Hindawi
Security and Communication Networks
Volume 2022, Article ID 3434920, 11 pages
https://doi.org/10.1155/2022/3434920

Research Article

A New Method for WebShell Detection Based on Bidirectional GRU and Attention Mechanism

Zhiqiang Liu[1,2] **Daofeng Li**[1,2] **and Lulu Wei**[1,2]

[1]*School of Computer and Electronics Information, Guangxi University, Nanning 530004, China*
[2]*Guangxi Colleges and Universities Key Laboratory of Multimedia Communications and Information Processing, Guangxi University, Nanning 530004, China*

Correspondence should be addressed to Daofeng Li; ldf-0123@gxu.edu.cn

Received 26 September 2021; Accepted 23 February 2022; Published 24 March 2022

Academic Editor: Yanhui Guo

With the rapid development of hacker technology, network security issues have become increasingly serious. Uploading WebShell is one of the most common attack methods used by network intruders. WebShell escape technology is changing with each passing day, and the traditional method based on feature matching is difficult to accurately detect. In order to detect WebShell more accurately and mitigate the threat caused by WebShell attacks, a WebShell detection method combining bidirectional GRU (gated recurrent unit) and attention mechanism is proposed for the first time. First, the sample is preprocessed to remove useless information such as annotations. Then, the sample is divided into a series of words, the word2vec model is used to obtain the word vector, and finally, the word vector is input into the network for prediction. According to the experimental results, compared with peer methods, the method in this study performs better in performance indicators such as accuracy rate, recall rate, and F1 value. The model not only detects the PHP-type WebShell but also has a good performance on the WebShell written in JSP, ASPX, or ASP languages. The detection accuracy of PHP-type, JSP-type, and ASP-type WebShell reached 99.36%, 99.23%, and 99.87%, respectively, and the recall rate was 98.6%, 99.13%, and 99.56%, respectively.

1. Introduction

The development and popularization of network information technology have injected new vitality into modern society. Today, the Internet has played an important role in all aspects of life, greatly facilitating our lives. However, the Internet is a double-edged sword. While facilitating life, it also provides new criminal methods to lawbreakers. In recent years, illegal cyberattacks have become more frequent. Cyberattacks can affect personal privacy and even threaten national security. According to the "2020 China Internet Network Security Report" [1] issued by the National Computer Network Emergency Technology Coordination Center of China in 2020, the number of server IPs controlled by Trojan horses or bots in China reached 65,865 and independent host reached up to 8,560,434, the number is terrifying. Nearly 100,000 web pages have been tampered with, including nearly 500 government websites. In 2020, more than 50,000 websites were tested for implanting WebShell, including 256 government websites. The network security situation is severe. It is urgent to raise citizens' awareness of network security and develop network security technology.

Among many network attack methods, uploading WebShell is especially favoured by attackers. WebShell is a web page backdoor written in a web programming language. Commonly used languages include PHP, JSP, ASPX, etc. It is flexible and efficient, and its concealed features make it appear very frequently during intrusions. In the process of network intrusion, the attacker first checks the website system to find out the vulnerabilities that may be valuable, including SQL injection [2], weak password, arbitrary file upload, arbitrary code execution, and remote file inclusion [3]. Further, through these vulnerabilities, an attacker can upload WebShell. Through WebShell, attackers can collect information, manipulate databases, edit files, execute system

commands, and elevate permissions to provide protection for further intranet intrusions [4]. WebShell plays an important role for intruders. If the WebShell uploaded by the intruder can be accurately detected and cleared in time, the network security can be largely guaranteed.

WebShell is written in the same language as normal web page files, with little difference from normal files. The current methods of detecting WebShell can be roughly classified into two categories, namely, dynamic detection and static detection. Dynamic detection can further determine whether the file is WebShell by monitoring network traffic or using HOOK technology [5] to observe the behaviour of the file. This type of method has high false positives and is inconvenient to implement. Static detection mainly recognizes text without running the text by collecting statistical features of the text, analyzing log information, and constructing a feature database. This type of method has a high accuracy rate, but it is difficult to detect unknown samples. In order to further improve the accuracy of the detection algorithm, the ability to detect unknown samples is improved. We propose a detection method based on bidirectional GRU and attention mechanism. Firstly, the meaningless content such as annotation information is removed in the sample. Then, segment the sample to get a series of words. Next, the words are vectorized and input into the network for detection. Experiments show that this method can accurately detect WebShell and has higher accuracy and recall rate compared with other methods.

The main contributions of this study are as follows:

(1) For the first time, the bidirectional GRU network and attention mechanism are used to detect malicious WebShell code.

(2) The method proposed in this study can effectively detect the WebShell of multiple languages such as PHP, ASP, and Java, rather than a single one.

(3) The experimental results show that it surpasses the previous research in performance indicators such as recall rate, accuracy rate, precision, and F1 value.

(4) There is no need to convert the sample into intermediate code, and the original text is directly used for detection, which is lighter than the previous method.

In the next section, we review some related research and outline the motivation of our research. In Section 3, we will introduce our model in detail. The experiment and result analysis are in Section 4. Next is a brief application scenario description. Finally, we summarize our work in Section 5.

2. Related Works

Researchers have done some research to detect WebShell. In terms of traditional static detection, Yong [6] et al. proposed a php-WebShell detection method that combines FastText [7] and RF (random forest) algorithms. On the one hand, it collects statistical characteristics such as the longest word length, information entropy, coincidence index, feature value, and blacklist function of the text. On the other hand, the sample code is converted into an opcode sequence through VLD [8], and the FastText model is used for modelling. Then, the two types of features are combined and input to the random forest algorithm for detection. The experimental results show that their method obtains a high accuracy rate, but the processing process of the method is complicated, and it is not easy to apply the method to WebShell written in other types of languages. You et al. proposed a php-WebShell detection method of NB-opcode [4]. After converting the PHP code into opcode, the bag-of-words model and the TF-IDF (term frequency-inverse document frequency) algorithm are used to select high-frequency and distinguishable words. In the experiment, the SVM (support vector machine), random forest algorithm and NB-opcode algorithm, NB-opcode are compared to achieve the best performance. Its accuracy, F1 value, and recall rate all exceed 97%, but the method proposed in this paper is not easy to be used to detect webshells written in other languages, and recall rate and accuracy rate can be further improved. Zhang et al. proposed a combined model that can identify PHP- and JSP-type WebShell in 2020 [9]. First, the samples are classified according to the type of writing language, the WebShell classified as PHP type extracts the abstract syntax tree structure, and WebShell classified as JSP type extracts the bytecode. The TF-IDF and word2vec are used to vectorize the abstract syntax tree and byte information. Finally, neural networks, XGBoost (eXtreme gradient boosting) [10], RF and SVM classification algorithms, the XGBoost algorithm are compared to achieve the best results. Onan proposed multiple classifier systems for web page classification [11], and four different feature selections and four different ensemble learning methods are used based on four different base learners. The results show that bagging and random subspace ensemble methods and correlation-based and consistency-based feature selection methods are used to obtain better results in terms of accuracy rates. Designing a Webshell detection model can also refer to this web page classification model.

In terms of dynamic detection, Wang et al. proposed a high-speed network-based WebShell detection scheme [12], which can realize WebShell detection and traceability by capturing and analyzing network traffic and matching with known signatures. However, during the pilot deployment in a real network environment, this solution has many false positives and its generalization ability needs to be improved. Through the above analysis, the traditional static detection and dynamic detection research methods are mainly used to detect PHP WebShell, while other types of WebShells are less researched. On the one hand, the traditional method of detection has a high false alarm rate and the accuracy rate can be further improved. On the other hand, traditional detection methods are mostly based on the characteristics of known samples, and the detection ability of new samples is very weak.

Deep learning technology has been applied in many fields such as image recognition, machine translation, sentiment analysis [13], text classification [14], and spam detection [15] and has achieved very good performance. Xian proposed a WebShell recognition method based on CNN (convolutional neural network) [16]. By obtaining the

opcode of the PHP sample, the opcode sequence is converted into a byte stream and further converted into a 2D image. The experimental results show that this method of converting codes into images and then using convolutional neural networks for classification is very effective. The accuracy of this method exceeds 96%. Zhou proposed a scheme based on RNN (recurrent neural network) [17], which uses a two-layer GRU structure and uses TF-IDF to extract 200 keywords for training and judgment. The experimental results show that the scheme has higher accuracy than support vector machines and CNN. Onan proposed an effective sarcasm identification framework on social media data [18] using three-layer stacked bidirectional LSTM to identify sarcastic text documents. The experimental results show the presented model yields promising results with a classification accuracy of 95.30%. Solutions based on deep learning can achieve better accuracy and recall rates than traditional machine learning methods and have the ability to identify new samples. In recent years, the attention mechanism [19] has been used in the field of text classification and achieved good results. The attention mechanism can assign important information with heavier weights while irrelevant information will be assigned a lower weight, which can further improve the accuracy of text classification.

The WebShell can be seen as a sequence, which is very suitable for processing with RNN. We can design RNN-based detection models. In view of this, this study proposes a WebShell detection method based on the deep learning algorithm, which can accurately identify multiple types of WebShell.

3. Model Architecture

The overall structure of the detection model designed in this study is shown in Figure 1, which includes sample preprocessing, word vectorization, bidirectional GRU layer, attention layer, and fully connected layer.

3.1. Sample Preprocessing. The raw sample script contains a lot of comment information, which does not contain actual semantic information. First, the comment information in the code was removed according to the programming syntax. Then, word segmentation is performed, and the sample is segmented according to the symbol list "<|(|)|[|]|"|'|<|>|; |=|/|?|$|#|@|,|.|{|}|!|-|:|%". When one of these characters is encountered, a split is done. The word segmentation result of a WebShell sample is shown in Figure 2:

After the word segmentation is performed according to the above rules, the number of words contained in the sample after the word segmentation is not the same. To complete the training and detection, the number of words in the sample needs to be unified. The method of unifying the number of words will be described in Section 4.

3.2. Word Vectorization. The text vectorized representation method is divided into discrete representation and distributed representation. Discrete representation easily leads to problems such as data sparseness and loss of semantic information. One-hot and bag-of-words models (BOW) are two typical discrete representation methods. One-hot uses a sparse matrix to vectorize a set of words. When the number of words is large, the dimension of the one-hot vector is huge and this method cannot retain semantic information. BOW assumes that for a document, information such as word order and word meaning is ignored, and each word is regarded as independent and does not depend on the appearance of other words. The vector dimension generated by this method is consistent with the number of words. When the corpus increases, the vector dimension will inevitably be large and sparse, and the method cannot retain semantic information. Word2Vec [20] is a distributed representation method. The training results not only consider contextual information but also that the vector is dense and the dimensions can be set according to actual needs. Word2Vec is a shallow neural network model. There are CBOW (continuous bag-of-words) and skip-gram models, as shown in Figures 3 and 4, respectively.

CBOW model is a three-layer neural network that inputs known context information and outputs predictions for the current word. The CBOW model performs one-hot encoding on the input words, then sums them through a hidden layer, and finally calculates the generation probability of each word through the activation function *softmax*. By training the network, the overall probability of word generation in the corpus is maximized, and the weight matrix of the neural network obtained at this time is the word vector result. The skip-gram model is the opposite of the CBOW model. Skip-gram model predicts the probability of surrounding words by inputting a known word. This article chooses the CBOW model to train word vectors.

3.3. Bidirectional GRU Layer. RNN is composed of an input layer, a hidden layer, and an output layer. The structure is shown in Figure 5.

At time t, the input of the network is x_t, the state of the hidden layer is s_t, and the output is o_t. The state s_t at time t is not only related to the output at time t but also related to the state of the hidden layer at the previous time. At $t = 1$, the calculation of formulas (1)–(3) is performed, s_0 is generally 0, and W, U, and V are the weight matrices shared at each time step, which are randomly initialized and then trained. x_1 is the input at t_1, s_1 is the hidden layer state at t_1, and o_1 is the output at t_1.

$$h_1 = Ux_1 + Ws_0, \tag{1}$$

$$s_1 = f(h_1), \tag{2}$$

$$o_1 = g(Vs_1). \tag{3}$$

When the time is t, the expansion of RNN is as shown in the following formulas:

$$h_t = Ux_t + Ws_{t-1}, \tag{4}$$

$$s_t = f(h_t), \tag{5}$$

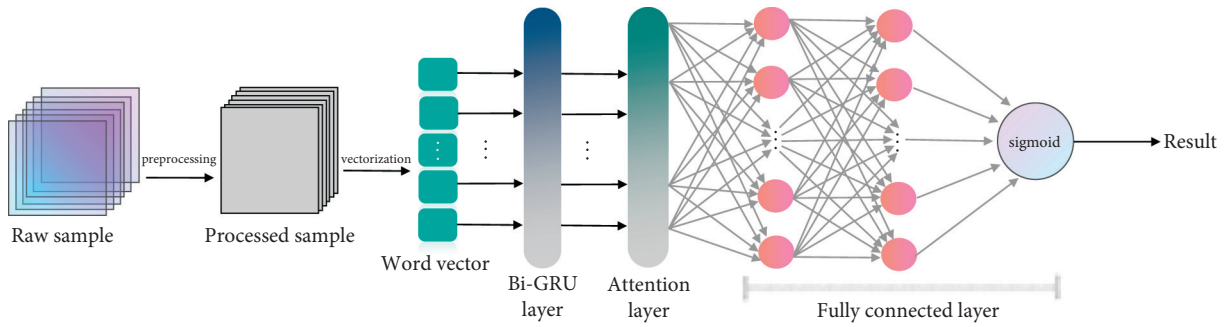

Figure 1: The structure of the detection model.

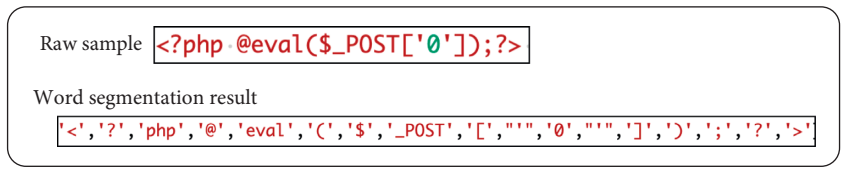

Figure 2: Word segmentation demonstration.

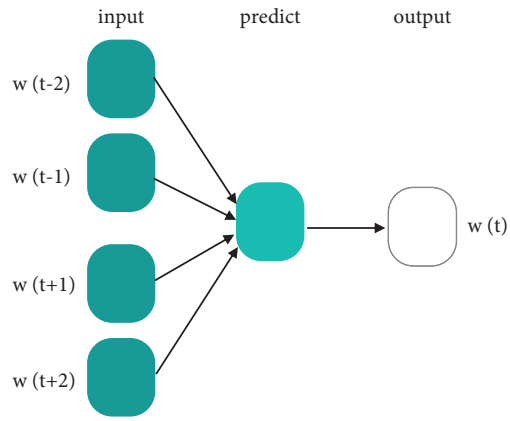

Figure 3: CBOW model.

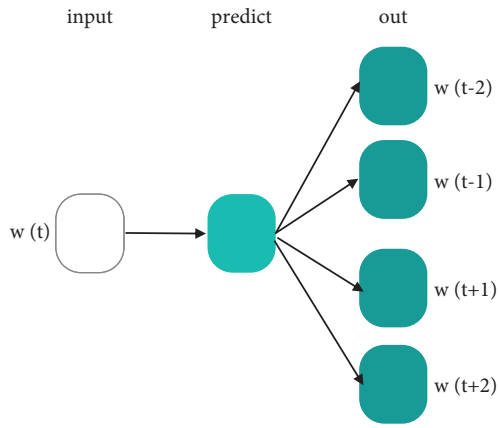

Figure 4: Skip-gram model.

$$o_t = g(Vs_t). \tag{6}$$

Among them, f and g are activation functions, h_t is the hidden state at time t, s_{t-1} is the cell state at time $t-1$, s_t is the cell state at time t, and o_t is the output at time t.

RNN works well when processing sequence problem, but the problem of gradient disappearance or gradient explosion is prone to occur during the training process. LSTM [21] (long short-term memory) is a special RNN for long-term information-dependent learning, and it has been widely used in sequence problems. Compared with the ordinary RNN structural unit containing only a single tanh activation layer, the LSTM cell structural unit is more complex. It contains input gates, output gates, and forget gates. It transmits the long-term memory information of the cell through a channel, effectively reducing the risk of disappearing and exploding gradients. As a variant of LSTM, GRU [22] has a structure as shown in Figure 6.

GRU merges the input gate and forget gate in LSTM into one update gate and merges the unit state with the hidden state. The calculation formula for the output h_t of the hidden node of the GRU unit at time t is shown in the following equation:

$$h_t = (1 - z_t) * h_{t-1} + z_t * h_t^*. \tag{7}$$

Here, z_t represents the value of the update gate, h_{t-1} is the value of the information retained by the previous unit, and h_t^* is the new memory information of the current node. It can be seen from formula (7), there are update gate controls, in which information in the past is forgotten and in which information in the current node is retained. The larger the value of z_t, the more past information will be forgotten and the more current state information will be remembered. The calculation formulas of z_t, h_t^*, and r_t are as shown in the following formulas:

$$z_t = \sigma(W_z X_t + U_z h_{t-1}), \tag{8}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \tag{9}$$

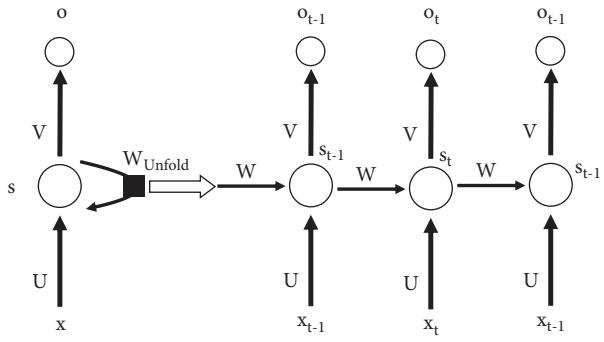

Figure 5: Ordinary RNN structure.

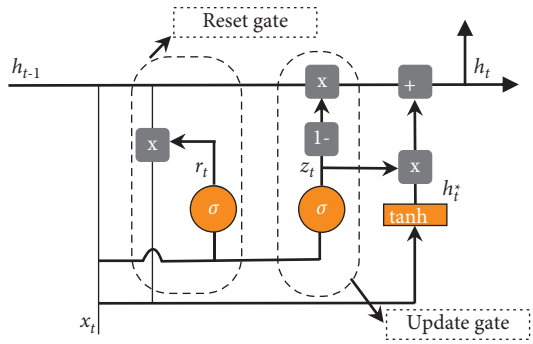

Figure 6: GRU unit structure.

$$h_t^* = \varnothing\left[W_h x_t + r_t * \left(U_r h_{t-1}\right)\right], \tag{10}$$

where σ is the sigmod activation function, $\varnothing$ is the tanh activation function, x_t is the input of the unit at time t, and W and U are the weight matrices that need to be trained. r_t is the reset gate. Compared with LSTM, GRU has a simpler structure and fewer training parameters. The method in this study chooses GRU as the basic unit.

The context in the sample has strong semantic connections. For example, in a WebShell sample, a certain part of the code may be harmless when it appears alone, but it is harmful when it appears in combination. Some normal samples, in order to achieve specific functions, also use part of the code commonly used in WebShell, if the context is not considered globally, it may cause the sample to be falsely reported. In order to better retain the information in the code and improve the accuracy of detection, this article uses a bidirectional GRU structure, as shown in Figure 7.

As shown in the figure, at time t, the output of the hidden node is determined by the two GRUs in opposite directions. The calculation formulas are as follows:

$$\begin{aligned} h_t^{\rightarrow} &= GRU\left(h_{t-1}^{\rightarrow}, x_t\right), \\ h_t^{\leftarrow} &= GRU\left(h_{t-1}^{\leftarrow}, x_t\right), \\ h_t &= W_t h_t^{\rightarrow} + U_t h_t^{\leftarrow} + b_t. \end{aligned} \tag{11}$$

Among them, $h_t^{\rightarrow}$ is the state of forward GRU input, $h_t^{\leftarrow}$ is the state of reverse output, W and U are trainable weight matrices, and b is the bias term.

3.4. Attention Layer. The attention mechanism was first introduced to the task of machine translation [23] and has now become an important concept in the field of neural networks. In the field of AI (artificial intelligence), the attention mechanism has become an important part of the neural network structure and has a wide range of applications in the fields of natural language processing, statistical learning, speech, and computers [24]. In this study, the attention layer uses the "feed-forward" attention mechanism [25], which is a concise attention mechanism that does not require query variables. The formula is described as follows:

$$\begin{aligned} z_t &= a\left(h_t\right) \\ &= \tanh\left(W h_t + b\right), \\ \alpha_t &= \operatorname{softmax}\left(z_t\right)\theta, \\ c &= \sum_{t=1}^{T} \alpha_t h_t. \end{aligned} \tag{12}$$

Among them, h_t is the output state of the hidden layer at time t and a is a trainable function related to h_t. The perceptron model is often used for training, and tanh is the activation function. The attention weight z_t at each moment is obtained through perceptron training. Further, normalized by the softmax function, the sum of each weight is made as 1, the standardized attention weight α_t is obtained, and finally, each attention weight and the corresponding input weighted sum are used to obtain the new hidden layer state. In the new state, important information is strengthened and unimportant information is weakened.

3.5. Fully Connected Layer. As shown in Figure 1, the fully connected layer is a three-layer network that functions as a classifier. After the sample is vectorized, the bidirectional GRU layer and the attention layer are processed to obtain a set of abstract features, which are input to the fully connected network. During training, the network optimizes the loss function and adjusts the weight of the network to achieve the mapping between features and results. Compared with the *sigmoid* and tanh activation functions, the *Relu* activation [26] function used in network training has the advantages of less calculation and the gradient is not easy to disappear. At the same time, the *Relu* function will make the value of some neurons become zero, reducing the mutual dependence between parameters. To a certain extent, the occurrence of over-fitting problems is alleviated. The method in this study uses the *Relu* activation function between the first layer and the second layer and second-third layers of the fully connected layer, and the calculation result of the third layer is input to a *sigmoid* function to obtain the predicted probability.

4. Experiment

4.1. Experimental Environment Setup. The computer hardware used in the experiment is listed in Table 1.

The *Python* language is used to implement the method in this article, and the visual studio code software is used to write the code. In the experiment, the genism [27] library

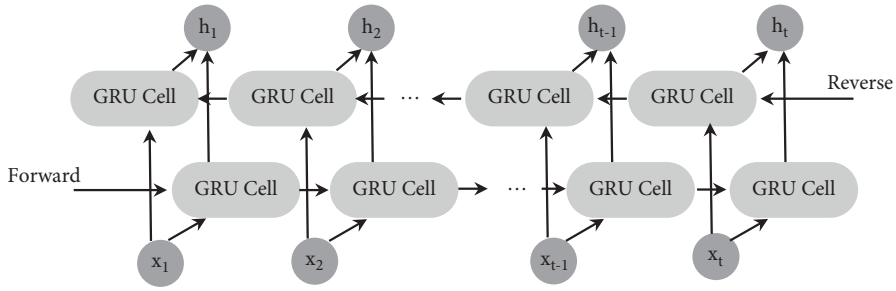

Figure 7: Bidirectional GRU structure.

Table 1: Hardware configuration.

Hardware name	Performance parameters
Processor	i5-10400 2.9 GHz
RAM	16 GB 2666 MHz
Hard disk	512 GB SSD
Graphics card	GeForce GTX 1050Ti 4 GB

Table 2: Software configuration.

Software name	Version	Description
Python	3.8.3	Programming language
Genism	4.0.1	Natural language processing library
Tensorflow	2.4.1	DL library to build and train network models
Keras	2.4.1	DL library to build and train network models
Numpy	1.19.5	Matrix operation library
Matplotlib	3.4.1	Drawing library for graphics drawing

was used to obtain sample word vectors, and tensorflow [28] and keras [29] were used to implement and train the network model. Table 2 lists the software version information and brief description.

4.2. Dataset. The samples in the experiment consist of WebShell samples and normal samples. WebShell samples are collected through GitHub and normal samples are collected from some open-source project files. Table 3 lists the main sample sources.

By deduplicating the collected samples, 4059 WebShell samples are obtained, including 2500 PHP files, 730 JSP files, and 829 ASPX and ASP files. The main sources of normal samples are open-source projects. For example, the normal sources of PHP-type samples are WordPress, joomla, DedeCMS, and drupal. There are a total of 19281 normal samples, including 10,000 PHP files, 4420 JSP files, and 4861 ASPX and ASP files. In the experimental part, we first apply the method to the detection of PHP-type samples. The ratio of the training set to the validation set is 4:1. The training set contains 2000 WebShell samples and 8000 normal samples, and the remaining 500 WebShell samples and 2000 normal files are used as the verification set. For the JSP-type detection experiment, 500 WebShell samples and 4000 normal samples are selected to form the training set, and the remaining 230 WebShell samples and 420 normal samples are selected to form the verification set. In the ASPX- and APS-type detection experiments, 600 WebShell samples and 3,500 normal samples constitute the training set, and the remaining 229 WebShell and 1361 normal samples are used as the validation set.

4.3. Evaluation Criteria. In the experiment, WebShell samples are marked as positive samples and normal files are marked as negative samples.

The detection of WebShell scripts is a two-class problem. In order to objectively evaluate the effect of the proposed method, the accuracy, precision, recall, and F1 value [30] are selected as performance indicators. The confusion matrix is listed in Table 4.

When the true value of the sample is WebShell and the algorithm judgment result is also WebShell, it is a real example (TP). When the true value of the sample is WebShell and the algorithm judges it as a normal file, it is a false negative (FN). When a normal file is accurately identified as a normal file by the algorithm, it is a true negative case (TN). When a normal file is recognized as WebShell, it is counted as a false positive (FP).

The accuracy rate (acc) indicates the proportion of the correctly classified samples in the total sample, the precision rate (pre) indicates the proportion of true cases in all the samples judged to be positive, and the recall rate is the proportion of all positive cases in the sample that are accurately identified. The F1 value is an evaluation index of comprehensive accuracy and recall, which is used to illustrate the overall performance of the algorithm. The relevant calculation formula is as follows:

$$\begin{aligned} acc &= \frac{TP + TN}{TP + TN + FP + FN} * 100\%, \\ \text{pre} &= \frac{TP}{TP + FP} * 100\%, \\ \text{recall} &= \frac{TP}{TP + FN} * 100\%, \\ F1 \text{ value} &= \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} * 100\%. \end{aligned} \tag{13}$$

4.4. Parameter Setup. In the process of representing the sample as a vector, the choice of vector dimension is very

Table 3: Data sources.

Sample type	Project URL
WebShell	https://github.com/tennc/webshell (accessed on 21 March, 2021) https://github.com/ysrc/webshell-sample (accessed on 21 March, 2021) https://github.com/xl7dev/WebShell (accessed on 21 March, 2021) https://github.com/JohnTroony/php-webshells (accessed on 23 March, 2021) https://github.com/tanjiti/webshellSample (accessed on 23 March, 2021) https://github.com/DeEpinGh0st/PHP-bypass-collection (accessed on 23 March, 2021) https://github.com/webshellpub/awsome-webshell (accessed on 23 March, 2021)
Normal	https://github.com/WordPress/WordPress (accessed on 25 March, 2021) http://updatenew.dedecms.com/base-v57/package/DedeCMS-V5.7-UTF8-SP2.tar.bz2 (accessed on 25 March, 2021) https://github.com/joomla/joomla-cms (accessed on 25 March, 2021) https://github.com/drupal/drupal (accessed on 25 March, 2021) https://github.com/SeriaWei/ASP.NET-MVC-CMS (accessed on 25 March, 2021) https://github.com/sanalog/Sanalog-1.5.2-Web (accessed on 25 March, 2021) https://github.com/apache/tomcat (accessed on 25 March, 2021)

Table 4: Confusion matrix.

Actual value	Predicted value	
	Positive	Negative
Positive	TP	FN
Negative	FP	TN

important. Low vector dimensionality can reduce the amount of calculation, but the ability to express sample semantics is weak; high vector dimensionality may lead to excessive calculation. In order to choose the appropriate dimension, we successively counted the influence of 20 to 300 dimensional vectors on the experiment. An ordinary GRU network is used as the basic algorithm, 32 network units are set up, the number of training rounds is set to 5, and the accuracy of each dimension experiment is successively counted. The results are shown in Figure 8.

When the dimension is 20, the accuracy of the algorithm exceeds 93%. With the increase of the vector dimension, the accuracy of the algorithm gradually improves. When the dimension reaches 200, the accuracy of the benchmark algorithm reaches 98.16%. When the dimension exceeds 200, the accuracy improves slightly. But the calculation time is growing rapidly. Therefore, we determined the word vector dimension to be 200 dimensions.

After preprocessing the word segmentation, the number of sample words is different. It is necessary to select the appropriate number of each sample word. We counted and analyzed the distribution of the number of words in the PHP-type samples. The result is shown in Figure 9

Among the 12,500 PHP samples, the number of words in the sample is mainly distributed between 1 and 600. The number of samples with the number of words within 200 reached 10,269, accounting for 82.15%. The number of words is within 400 and the sample number is 11397, accounting for 91.18%. The number of words is within 600 and the sample number is 11,691, accounting for 93.53%. Therefore, we fixed the number of words to 800 while taking into account the preservation of sample information and the amount of calculation. When the number of words in the sample is greater than 800, we directly select the first 800 words in the sample and directly discard the remaining. When the number of words in the sample is less than 800, we fill in the sample content repeatedly until the number of sample words reaches 800. Through the above processing, the number of sample words is set to 800.

The collected samples are preprocessed according to the method shown in Section 3.1 to obtain a series of words, and the Word2Vec model in the genism library is used for word vector training. When training the Word2Vec model, the word vector dimension is set to 200 dimensions, words with no less than 4 occurrences are selected, and the window size is set to 5. The model is trained for a total of 5 rounds to obtain the word vector model used in the experiment.

During the experiment, the number of cell units used by the bidirectional GRU layer was set to 32, and the dropout ratio was set to 0.1. The nodes of the fully connected layer are 32, 16, and 1 in sequence. A dropout layer is added between layers, and the dropout probability is set to 0.1. The network training uses the Adam optimization function [31], the loss function uses binary cross entropy, and there is a total of 40 rounds of training.

4.5. Experimental Results and Analysis. The PHP-type WebShell is the most popular, and the obfuscation techniques are multiple and diverse. Researchers have done the most research on PHP-type WebShells. We first experiment with PHP-type WebShell and then applied the method to identify the JSP-, APSX-, and ASP-type WebShell. For PHP-type detection, we conducted six experiments, using LSTM network, bidirectional LSTM network, bidirectional LSTM + attention network, GRU network, bidirectional GRU network, and bidirectional GRU + attention network. In each set of experiments, the network structure of the fully connected classification layer and the hyperparameter settings are consistent. The experimental results are listed in Table 5.

By inputting the word vector obtained after preprocessing, the accuracy of the LSTM network and the GRU network both exceeded 98%, indicating that the extracted word vector as the original feature is feasible for WebShell detection. When using a GRU network, the algorithm's recall rate is 95.8%, which means that there will only 5 false

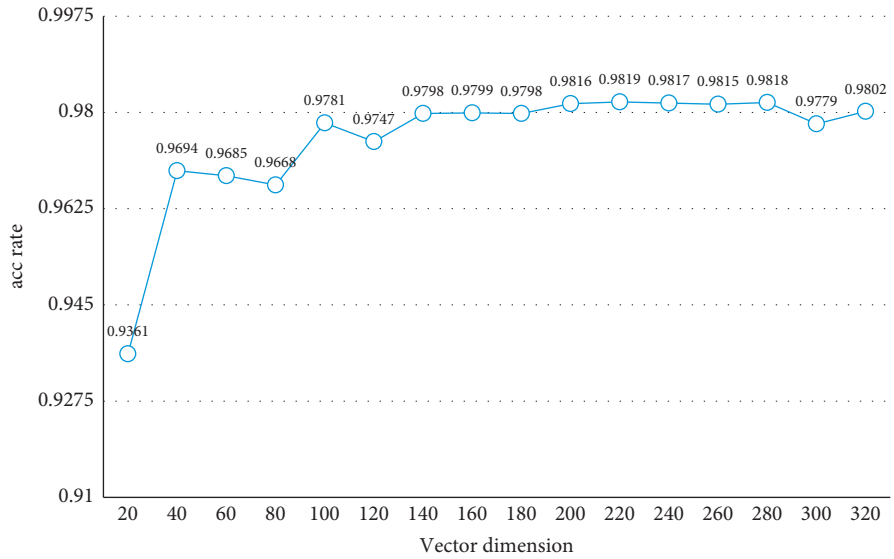

Figure 8: Relationship between accuracy and vector dimensions.

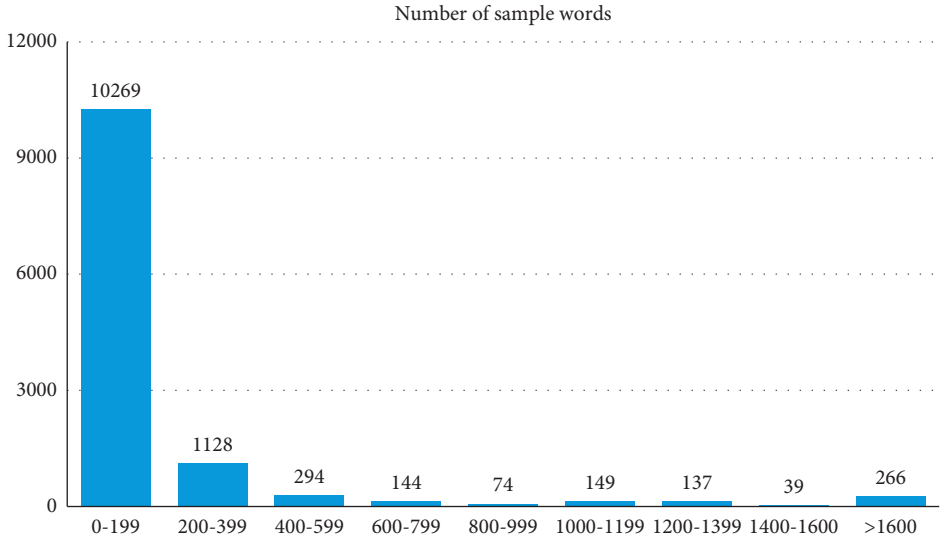

Figure 9: Word count analysis.

negatives for 100 WebShells on average; while the LSTM network has a recall rate of 92.8%, which is significantly lower than the GRU network. The bidirectional recurrent neural network extracts feature from two opposite directions, which can effectively improve the performance of the algorithm, which is verified in the experiment. Compared with the LSTM network, the bidirectional LSTM network has improved in four performance indicators. Among them, the recall rate is increased by 3.8%, and the algorithm performance is improved significantly. Compared with the GRU network, the bidirectional GRU network has a slight decrease in accuracy, but it has improved accuracy, recall, and F1 performance. It shows that the bidirectional RNN can extract richer characteristic information compared with the unidirectional RNN, which can improve the performance of the algorithm to varying degrees. After adding the attention mechanism layer to the bidirectional GRU network and the bidirectional LSTM network, the accuracy, recall, and F1 value are further improved; both the accuracy rate exceeds 99.3%, which means that there are an average of 1000 samples and only 7 misjudgments. The bidirectional GRU network combined attention mechanism network has achieved the best performance among the six methods in terms of accuracy, recall, and F1 value. Among them, the recall rate is 98.6%, the accuracy rate is 99.36%, and the F1 value is 98.4%.

We compared the methods proposed in references [4, 6, 32, 33], and the performance indicators of each method are listed in Table 6.

Reference [4] uses the combination of Naive Bayes and opcode, the accuracy, precision, and F1 value are all over 97%, and the recall rate is 96.8%. Reference [6] combines

Table 5: PHP-type WebShell detection results.

Method	acc (%)	pre (%)	recall (%)	F1 value (%)
LSTM	98.20	98.10	92.80	95.38
GRU	98.88	98.56	95.80	97.16
Bidirectional LSTM	99.08	**98.77**	96.60	97.67
Bidirectional GRU	99.12	98.19	97.40	97.79
Bidirectional LSTM + attention	99.32	98.59	98.00	98.29
Bidirectional GRU + attention	**99.36**	98.21	**98.60**	**98.40**

The bold text means the best results in each performance comparison.

Table 6: Comparison of PHP-type WebShell detection performance.

Method source	acc (%)	pre (%)	recall (%)	F1 value (%)
Reference [4]	97.40	97.20	96.80	97.00
Reference [6]	99.23	97.65	97.92	97.78
Reference [32]	94.4	93.2	96.8	94.97
Reference [33]	98.91	**99.25**	95.73	97.46
This study	**99.36**	98.21	**98.60**	**98.40**

FastText, static features, and random forest algorithms, and the accuracy is as high as 99.23%, and the other three types of performance indicators all exceed 97.5%. Reference [32] is a deep learning algorithm that uses a multi-layer perceptron model to convert the original code into bytecode and input it into the network. This scheme has achieved good results, but each performance needs to be further improved. Reference [33] is an ensemble learning algorithm that combines logistic regression algorithm, multi-layer perceptron, and random forest algorithm, using multiple weak classifiers to integrate into a strong classifier, and it achieved 99.25% accuracy on the experimental dataset. Compared with the above four schemes, the method in this study has improved performance in varying degrees except that the accuracy is lower than the scheme proposed in reference [33]. Compared with the multi-layer perceptron method in reference [32], the accuracy of the method proposed in this study is improved by nearly 5%, the accuracy is improved by more than 6%, and the recall rate and F1 value are also greatly improved. Although the accuracy of the method in reference [33] is higher than that in this article, the accuracy of our method is improved by 0.45%, recall rate is improved by 2.87%, and F1 value is improved by 0.94%. For further analysis, we found the total number of WebShell samples in the dataset used in reference [33] is only 571, while the number of normal samples is 5379, the ratio is close to 1:10, the division ratio of training set and validation set is 7:3, and the number of WebShells in the verification set is only 172. In order to fully evaluate the performance of the algorithm, it is necessary to run the algorithm on a larger dataset.

Further, we experimented with the detection performance of the bidirectional GRU network combination attention mechanism model on JSP-, ASPX-, and ASP-type WebShell, and ASPX and ASP types were tested together as the same type. The experimental results are listed in Table 7:

The detection performance of JSP, ASP, and ASPX types of WebShell is greatly improved compared with the PHP-type detection performance. Analyzing the reasons, on the one hand, we found that in the field of web development, PHP language is used more frequently than JSP, ASPX, and ASP, which causes attackers to be more inclined to study the escape technology of PHP-type WebShell; on the other hand, the grammatical structure of PHP language is more flexible. The attackers can use more flexible escape methods, and it is more difficult to detect the PHP-type WebShell. In the detection of JSP-type WebShell, 228 were detected out of 230 WebShells in the verification set, with a recall rate of 99.13%. For ASP and ASPX types of WebShell detection, only one WebShell sample in the verification set has false positives and a common sample has false positives, with an accuracy rate of 99.87%.

As listed in Table 8, the detection method for JSP-type WebShell has superior performance compared with the proposed scheme.

Reference [34] first uses an abstract syntax tree to obtain sample features and then uses BP neural network for training and classification. Reference [35] is a session-based detection scheme. Reference [36] uses the TF-IDF algorithm to extract features and then uses XGBoost for training and classification. Reference [37] combines abstract syntax tree and XGboost algorithm. Compared with the other schemes, the method in this study is superior in accuracy, recall, and F1 index. In terms of accuracy, it is slightly insufficient compared to the other three schemes.

The detection research of ASPX- and ASP-type WebShell is too few to compare.

4.6. Application Scenarios. The WebShell detection system has many application scenarios. For example, the detection model can be integrated into an existing network security detection system to develop an independent WebShell detection system. The system initial file hash table is established when the detection system is initialized. Later, by monitoring the changes of the files, when the hash of the specified type of file changes or when a specified type of file is created, the file is sent to the detection system and then the system will detect it. When an attacker uploads a new file or inserts malicious code into an existing file, the detection is triggered. If an abnormality is detected, the system alerts the network

Table 7: JSP-, ASP-, and ASPX-type WebShell detection results.

Language type	acc (%)	pre (%)	recall (%)	F1 value (%)
JSP	99.23	98.70	99.13	98.91
ASP and ASPX	99.87	99.56	99.56	99.56

Table 8: Comparison of JSP-type WebShell detection performance.

Method source	acc (%)	pre (%)	recall (%)	F1 value (%)
Reference [34]	Not mentioned	99.10	98.7	98.90
Reference [35]	95.97%	96.15	90.36	93.17
Reference [36]	97.09%	**99.38**	96.76	98.05
Reference [37]	98.73%	98.84	95.69	97.23
This study	**99.23%**	98.70	**99.13**	**98.91**

The bold text means the best results in each performance comparison.

administrator and interrupts access to the file. At the same time, the abnormal file is collected and filed for further processing by the administrator.

5. Conclusions

This study proposes a WebShell detection method based on a bidirectional GRU network and attention mechanism. By analyzing the relationship between the dimension of the word vector and the detection accuracy, the distribution of the number of words, the appropriate dimension of the word vector, and the number of words are determined. Through detailed experiments, the feasibility and effectiveness of the method in this study are proved. The method in this study does not involve the conversion of intermediate codes and can effectively identify WebShells written in multiple languages. Compared with the existing methods, the method in this study has achieved better performance indicators such as accuracy, recall, and F1 value. But the accuracy and recall rate can be further improved. In future research, on the one hand, we will consider combining static characteristics and traffic characteristics to further improve accuracy rate and recall rate. But on the other, how to improve the interpretability of the model is also a problem to be studied.

Data Availability

The data used in this study are available in Table 3. Researchers can download it from the listed websites according to their needs. They are open source.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank everyone who helped during the research and preparation of the article. This research was funded by the National Natural Science Foundation of China (no. 61662004).

References

[1] National Internet Emergency Center, "China Internet Cybersecurity Report," 2020, https://www.cert.org.cn/publish/main/upload/File/2020%20Annual%20Report.pdf.

[2] L Qian, Z. Zhu, J. Hu, and S. Liu, "Research of SQL Injection Attack and Prevention Technology," in *Proceedings of the 2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF)*, pp. 303–306, Harbin, January 2015.

[3] A. Begum, M. M. Hassan, T. Bhuiyan, and M. H. Sharif, "RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh," in *Proceedings of the 2016 International Workshop on Computational Intelligence (IWCI)*, pp. 21–25, Dhaka, Bangladesh, December 2016.

[4] Y. Guo, H. Marco-Gisbert, and P. Keir, "Mitigating webshell attacks through machine learning techniques," *Future Internet*, vol. 12, no. 1, p. 12, 2020.

[5] Y. Song, Y. Shen, and G. Zhang, "The New INLINE Hook Technology Combination of Hard-Code Technology and Independent Code Injection," in *Proceedings of the 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 521–525, Beijing, China, August 2016.

[6] F. Yong, Q. Yaoyao, L. Liang, and H. Cheng, "Detecting Webshell Based on Random Forest with FastText," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence ICCAI 2018*, pp. 52–56, New York, NY, USA, March.2018.

[7] J. Armand, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of Tricks for Efficient Text Classification," 2014, https://arxiv.org/abs/1607.01759.

[8] PECL, "PECL::package::vld," 2021.

[9] H. Zhang, M. Liu, Z. Yue, Z. Xue, Y. Shi, and X. He, "A PHP and JSP Web Shell Detection System with Text Processing Based on Machine Learning," in *Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, December 2020.

[10] T. chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, New York, NY, USA, August 2016.

[11] A. Onan, "Classifier and feature set ensembles for web page classification," *Journal of Information Science*, vol. 42, no. 2, pp. 150–165.

[12] Y. Wang, H. Pan, T. Jing, and S. Yaxi, "Research and implementation on WebShell comprehensive detection and traceability technology based on high-speed network," *Netinfo Security*, vol. 21, no. 1, pp. 65–71, 2021.

[13] A. Onan, "Sentiment analysis on product reviews based on weighted word embeddings and deep neural networks," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 23, Article ID e5909, 2021.

[14] A. Onan, S. Korukoğlu, and H. Bulut, "Ensemble of keyword extraction methods and classifiers in text classification," *Expert Systems with Applications*, vol. 57, pp. 232–247, 2016.

[15] A. Baccouche, S. Ahmed, D. Sierra-Sosa, and A. Elmaghraby, "Malicious text identification: deep learning from public comments and emails," *Information (Switzerland)*, vol. 11, no. 312, 2020.

[16] Z. Xian and G. Gan, "Research on webshell detection based on BPTT algorithm," *Computer & Digital Engineering*, vol. 48, no. 2, pp. 372–377, 2020.

[17] L. Zhou, C. Wang, and S. H. I. Yin, "Research of Webshell detection based on RNN[J]," *Computer Engineering and Applications*, vol. 56, no. 14, pp. 88–92, 2020.

[18] A. Onan and M. A. Tocoglu, "A term weighted neural language model and stacked bidirectional LSTM based framework for sarcasm identification," *IEEE Access*, vol. 9, pp. 7701–7722, 2021.

[19] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," 2014, https://arxiv.org/abs/1409.0473.

[20] T. Mikolov, G. Corrado, C. Kai, and D. Jeffrey, "Efficient Estimation of Word Representations in Vector Space," in *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, Scottsdale, AZ, USA, January 2013.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] K. Cho, M. B. Van, C. GULcehre et al., "Learning phrase representations using RNN encoder- decoder for statistical machine translation," in *Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, ACL, Stroudsburg, PA, USA, October 2014.

[23] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," 2014, https://arxiv.org/abs/1409.0473.

[24] S. Chaudhari, V. Mithal, P. Gungor, and R. Ramanath, "An Attentive Survey of Attention Mod-Els," 2019, https://arxiv.org/abs/1904.02874.

[25] C. Raffel, P. Daniel, and W. Ellis, "Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Prob-Lems," 2015, https://arxiv.org/abs/1512.08756.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 315–323, Fort Lauderdale, FL, USA, January 2011.

[27] R. Rehurek and S. Petr, "Software framework for topic modelling with large corpora," in *Proceedings of the LREC 2010 Workshop New Challenges for NLP Frameworks*, pp. 46–50, University of Malta, Valletta, Malta, 2010.

[28] Zenodo, "Tensor," 2019, https://www.tensorflow.org/.

[29] N. Aakash, P. Sayak, and M. -R. Margaret, "Keras," 2015, http://keras.io/.

[30] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," *Lecture Notes in Computer Science*, Santiago de Compostela, Spain, 2005.

[31] D. Kingma and B. Jimmy, "Adam: a method for stochastic optimization," 2017, https://arxiv.org/abs/1412.6980v9.

[32] Z. Wang, J. Yang, M. Dai, R. Xu, and X. Liang, "A method of detecting webshell based on multi-layer perception," *Academic Journal of Computing & Information Science*, vol. 2, pp. 81–91, 2019.

[33] Z. Ai, N. Luktarhan, A. Zhou, and D. Lv, "WebShell attack detection based on a deep super learner," *Symmetry*, vol. 12, no. 9, p. 1406, 2020.

[34] H. Zhang, "A method for WebShell detection based on semantics analysis and neural network," *Cyberspace Security*, vol. 10, no. 2, pp. 17–23, 2019.

[35] Y. Wu, Y. Sun, C. Huang, P. Jia, and L. Liu, "Session-Based Webshell Detection Using Machine Learning in Web Logs," *Security and Communication Networks*, vol. 2019, Article ID 3093809, 2019.

[36] R.-J Zhao, Y. Shi, Z. Han, J. Long, and Z. Xue, "Webshell file detection method based on TF-IDF," *CONPUTER SCIENCE*, vol. 47, no. 2, pp. 373–377, 2020.

[37] M. A. O. Yu-qi, Y. Shi, and Z. Xue, "jsp_webshell Detection Method based on AST and XGBoost[J]," *Communications TechnoloGy*, vol. 53, no. 10, pp. 2543–2549, 2020.