A Method of Detecting Webshell Based on Multi-layer Perception

Zihao Wang[1], Jingjing Yang[1], Mengjie Dai[1], Ruoyu Xu[2], and Xiujuan Liang[1]

[1] School of Cybersecurity, Chengdu University of Information Technology, Shuangliu District, Chengdu, Sichuan Province, 610225, China
[2] University of Toronto Mississauga Campus, University of Toronto, Mississauga, Ontario, L5L 1C6, Canada

ABSTRACT. *WebShell is a commonly used tool for network intrusion. It has the characteristics of high, concealment, great harm and so on. The existing WebShell detection method has higher detection accuracy when detecting a known WebShell, but the accuracy of detection is low when it faces complex and flexible unknown and variant WebShell. To solve this problem, a WebShell detection method based on Multi-Layer Perceptron (MLP) neural network is proposed. Firstly, the sample source code is converted into a sample byte code by a compiler tool, and then the sample byte code is divided into byte code sequences using Bi-Gram. Secondly, TF-IDF is used to calculate the word frequency matrix, and on this basis, the feature matrix of trained sample set is selected. Finally, the detection model is obtained through multi-layer neural network training. The experimental results indicate that compared with the existing methods, the constructed detection model can significantly improve the detection accuracy, accuracy, and recall rate, and the detection, accuracy of unknown and variant samples can reach over 90%.*

KEYWORDS: *multi-layer perception, WebShell, machine learning, cyber security, intrusion detection*

1. Introduction

With the continuous development of Internet technology, B/S-based Web applications have been widely used in e-commerce, online forums, government administration, and other online platforms, playing an important economic, cultural and political role, and their security has been directly related. To property security, social stability, etc. However, due to limitations in technology and security awareness, existing web application platforms have various types of system vulnerabilities or application vulnerabilities, providing an opportunity for attackers to illegally enter web hosts. To achieve long-term control of the web host, an attacker typically uploads a malicious script to the web server after entering the host and hides it in the normal script folder. Through these malicious scripts, an attacker can create a backdoor to achieve long-term control of the web host. This type of malicious script that controls a web host through a web application is often referred to as WebShell. The 2016 China Internet Cyber Security Report [1] issued by the National Internet Emergency Center (CNCERT) pointed out that in 2016, CNCERT monitored 82,072 websites in China and was implanted into WebShell, of which commercial websites accounted for 62.3%. 4.8%, government websites accounted for 2.9%. WebShell is usually very concealed, and it is difficult to detect it in time. At present, traditional anti-virus software mainly uses the method of signature matching to detect WebShell. This method extracts its text features as malicious signatures by analyzing known WebShell code samples and forms a malicious signature database. This method can detect more than 90% of the known types of WebShell, but for unknown or variants of WebShell, the accuracy is even less than 60%. Through the highly concealed WebShell, attackers can control the Web host for a long time and use it for information theft, commercial extortion, botnet, and other illegal activities, and the degree of harm depends largely on the length of time the Web host is controlled. Therefore, it is very important to study methods that can detect various types of WebShell, such as known, unknown and variants.

2. Related research

Traditional WebShell detection methods fall into three main categories: static analysis, dynamic analysis, and diary analysis. Static analysis is performed before the script runs. The traditional detection method mainly detects malicious scripts by matching the signature strings in the file (such as common malicious code blocks, high-risk function names eval, system, etc.). Such methods based on signature matching are generally implemented by regular expression matching.

However, since the regular expression is essentially a finite state automaton, it is impossible to completely define the behavior characteristics and completely cover the risk model. Therefore, there is a bottleneck that cannot be overcome in reducing the false negative rate and false positive rate. In fact, Hansen et al. [2] have theoretically proved that there must be false negatives and false positives in the matching method based on regular expressions. Dynamic analysis is performed while the script is running. The traditional detection method mainly detects malicious scripts by analyzing the dynamic characteristics of the script execution process, such as analyzing eval execution context, file read and write operations, network traffic, and other dynamic behaviors. For example, Wrench et al. [3] use Web behavior based on behavioral similarity to dynamically detect WebShell, which can effectively detect malicious behaviors in active state. Du Haizhang et al [4] detect PHP code compilation process based on PHP extension and implement WebShell. Real-time dynamic detection; Ma Yanfa et al [5] use Web traffic analysis to detect WebShell on the WAF (Web Application Firewall) side, avoiding the drawbacks of traditional detection methods that require separate detection modules on all Web servers. The advantage of this type of method is that it can effectively identify various unknown samples, deformed samples, encrypted samples, etc., but there are also problems of high false positive rate. The diary analysis is performed after the script is run. The traditional detection method mainly detects the malicious script by analyzing the web server log file, such as using the page request feature, the access statistical feature and the page association feature, etc. [6]. This detection method is effective when the amount of log data is large, but there is a problem of high false positive rate. Due to the shortcomings of traditional detection methods, researchers began to look for detection solutions from a new perspective. At present, the more advanced detection methods include text-based statistical features and text-based semantic features.

The method based on text statistical characteristics is mainly based on statistical theory, trying to use various statistics as features to detect malicious scripts, such as open source program NeoPI [7] using information entropy, coincidence index, longest word, compression ratio and other features detection of file content. WebShell. Hu Jiankang et al. [8] extracted content attributes (number of words, maximum word length, etc.), basic attributes (number of evil function calls, maximum length of function parameters, etc.) and advanced attributes (file operations, database operations) as classification features, using The decision tree C4.5 classifier builds the detection model and is supplemented by the integrated learning algorithm Boosting to implement WebShell detection.

Methods based on text semantics attempt to discover malicious code behavior at the semantic level using natural language processing (NLP) techniques. For example, Ye Fei et al. [9] extracted the structural and content characteristics (page title, meta-information, keywords) of the script as classification features, and used the support vector machine (SVM) as the classification algorithm to construct the detection model; Deng et al. [10] used Lexical analysis is used to extract features, and WebShell is identified and detected from the perspective of grammar. Yi Nan et al [11] proposed a semantic-based malicious code detection method to describe and evaluate the behavior of files from a semantic perspective.

The method first constructs an abstract syntax tree (AST) according to the code file, extracts the smudge subtree from the complete tree through the node risk assessment table, and matches it with the malicious behavior feature map to obtain the risk value, and finally determines whether it is malicious through the threshold. Code file. The advantage of this kind of method is that the implementation is relatively simple, and the detection accuracy of the known malicious code is high, but the disadvantages are obvious, such as the detection effect on the unknown sample and the variant sample is poor.

It can be seen that the existing WebShell detection algorithm is more researched at the source text level, and is susceptible to WebShell escape methods such as code annotation and code obfuscation. In addition, the existing detection models mostly use the traditional classifier algorithm. The generalization ability of these classifier algorithms is weak, which leads to a sharp drop in detection performance of the detection model in the face of unknown mode samples, or even no detection at all.

3. WebShell detection method based on multi-layer neural network

In order to solve the above problems, this paper proposes a WebShell detection method based on multi-layer neural network. Compared with the traditional methods, it stays at the source level. This paper innovatively studies the source code compilation results, which can effectively avoid the impact of WebShell escape methods such as code annotation and code obfuscation. In addition, in the aspect of sample feature selection, the traditional method is mostly based on character feature code, so it can only cover a single code statement. The method uses bytecode sequence as sample feature, which effectively utilizes the context information of the code and greatly enhances the model. Detection accuracy.

In order to further improve the detection accuracy, the algorithm also applies the neural network classification algorithm-multilayer perceptron (MLP) to the detection model. Compared with the traditional classification algorithm, MLP has stronger nonlinear fitting ability and powerful generalization ability, which can effectively detect unknown samples that are difficult to handle by traditional methods.

According to statistics, in the known WebShell, WebShell is written in the PHP language. Therefore, this article focuses on the detection method of PHP type WebShell. This section will first analyze the PHP code compilation execution flow, and design the PHP code compilation results - byte code extraction method, then will introduce how to use the feature engineering method to extract the bytecode sequence as a sample feature, and finally how to use MLP The algorithm trains the WebShell detection model.

3.1 Bytecode acquisition

PHP is an interpreted language, and its code execution flow can be divided into Lexical Analysis stage, Syntax Analysis stage, OpCodes compilation stage and code execution stage. The execution flow chart is shown in the solid line in Figure 1. In the lexical analysis phase, Lexer reads the source character sequence in turn and splits it into Token sequences according to PHP syntax rules. In the parsing phase, Parser reads in the Token sequence and performs a syntax check to generate an Abstract Syntax Tree (AST). During the bytecode compilation phase, the PHP virtual machine Zend reads the abstract syntax tree and translates the operator nodes on it into the corresponding bytecode. In the code execution phase, the PHP virtual machine Zend loads the corresponding module according to the code requirements, initializes the running environment, and finally executes the bytecode and outputs the result.

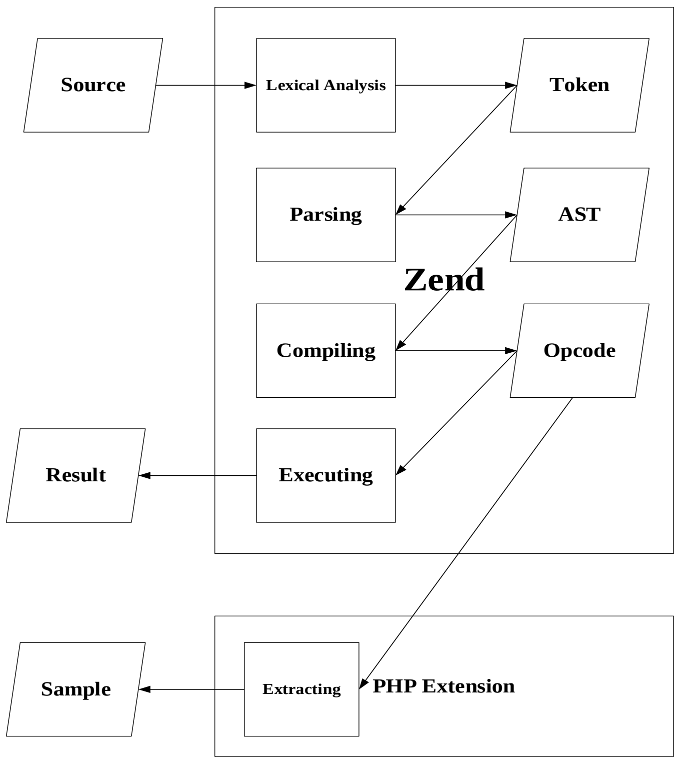

Figure. 1 PHP source code byte extraction process

From the process point of view, if you want to get the compiled bytecode of the PHP code, you need to add the corresponding bytecode output function to the PHP virtual machine Zend during the Executing phase. At this point, there are two ways to do this: modify the source code and add plugins. Modifying the source code method requires re-compilation of the PHP runtime environment, so it is rarely used in practical applications.

Adding a plug-in method uses a unified plug-in framework, so you only need to compile the plug-in itself separately, and you can easily add the function by modifying the configuration file. In addition, the add-on plug-in method can also distribute deployments in the PHP runtime environment of different systems, which is extremely convenient.

Based on the above considerations, this article uses the plug-in method to implement the bytecode centralized output function. After adding the bytecode extraction function, the original PHP virtual machine Zend will additionally output the bytecode samples during the execution to bytecode compilation, as shown in the solid box section of the dotted line in Figure 1.

3.2 Feature Extraction

Before using the bytecode sample set to train the detection model, feature extraction of the bytecode samples is required to determine the features that best distinguish between normal samples and WebShell samples.

In order to utilize the context information of the bytecode, this paper uses the binary syntax model (BiGram) to divide the bytecode sample set, which divides the adjacent two bytecodes into one phrase and counts the phrase in the word. The number of occurrences in the section code sample, which represents the bytecode sample. That is after the phrase segmentation is completed, each bytecode sample is represented as a word frequency vector.

The bytecode sample set segmented in the above manner may contain a large number of various bytecode phrases, that is, the word frequency vector characterizing the bytecode samples may have a very high dimension, which will bring huge computational pressure to subsequent model training. Therefore, it is necessary to filter the bytecode phrases and filter the parts in which the classification of the samples is not helpful. In this paper, the word frequency threshold method is used to filter the features with weak classification ability, that is, the frequency of occurrence of each byte code phrase in the entire sample set is first counted, then the phrase whose frequency is less than 30% is removed, and the remaining words are used to form the byte code. The phrase dictionary, and according to the description of each sample feature, the word frequency matrix of the bytecode sample set is obtained. After obtaining the word frequency matrix of the bytecode sample set, this paper uses the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm to perform word frequency analysis to obtain the importance of each sample in the sample set. The main idea of TF-IDF is that if a word or phrase has a high TF (Term Frequency) in a document and it rarely appears in other documents, the word or phrase is considered to have a good category. Differentiating ability, suitable for classification. Where TF refers to the number of times a word or phrase appears in a document, and IDF (Inverse Document Frequency) is a measure of the word or weight of a phrase. If a word has a low TF in multiple documents, but it appears frequently in a document, the word IDF value is larger. In contrast, the more common a word, the lower the IDF. Multiplying the TF value of a word by the IDF value yields the TF-IDF value. When this value is larger, it indicates that the word is more important in the document, and the more representative the document.

Using the TF-IDF algorithm to process the sample set word frequency matrix, a bytecode feature matrix that reflects the importance of each sample is obtained. Each column vector in the matrix corresponds to a bytecode sample.

A flow chart for feature extraction of bytecode samples is shown in Figure 2.

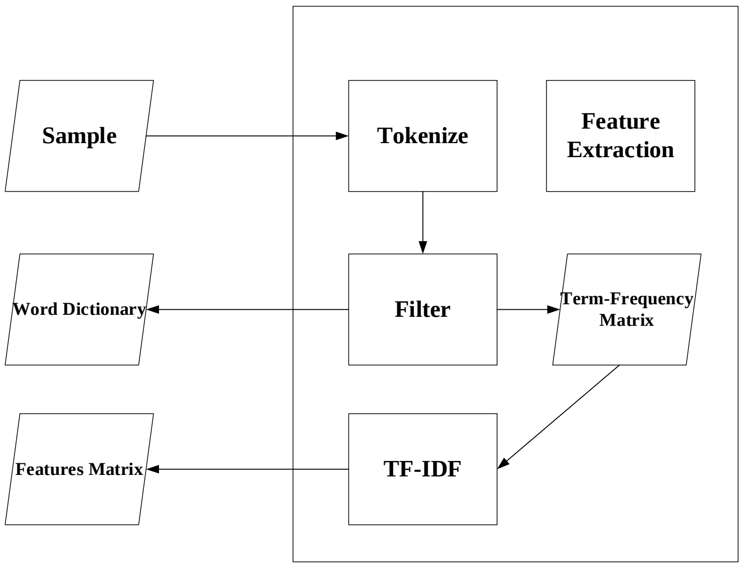

Figure. 2 Bytecode feature extraction process

3.3 Model training

After the feature extraction of the bytecode sample set is completed, the feature matrix can be used as an input, the labeling result is used as an expected output, and the classifier is used for training.

In the selection of classifiers, existing WebShell detection models generally use traditional classification algorithms to train samples, such as linear regression, decision trees, and naive Bayes classifiers. These classification algorithms have the common disadvantages: it is difficult to fit complex nonlinear relationships, and the generalization ability is weak, that is, it is difficult or even impossible to process samples of unknown patterns. In order to solve such problems, this paper uses the Multi-layer Perceptron (MLP) neural network algorithm to learn samples.

The MLP neural network is a forward-structured artificial neural network that uses a backpropagation algorithm for training. The network consists of an input layer, a hidden layer, and an output layer. The input layer is used to receive input data; the hidden layer can have multiple layers for learning data and storing training results, and the output layer is used for outputting results. The nodes of each layer are all connected to the next layer. Except for the input node, all other nodes multiply the input by its own weighting factor w, plus the offset b, and then the result of its own nonlinear activation function produces an output. The activation functions used by each layer are different, such as the middle layer node using the Sigmoid function as the activation function:

$$f(x)=\frac{1}{1+e^{-x}}$$

The output layer node uses the Softmax function as the activation function:

$$f(x_i)=\frac{e^{x_i}}{\sum_{n=1}^{N} e^{x_n}}$$

Where x_i is the input from the previous layer and N is the total number of nodes in the previous layer?

MLP is a generalization of perceptrons that overcomes the weakness that perceptrons cannot identify linear indivisible data. MLP has been mathematically proven to fit nonlinear relationships of arbitrary complexity with powerful generalization capabilities. Due to the above advantages, the detection performance of the WebShell detection model will be greatly improved after the introduction of the MLP neural network.

The MLP neural network constructed in this paper contains two layers of hidden layers.

The first hidden layer contains 5 nodes, and the second hidden layer contains 2 nodes. The L-BFGS algorithm is used to adjust the weight of each node. The network structure is shown in Figure 3.

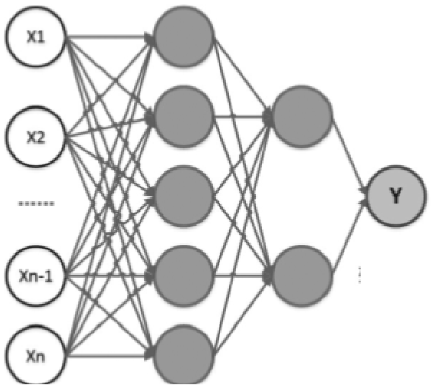

Figure. 3 MLP neural network structure

3.4 Sample testing

Once the model is trained, it can be used to detect unknown samples. The pre-processing steps are required before the source code samples are entered into the model. That is, the bytecode is extracted first to obtain the bytecode samples, and then the phrase segmentation is performed to obtain the phrase set. Finally, the bytecode phrase dictionary outputted by the model is used to filter and collect the phrase set, and the bytecode corresponding to the source code sample is formed. Feature Vector. The feature vector is input into the detection model and based on the result, it can be judged whether the input unknown sample is malicious.

The sample testing process is shown in Figure 4.

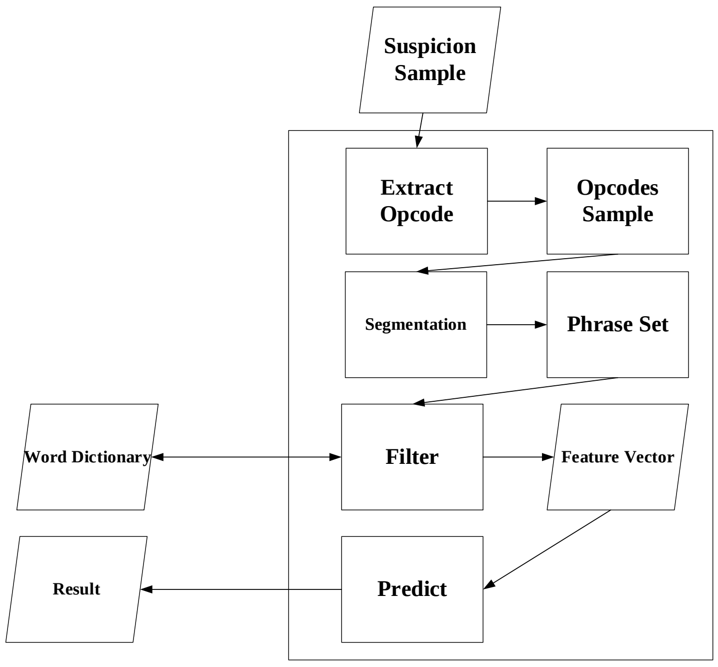

Figure. 4 Detection process for unknown samples

4. Experimental results and analysis

In this paper, the effectiveness of the proposed method is verified by experimental comparison. The experimental environment is: scikit-learn 0.19; the training sample set size is 6,000, of which 3000 WebShell negative samples are from GitHub's WebShell collection project, and another 3,000 positive samples are from popular open source projects such as PHP Wind and PHP CMS.

In order to accurately compare the performance of various detection methods, this paper uses the confusion matrix as the quantitative evaluation standard. The confusion matrix uses three indicators to measure the performance of the detection method: recall (Recall), precision (Precision), and accuracy (Accuracy). Among them, the recall rate indicates the proportion of the positive sample portion of the test set that is correctly predicted, the accuracy indicates the proportion of the test sample whose prediction result is positive, and the accuracy rate indicates the proportion of the entire sample set that is predicted correctly.

In the verification of the performance of each test model, this paper uses the cross-validation method. The source code sample set is divided into several subsets, some of which are used as training sets to train each detection model, and the remaining part is used as a validation set to perform performance verification on the training result model. In order to reduce the measurement error during the experimental test, all the test models were cross-validated 10 times, and the average value of each performance index was taken as the final performance result.

Table 1 compares three types of WebShell detection models: a naive Bayesian-based detection model at the text level, a naive Bayes-based detection model for the bytecode layer, and an MLP-based detection model at the bytecode level. It can be seen that the detection model based on Naïve Bayes is also improved when the detection object is changed from source code (text level) to compilation result (byte code level), and the detection accuracy of the model is increased from 0.797 to 0.834, indicating bytecode. The level of detection is better. In addition, when the traditional naive Bayesian classification algorithm is replaced by the neural network classification algorithm MLP, the accuracy of the model is increased from 0.834 to 0.944, the accuracy is improved from 0.761 to 0.932, and the recall rate is increased from 0.771 to 0.968, which greatly improves the detection model. The performance indicates that the neural network-based detection model has better fitting ability and better detection ability for unknown samples.

Table 1 Performance comparison of various WebShell detection models

Performance	**Naive Bayesian-based detection model at the text level**	**Naive Bayes-based detection model at the bytecode level**	**MLP-based detection model at the bytecode level**
Accuracy	0.797	0.834	0.944
Precision	0.756	0.761	0.932
Recall	0.781	0.771	0.968

5. Conclusion

This paper introduces the harm and characteristics of WebShell, analyzes the existing WebShell detection methods in detail, and points out two common problems: usually research at the source level, which is easy to be affected by traditional WebShell escape methods such as code comments and code confusion. Detection models usually use traditional classifiers such as linear regression, decision trees, naive Bayes, etc. to study the samples, which limits the detection accuracy of the model, and on the other hand, makes it difficult to deal with unknown types of WebShell. In order to solve the above problems, this paper proposes a WebShell detection method based on multi-layer neural network. This method extracts features in the code compilation results, effectively solving the impact of the traditional WebShell escape means. In addition, using the bytecode sequence as a sample feature effectively utilizes the context of the code and improves the detection efficiency of the model. The method also uses the Multiply Layer Perception algorithm in the neural network classification algorithm to learn the samples, which effectively improves the generalization ability of the detection model, that is, the ability to detect unknown types of WebShell is stronger. The next step is to extend the detection range from bytecode to data, that is, to combine the instruction and data to construct a detection model to improve detection accuracy.

References

[1] National Internet Emergency Center. China Internet Network Security Report in 2016 [EB/OL]. (2017-05-27) [2018-01-12]. http://www.cert.org.cn/publish/main/46/2017/20170527151228908822757/ 20170527151228908822757_.html

[2] Hansen R J, Patterson M L.Guns and Butter:Towards Formal Axioms of Input Validation[J].Black Hat USA, 2005 (08): 1-6.

[3] Wrench P M, Irwin B V W.Towards a PHP Webshell Taxonomy Using Deobfuscation-assisted Similarity Analysis [C]. 2015 Information Security for South Africa (ISSA), 2015.

[4] Deng L Y,Lee D L,Chen Y H,et al.Lexical Analysis for the Webshell Attacks [C]. 2016 International Symposium on Computer, Consumer and Control (IS3C), 2016.

[5] Kelly K. O'Brien, Colquhoun H, Levac D , et al. Advancing scoping study methodology: a web-based survey and consultation of perceptions on terminology, definition and methodological steps [J]. Bmc Health Services Research, 2016, 16 (1): 305.

[6] Byczkowski T L, Munafo J K, Britto M T . Family perceptions of the usability and value of chronic disease web-based patient portals [J]. Health Informatics Journal, 2014, 20 (2): 151-162.

[7] Zhang Yanjun, Yang Xiaodong, Liu Yi, Zheng Dayuan, Bi Shujun. Research on the Construction of Wisdom Auditing Platform Based on Spatio-temporal Big Data [J]. Computer and Digital Engineering, 2019, 47 (03): 616-619+637.

[8] Yi Liu, Jiawen Peng, and Zhihao Yu. 2018. Big Data Platform Architecture under the Background of Financial Technology: In the Insurance Industry as an Example. In Proceedings of the 2018 International Conference on Big Data Engineering and Technology (BDET 2018). ACM, New York, NY, USA, 31-35.

[9] Y. Wu, Y. Liu, A. Alghamdi, K. Polat, and J. Peng, \Dominant dataset selection algorithms for time-series data based on linear transformation," CoRR, vol. abs/1903.00237, 2019. [Online]. Available: http://arxiv.org/abs/1903.00237

[10] YE Fei, GONG Jian, YANG Wang. Black Box Detection of Webshell Based on Support Vector Machine [J]. Journal of Nanjing University of Aeronautics & Astronautics, 2015 (06): 924-930.