, Investigation, Validation.
Shenwen Zhang: Data curation.
Shuangshuang Tao: Project administration, Supervision.
*Luqun Li: Formal analysis, Writing - Reviewing and Editing.