Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 5533963, 12 pages
https://doi.org/10.1155/2021/5533963

Research Article

Webshell Detection Based on Executable Data Characteristics of PHP Code

Zulie Pan,[1,2] **Yuanchao Chen**,[1,2] **Yu Chen**,[1,2] **Yi Shen**,[1,2] **and Xuanzhen Guo**[1,2]

[1]*College of Electronic Engineering, National University of Defense Technology, Hefei 230011, China*
[2]*Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China*

Correspondence should be addressed to Yuanchao Chen; chenyuanchao@nudt.edu.cn

Received 7 January 2021; Revised 3 February 2021; Accepted 8 March 2021; Published 23 March 2021

Academic Editor: Di Zhang

Copyright © 2021 Zulie Pan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A webshell is a malicious backdoor that allows remote access and control to a web server by executing arbitrary commands. The wide use of obfuscation and encryption technologies has greatly increased the difficulty of webshell detection. To this end, we propose a novel webshell detection model leveraging the grammatical features extracted from the PHP code. The key idea is to combine the executable data characteristics of the PHP code with static text features for webshell classification. To verify the proposed model, we construct a cleaned data set of webshell consisting of 2,917 samples from 17 webshell collection projects and conduct extensive experiments. We have designed three sets of controlled experiments, the results of which show that the accuracy of the three algorithms has reached more than 99.40%, the highest reached 99.66%, the recall rate has been increased by at least 1.8%, the most increased by 6.75%, and the F1 value has increased by 2.02% on average. It not only confirms the efficiency of the grammatical features in webshell detection but also shows that our system significantly outperforms several state-of-the-art rivals in terms of detection accuracy and recall rate.

1. Introduction

Webshell is a web backdoor written in the web scripting languages that provide a covert way to communicate with the server [1]. Along with the fast development of the Internet, web attacks occur much more frequently, among which implanting webshell into target websites is one of the most commonly used means for attackers [2]. Attackers can use webshell to gain control of the website server, so as to further conduct information sniffing, data theft, or tampering. In order to bypass the webshell detection tools, attackers usually use obfuscation and encryption or just embed webshell codes into normal files, thus preventing webshell files from being detected. Therefore, precisely detecting and identifying webshell are growing much tougher, and how to accurately conduct webshell detection has already become one important problem for preventing web-based attacks.

Currently, there are two distinct technical routes of webshell detection—dynamic feature detection and static feature detection. Specifically, dynamic feature detection is mainly based on the webshell file behavior [3], webshell communication traffic [4–6], and other characteristics. It only works when webshell is dynamically executed. On the one hand, dynamic detection needs hook [7], sandbox [8] and other technologies. On the other hand, it has to detect the operation of files and traffic communication. Therefore, dynamic detection could consume much of the computing resources of the server and significantly deteriorate its performance. As a result, static feature detection methods have become the focus of research.

The static feature detection methods are mainly based on webshell text content as well as web log information [9, 10] for analysis and detection. Although regular expressions [11] are the earliest widely used method to detect the contents of webshell, they are confined to be extracted from the existing webshell and need to be constantly updated. To this end, the static feature detection method cannot detect the unknown webshell. Moreover, since webshell code obfuscation and encryption technology continues to mature, detection methods based on regular expressions can be easily bypassed. Machine learning emerges recently in various fields including cyberspace security [12]. Some researchers began to apply this technology to webshell detection, and results prove that it can play a vital role in webshell detection. The construction of feature engineering, namely, the features which are involved are the webshell characteristics we choose for model training, plays a critical role in webshell detection methods based on machine learning.

The main contributions of this paper are summarized as follows:

(1) We first proposed the concept of the executable data characteristics of PHP code and used it for webshell detection research. From the perspective of the grammatical features of webshell, we constructed a webshell detection model based on the executable data characteristics of PHP code

(2) We construct a cleaned data set to facilitate subsequent related research via collecting 17 existing webshell data sets on Github. The md5 algorithm has been used to remove the duplicate webshell samples and some non-PHP code webshell files through manual analysis

(3) We conduct extensive experiments to evaluate the performance of our proposed method. The results show that the executable data characteristics of PHP code is one of the important grammatical features of PHP webshell, which significantly improves the accuracy of the detection model. Moreover, the executable data characteristics of PHP code can better improve the distinguishing ability of the detection model compared with the traditional static statistical characteristics

In the next section, we review some representative related research to outline the motivation of our research. In Section 3, we introduce our system model in detail, including the opcode text vector library and data executable features of PHP code. In Section 4, we systematically evaluate our detection model. Finally, we summarize our work in Section 5.

2. Related Work

For webshell detection based on static features, the feature selection is mainly divided into two types: text features and grammatical features. Current research on webshell detection using machine learning mainly focuses on the text feature. In order to make up for the shortcomings of the detection method using regular expression detection, researchers have proposed a detection method based on statistical features from the perspective of text features. That is, by extracting statistical features in webshell, such as information entropy, longest words, index of coincidence, and compression ratio, these features combine to form a feature matrix for model construction. And then, through experiments, it was found that PHP opcode can help improve the capability of the detection of webshell in PHP language. By combining the statistical features, PHP opcode word frequency, PHP opcode text vector library, signature functions, word relevance, and other features together to construct a feature matrix as an input training model for classification, significant performance has been achieved.

Hu et al. proposed a webshell detection model based on the decision tree focusing on the detection of PHP webshell [13]. The basic attributes of the sample, such as the number of words and lines in the text, called functions, are extracted to construct the features and train the decision tree classification model for webshell detection. Although the number of words, line numbers of the text, and called functions cannot distinguish between normal files and webshell files, this research has made a good attempt to build a detection model using the text features of webshell.

Meng et al. [14] build a matrix to make use of the SVM algorithm training model for webshell detection. To this end, web page attribute characteristics are extracted. These characteristics include page length, code lines, number of comments and other characteristics, and page operation attributes. Besides encryption and decryption function calls, system, eval, exec, shell_exec, and other function calls, character operation function calls, system function calls, file operations, FTP (File Transfer Protocol) operations, database operations, ActiveX control calls, and other features are all involved in the matrix building.

Hu proposed a model of webshell detection based on Bayesian theory [15] and extracted the common statistical features of file analysis for the construction of feature engineering. Features involved are information entropy, longest word length, compression ratio, index of coincidence, and other features, all of which would be put into the Bayesian classifier to train the model for webshell detection.

The model of detecting webshell based on random forest with FastText was first proposed by Fang et al. to extract the opcode of PHP code as a feature for model construction [16]. By extracting the opcode of the PHP code to train the FastTest model, the FastText model is used for preclassification processing. Statistical features such as the longest string, information entropy, index of coincidence, text features such as sensitive functions, and blacklist keywords are used as input to train random deep forest classification model.

Cui et al. proposed the model of webshell detection based on random forest–gradient boosting decision tree algorithm [17]. First attain opcode hash vector and text vector library features extracted from PHP opcode processed by the TF-IDF (term frequency–inverse document frequency) [18] vector. All these features are used to train a random deep forest model, so as to obtain preliminary preprocessing results. Combined with statistical features such as information entropy, index of coincidence, compression ratio, length of the longest word, and number of signature function matches, to construct a feature matrix as the input of the GBDT (Gradient Boosting Decision Tree) classifier for model training, the final detection result is obtained.

In 2019, Li et al. proposed a webshell detection model based on the word attention mechanism [19]; first, use word2vec to vectorize the text content, then use the GRU (Gated Recurrent Unit) model and the attention mechanism model for training, and finally input to a sigmoid function full connection layer for webshell classification.

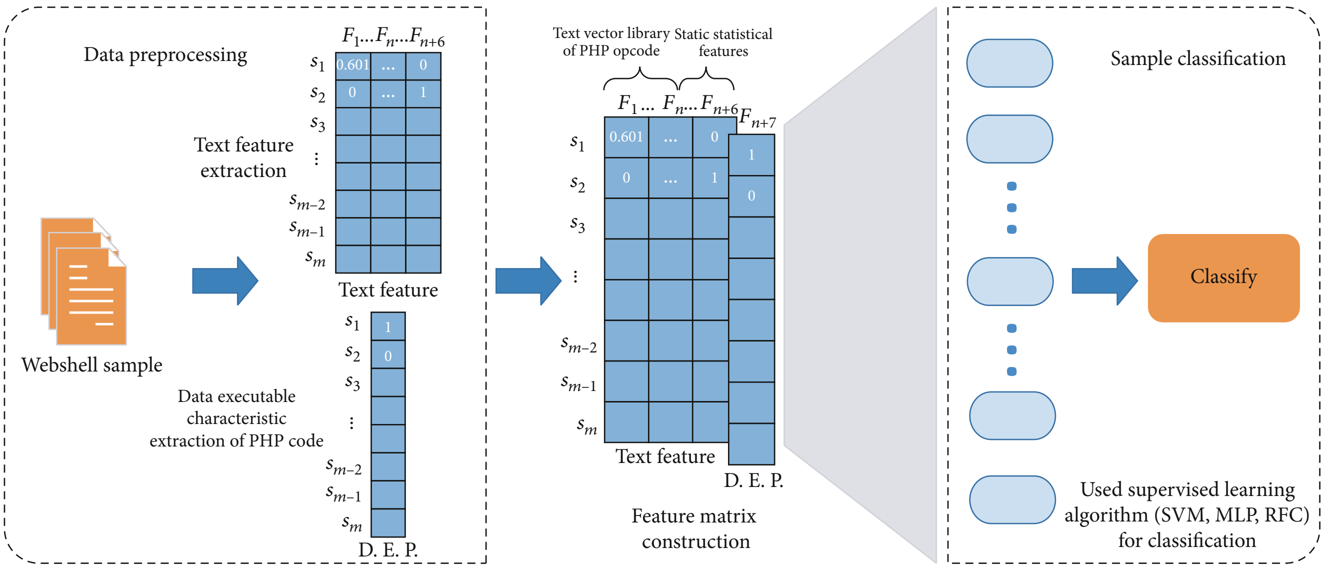
Data preprocessing
Text feature extraction
Webshell sample
Text feature
Data executable characteristic extraction of PHP code
D. E. P.
Text vector library of PHP opcode
Static statistical features
Text feature
D. E. P.
Feature matrix construction
Sample classification
Classify
Used supervised learning algorithm (SVM, MLP, RFC) for classification

Figure 1: The structure of the detection model.

The statistical characteristics are mainly the attribute values of some aspects of the file, which summarize the characteristics of webshell from the perspective of the entire file. However, with the rapid development of web services, a large number of web service frameworks have emerged, and developers have begun to use code obfuscation and encryption techniques in the project code to avoid the leakage of source code, which will in the end cause the statistical feature of normal files to resemble webshell, making the webshell detection model based on statistical features lose its original advantages. This also fully illustrates that webshell detection based on statistical characteristics is not comprehensive. Webshell not only has the attributes of a file but also has the structure of a scripting language.

According to the analysis above, due to the growing disadvantages of statistical features, this paper is dedicated to proving that the executable data characteristics of the PHP code can be effective in webshell detection.

3. Model Architecture

The structure of the detection model is shown in Figure 1, which includes three parts: data preprocessing, feature matrix construction, and classifications via supervised learning algorithms.

Firstly, we preprocess the data and extract the features from the text and data executable, respectively. From the view of text, we choose to use the text vector library of opcode with the best discrimination ability in the current research to combine the static statistical features of the samples. Secondly, we combine the PHP opcode text vector library extracted in the preprocessing stage, the static statistical features of samples, and the executable data characteristics of PHP code to form the feature matrix to describe the samples. Finally, supervised learning algorithm is used for training classification.

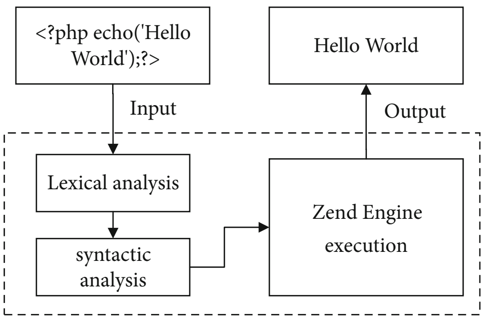

Figure 2: The running process of PHP code.

3.1. Static Statistical Features. We selected six types of static statistical features used in the literature [12], including information entropy, index of coincidence, length of the longest word, amount of matched signatures, data compression ratio, and uses of the eval function. Combining these features, we can get a 6-dimensional feature vector.

3.2. Text Vector Library of PHP Opcode. The running process of PHP code includes three stages: lexical analysis, syntax analysis, and Zend engine execution. After these three stages of processing, the corresponding results of PHP code will be output, as shown in Figure 2.

Through lexical analysis, PHP code is divided into language fragments. After syntax analysis, the language fragments are transformed into meaningful expressions which are input into Zend engine for execution. In the Zend engine execution stage, it will translate the expressions generated by syntax analysis into a series of opcodes and perform corresponding operations according to opcode. For example, the original code of a webshell sample is as follows.

The opcode sequence generated by Zend engine coding is as follows: [‘FETCH CONSTANT’,‘FETCH R’,‘FETCH DIM R’,‘INCLUDE OR EVAL’,‘RETURN’], as shown in Table 1.

```
<?phpeval($_REQUEST['password']) ; ?>
```

Code 1

Table 1: Opcode sequence.

No.	Opcode
0	FETCH CONSTANT
1	FETCH R
2	FETCH DIM R
3	INCLUDE OR EVAL
4	RETURN

```
<?php
error_reporting(E_ALL^E_NOTICE);
define('%uFFFD%uFFFD','%uFFFD%uFFFD%uF
FFD');
$_SERVER[%uFFFD%uFFFD]=explode('|-|; |(
',"password");
eval($_REQUEST[$_SERVER{%uFFFD%uFFFD}[
0]])
?>
```

Code 2

That is to say, Zend engine will parse and execute according to the opcode sequence generated by compilation and get the above webshell code by obfuscating encryption:

The opcode sequence of this code is as follows: ['INIT_FCALL', 'SEND_VAL', 'DO_ICALL', 'INIT_FCALL', 'SEND_VAL', 'SEND_VAL', 'DO_ICALL', 'FETC_CONSTANT', 'INIT_FCALL'. 'SEND_VAL', 'SEND_VAL', 'DO_ICALL', 'FETCH_W', 'ASSIGN_DIM', 'OP_DATA', 'FETCH_CONSTANT', 'FETCH_R', 'FETCH_DIM_R', 'FETCH_DIM_R', 'FETCH_R', 'FETCH_DIM_R', 'INCLUDE_OR_EVAL', 'RETURN']. By comparison, it can be found that opcode generated by obfuscated encryption contains the opcode of source code. From the perspective of opcode, after obfuscated encryption, webshell code adds some additional operations for changing code style on the basis of original operation sequence and does not change its original core opcode sequence. Therefore, the use of opcode in webshell detection can play a great role in webshell detection. Document [16] first demonstrated the application of PHP opcode to webshell detection through the comparative experiment for the first time in 2018, which can improve the discrimination ability of the detection model.

In the previous research, we usually use PHP's VLD [20] extension to extract opcode, but when we are dealing with the large amount of code after confusion encryption, the low efficiency of VLD extension processing the output file opcode is unbearable, so we use PHP' native debugger PHPDBG [21] to extract opcode of PHP code.

In the detection model of this paper, we focus on the text vector library of opcode, and the opcode sequence of each PHP file has a certain correlation; that is to say, each opcode has a certain correlation with the opcode before and after it. Therefore, n-gram model is used to preprocess opcode to generate the opcode word frequency matrix, and then, the TF-IDF model is introduced to calculate the TF-IDF value of each opcode segment. The text vector library of opcode is generated by filtering out the opcode fragments with less distinguishing ability. Its concrete process is shown in Figure 3.

N-gram [22] is a probabilistic language model based on the Markov assumption that the occurrence of the n th word is related to the $(n-1)$th words; the probability of the entire sentence is equal to the probability product of each word appearing. Assuming that the sequence$S$ is composed of the word sequences W_1, W_2, W_3,...,W_n, the probability of the sentence S appearing could be expressed in this way:

$$P(S) = P(W_1)P(W_2)P(W_3)\cdots P(W_n). \tag{1}$$

The probability of each word occurrence is calculated by sample statistics when the model is built. Using the N-Gram model to preprocess opcode, a large number of opcode sequence samples are divided into n-length opcode corpus fragments. For example, opcode sequences are as follows: ['ASSIGN', 'INCLUDE_OR_EVAL', 'CONCAT', 'INCLUDE_OR_EVAL', 'RETURN'], ['ASSIGN', 'INIT_FCALL', 'SEND_VAL', 'DO_ICALL', 'CONCAT', 'INCLUDE_OR_EVAL', 'RETURN'], The corpus fragments generated by N=3 are shown in Table 2.

The frequency matrix of opcode sequence can be constructed by calculating the number of occurrences of corpus fragments:

$$\begin{bmatrix} [1\ 0\ 1\ 0\ 1\ 0\ 0] \\ [0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1] \end{bmatrix} \tag{2}$$

TF-IDF [18] is used to evaluate the importance of a corpus fragment in the corpus, where TF (word frequency) represents the frequency of a corpus fragment in the corpus. To be specific, $n(i, j)$ represents the number of occurrences of corpus fragment t_i in file d_j, and $\sum_k n_{k,j}$ means the total number of occurrences of all corpus fragments in file d_j. The calculation formula is given like this:

$$tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}. \tag{3}$$

IDF (Reverse File Frequency) is used to indicate whether a corpus fragment has good class discrimination ability in the corpus. The formula is (4), in which $|D|$ represents the sum of the files in the corpus. $|\{j : t_i \in d_j\}|$ represents the number of files that contain corpus fragments t_i. If the corpus fragment is not in the corpus, it will result in a denominator of 0. So generally, the denominator is symbolled as $|\{j : t_i \in d_j\}| + 1$. If fewer files contain corpus fragments t_i, the IDF value grows with files in corpus fragments decreasing. This shows that

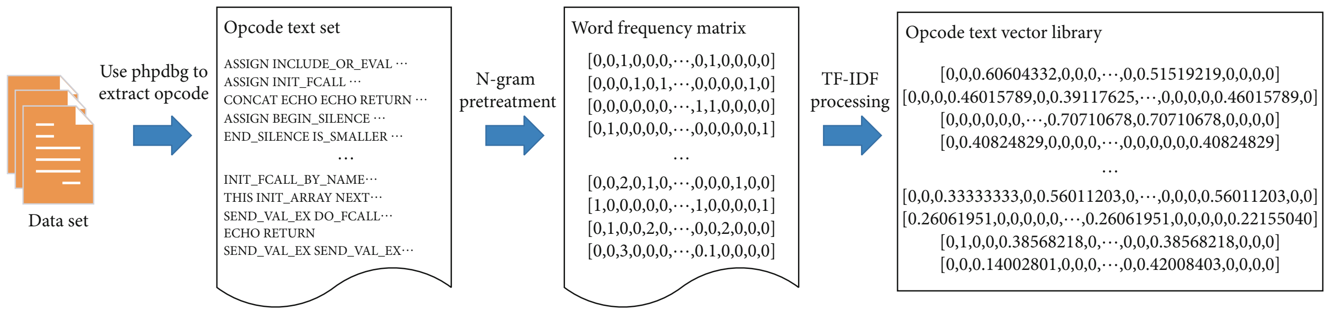

Figure 3: PHP opcode processing flow.

Table 2: The corpus fragments generated by N =3.

#	Corpus fragments
0	[‘ASSIGN’, ‘INCLUDE_OR_EVAL’, ‘CONCAT’]
1	[‘ASSIGN’, ‘INIT_FCALL’, ‘SEND_VAL’]
2	[‘CONCAT’, ‘INCLUDE_OR_EVAL’, ‘RETURN’]
3	[‘DO_ICALL’, ‘CONCAT’, ‘INCLUDE_OR_EVAL’]
4	[‘INCLUDE_OR_EVAL’, ‘CONCAT’, ‘INCLUDE_OR_EVAL’]
5	[‘INIT_FCALL’, ‘SEND_VAL’, ‘DO_ICALL’]
6	[‘SEND_VAL’, ‘DO_ICALL’, ‘CONCAT’]

corpus segment shows excellent performance in distinguishing categories.

$$idf_{i,j,D} = \log \frac{|D|}{\left|\{j : t_i \in d_j\}\right|}. \tag{4}$$

TF-IDF is the product of TF and IDF, and the formula is (5). The TF-IDF value can be used to filter out some of the referenced corpus fragments, reserving corpus fragments with good discriminatory ability.

$$tf - idf_{i,j,D} = tf_{i,j} * idf_{i,j,D}. \tag{5}$$

The opcode text vector library can be obtained by passing the frequency matrix generated by the *n*-gram preprocessing above into the TF-IDF model.

By transmitting frequency matrix generated by *n*-gram preprocessing into the TF-IDF model, the opcode text vector library is attained:

$$\begin{bmatrix} [0.6317 \quad 0 \quad 0.4494 \quad 0 \quad 0.6317 \quad 0 \quad 0] \\ [0 \quad 0.4712 \quad 0.3351 \quad 0.4712 \quad 0 \quad 0.4712 \quad 0.4712] \end{bmatrix} \tag{6}$$

3.3. The Executable Data Characteristics of PHP Code. In the parsing of PHP language, there is no difference between data segment and code segment. When PHP receives data from users, the input data may not only be processed as characters but also be parsed and executed as PHP code. For example, the PHP code is as follows.

```
<?phpecho($_GET[‘txt’]) ; ?>
```

Code 3

```
<?phpeval($_GET[‘txt’]) ; ?>
```

Code 4

We can directly get knowledge about the function of the code. That is to say, the data input by the user would be printed in the page by the echo function. And the overall function of the code would not be affected due to the different inputs from users, so the function of the code is clear. If the echo function in the above code is replaced with eval, it will become a sentence webshell in PHP.

The specific function of this code cannot be determined directly. If the data obtained by $_GET[‘txt’] is ‘1 +1’, this code would output the calculation result ‘2’. If the data obtained by $_GET[‘txt’] is the function phpinfo() in PHP, this code would print the relevant configuration information of the server of PHP language. If the acquired data is ‘system (whoami)’, the user input is converted into a system function to execute the corresponding system command, which means that the data entered by the user is actually executed as a PHP code in this code. To sum up, input data determines the actual function of the code, so we make a definition as follow:

Definition 1. In a section of PHP code, the input data is parsed and executed as PHP code or system command to determine the actual function of this section of code, which is called data executable characteristic.

And webshell is able to realize various functions through a simple piece of code. For example, you can get information about the running environment of the web server; perform file uploading, downloading, or editing operations; connect to the database; get the command execution environment of the server; and so on. Therefore, we are certain that the majority of webshell will respond to users’ inputs to achieve the correspondence functionality. In other words, the majority of webshell possesses data executability features.

```
array (
  0: Stmt_Expression (
    expr: Expr_Eval (
      expr: Expr_ArrayDimFetch (
        var: Expr_Variable (
              name: _REQUEST
        )
        dim: Scalar_String (
                    value: txt
        )
      )
    )
  )
)
```

Code 5

An abstract syntax tree [23] for PHP code is needed to extract executable data characteristics of PHP code. An abstract syntax tree is a tree that represents the grammatical structure of a program's source code, where each node represents a structure in the source code. Abstract syntax tree for PHP code can be generated by using PHP-Parser [24]. PHP-Parser is an open-source PHP abstract grammar tree generation tool programmed in PHP language based on the Zend engine. For example, the abstract syntax tree generated by the sentence webshell mentioned above through PHP-parse is as follows.

Stmt and Expr represent the declaring node and the expression node, respectively. Besides, variables in the formula are expressed by Variable, and a string constant is represented by Scalar_String. Through the abstract grammar tree, we can intuitively understand the overall grammar structure of the code. Extracting the executable data characteristics of the PHP code allows us to analyze Eval, FuncCall, MethodCall, and ShellExec nodes in Expr nodes of the abstract syntax tree by matching, analyzing, and judging the attributes (function name, parameter type) of these nodes to get knowledge that whether the function in the expression node can execute data as the PHP code or system command and whether the parameter of the function is a variable node. Variable node, the function of expression node, executes variable parameters as PHP code or system command, while the actual function of the code is dynamically determined by the value of the variable. If all the above conditions are met, the PHP code is judged to have executable data characteristics. The algorithm to extract the executable data characteristics of the PHP code shown in Algorithm 1:

```
Input: PHP language samples files
Output: one-dimensional matrix of executable data characteristics of the sample
  1. Convert PHP code to abstract syntax tree, turn to step 2.
  2. Judges whether there are Eval, FuncCall, MethodCall,
     or ShellExec nodes under Expr nodes in the abstract
     grammar tree, and if matched, turn to step 3; else,
     return 0.
  3. Judges whether the functions in the nodes above are
     functions that can execute the data as PHP code or
     system commands, such as eval, exec, system, etc. If
     the answer is yes, turn to step 4; else, return 0.
  4. Judge the type of parameter in the function, whether
     the parameter is variable. If it is, return 1, if not,
     return 0.
```

Algorithm 1: Executable data characteristic extraction of PHP code.

Finally, a one-dimensional matrix is generated to describe the executable data characteristics of the sample. Samples which possess executable data characteristics will be marked as 1, and samples without data executability are marked as 0.

3.4. Feature Matrix Construction and Supervised Learning Algorithm. The feature matrix is constructed by combining the text vector library, sample static statistical features, and data executable features of opcode. The characteristic matrix is constituted by M rows and $N + 7$ columns. M represents a total of M samples. The first N columns represent the text vector of the sample opcode. Column $N + 1$ to $N + 6$ represent the static statistical characteristics of the sample. The $N + 7$th column represents the executable data characteristics of the sample. This paper focuses on whether the executable data characteristics of PHP code can play a vital role in webshell detection. Therefore, the impact of specific algorithm on detection is not discussed. Supervised learning refers to using a set of labeled data to learn its mapping from input to output and applies this mapping relationship to unknown data in order to achieve the purpose of classification or regression. The constructed feature matrix is introduced into the supervised learning algorithm to construct the model. Finally, the model is used to classify each test data.

4. Experimental Analysis

4.1. Data Sets. Since we found there is no ready-made and cleaned data set of PHP webshell on the internet available, we collected a total of 6021 webshell samples of PHP languages from 17 open-source projects. Github projects involved are shown in Table 3.

Since webshell samples collected by each github project inevitably include partial duplicate sample files, in order to avoid repeated webshell sample files affecting the experimental results, we used the md5 algorithm to reprocess 6,021 webshell samples and obtained a total of 3,211 nonrepeated webshell sample files. Meanwhile, in order to ensure the accuracy of the data, 294 non-PHP webshell files were excluded by manual analysis, so the final number of webshell samples was 2,917. By making a test of executable data characteristics, a total of 2,696 samples were found to have this property. And the remaining 221 webshell samples were analyzed manually only to find that samples that did not detect the executable data characteristics of PHP code were only related to operations with only one function such as file upload operation, database connection operation, and file system directory operation. This data also fully demonstrated

Table 3: Github projects lists.

No.	Github projects
0	JohnTroony/PHP-Webshells
1	xl7dev/Webshell
2	ysrc/Webshell-sample
3	tennc/Webshell
4	BlackArch/Webshells
5	JoyChou93/Webshell
6	bartblaze/PHP-backdoors
7	WangYihang/Webshell-sniper
8	tanjiti/WebshellSample
9	tdifg/Webshell
10	LandGrey/Webshell-detect-bypass
11	backlion/Webshell
12	Webshellpub/awsome-Webshell
13	x-o-r-r-o/PHP-Webshells-Collection
14	S9MF/S9MF-PHP-Webshell-bypass
15	backdoorhub/shell-backdoor-list
16	amitnaik/PHP-backdoor

Table 4: Content management system lists.

#	CMS	Version
0	Wordpress	5.4
1	Joomla	3.9.16
2	Laravel	7.6.2
3	PHPBB	3.3.0
4	Typecho	1.1
5	ThinkPHP	5.0.24
6	Seacms	10.1
7	MetInfo	7.0.0
8	DiscuzX	3.4

the truth that most PHP language webshell had executable data characteristics of PHP code. We used the same approach to collect 9,736 samples of nonrepeated normal web pages in the PHP language from 9 well-known open-source web content management systems (content management system (CMS)). The relevant CMS is shown in Table 4.

We divided the 2,917 webshell samples and 9,736 normal web page samples obtained by processing into the training set and the test set randomly according to the ratio of 7 : 3; that is, the training set consists of 2,044 webshell samples and 6,815 normal web page samples totaling 8,859, and the test set consists of 3,794 samples which consisted by 873 webshell samples and 2,921 normal web page samples. After the assignment was completed, the normal web samples and webshell samples for training were combined to form the training set of the model. The normal web samples and the webshell samples for classification testing were mixed to form the test set of the model. In the previous research, the data set is usually preprocessed uniformly, and then, the training set and the test set are divided. Since we use the N-gram and TF-IDF algorithm to extract the opcode text vector library, the divided training set and the test set will have relevance in the opcode text vector library if the data is unified preprocessing. With the aim of eliminating the relevance of the training set and test set, we have divided the training set and test set before preprocessing. The preprocessing of the data and the construction of the feature matrix are carried out separately to ensure that the data in the training set and the test set are uncorrelated and to ensure the validity of the experimental results. We uploaded the cleaned data set to the Github project for use in the subsequent experiments of webshell detection research, which can be downloaded from https://github.com/Cyc1e183/PHP-Webshell-Dataset.

4.2. Algorithm Parameter Setting. Aiming to better compare the experiments' results and analyze the impact of executable data characteristics of PHP code on model performance, we adopt the random forest (RF) algorithm [25], support vector machine algorithm (SVM) [14], and multilayer perceptron (MLP) [26] with the same training set for webshell classification. Besides, we do not explore the optimal situation of algorithm parameter setting. In the experiment, we set the N value of the N-gram algorithm used in opcode text vector library processing as '3'; The sample-set segmentation strategy of the random forest algorithm was set to information entropy, the number was set to 100, the value of the random seed was set to 2, and the remaining parameters were used for model training with default settings. Set the kernel type of the support vector machine algorithm to the linear kernel function. The penalty factor is set to 1. The remaining parameters are trained by default. The weight optimization algorithm in multilayer perceptron was set to a random gradient-based optimization algorithm. The regularization parameter is set to 0.0001. The hidden layers are set to 1 with 100 hidden units in this layer. We choose the logistic function as the hidden layer activation function. The random seed was set to 1. The maximum number of iterations was set to 150, while the initial learning rate was set to 0.089. The remaining parameters were set by default design for model training.

4.3. Experimental Results and Analysis. We divided the experiment into three groups. The experiment uniformly used the same parameter for random forest algorithm, support vector machine algorithm, and multilayer perceptron. The first group combined the opcode text vector library, sample static statistical features, and executable data characteristics of the PHP code to form a feature matrix construction model for the experiment (we call it model 1). The second group used the opcode text vector library combined with static statistical features as a feature matrix to build a model (we call it model 2), and the last group used the opcode text vector library and executable data characteristics of PHP code to constitute a feature matrix construction model for experimentation (we call it model 3). Three experiments are set to compare the excellency of executable data characteristics of the PHP code with executable data characteristics. What is more, we can also know whether executable data characteristics of PHP code would affect the performance of the detection model.

Table 5: Confusion matrices of each group of experiments.

Model name	Classification algorithm	Actual	Predicted Positive	Predicted Negative
Model 1	Random forest	Positive	TP = 869*	FN = 4
		Negative	FP = 9	TN = 2912*
	Support vector machine	Positive	TP = 865*	FN = 8
		Negative	FP = 14	TN = 2907
	Multilayer perceptron	Positive	TP = 867*	FN = 6
		Negative	FP = 12	TN = 2909*
Model 2	Random forest	Positive	TP = 854	FN = 19
		Negative	FP = 9	TN = 2912*
	Support vector machine	Positive	TP = 806	FN = 67
		Negative	FP = 12	TN = 2909*
	Multilayer perceptron	Positive	TP = 848	FN = 25
		Negative	FP = 24	TN = 2897
Model 3	Random forest	Positive	TP = 867	FN = 6
		Negative	FP = 16	TN = 2905
	Support vector machine	Positive	TP = 863	FN = 10
		Negative	FP = 34	TN = 2887
	Multilayer perceptron	Positive	TP = 859	FN = 14
		Negative	FP = 17	TN = 2904

*The maximum value of the same algorithm in different models.

We marked the webshell sample as positive and the normal web page file sample as negative in the experiment. Confusion matrices of each group of experiments are shown in Table 5.

While TP (true positive) indicates the number of webshell samples that the model recognizes correctly, FN (false negative) indicates the number of webshell samples recognized by the classification model as normal web page files. In contrast, TN (true negative) indicates the number of normal web page files that the classification model correctly identifies, and FP (false positive) means that the classification model recognizes the number of normal web pages as webshell. After comparing and analyzing the experimental data, adding the executable data characteristics of PHP code, the FN and FP values displayed in the confusion matrix have been significantly reduced.

We can see from the data in Table 5 that, compared to model 2 and model 3, the random forest and MLP models in model 1 have achieved the maximum value of TP and TN, while the TP in the SVM model has also achieved the maximum value. TP indicates the number of webshell samples that the model correctly identified, it is more critical to system security to be able to accurately identify the webshell. Compared with model 2, the TP of the three models' results has increased by 31 in model 1; the value of TN is not significantly improved. Compared with model 2, model 3 has an average increase of 9 in the TP value of the three models' results.

To more accurately conduct the experimental evaluation and make it easy for us to compare with other detection methods, we used five commonly used evaluation indicators for webshell detection evaluation: accuracy, precision, recall, F1 values, and comprehensive evaluation using ROC curves [27]. The four evaluation indicators are calculated as follows:

$$\begin{aligned}\text{Accuracy} &= \frac{(\text{TP}+\text{TN})}{(\text{TP}+\text{FN}+\text{FP}+\text{TN})},\\ \text{Precision} &= \frac{\text{TP}}{(\text{TP}+\text{FP})},\\ \text{Recall} &= \frac{(\text{TP})}{(\text{TP}+\text{FN})},\\ \text{F1} &= \frac{(2 * \text{Precision} * \text{Recall})}{(\text{Precision}+\text{Recall})}.\end{aligned} \tag{7}$$

While accuracy indicates the proportion of correctly predicted samples to the whole test set, precision indicates the proportion of predicted true-positive samples to all tested positive samples. Recall rate indicates the proportion of correctly predicted webshell samples. The F1 value is a comprehensive evaluation index combining the accuracy and recall rate. The evaluation index results of the model are shown in Table 6.

Model 1 involves a feature matrix that combines the opcode text vector library, sample static statistical features, and the executable data characteristics of PHP code. Model 2 adapts a feature matrix that only uses the combination of

Table 6: The evaluation index results.

Model name	Classification algorithm	Evaluation index			
		Accuracy	Precision	Recall	F1 score
Model 1	Random forest	0.9966*	0.9897*	0.9954*	0.9926*
	Support vector machine	0.9942*	0.9841	0.9908*	0.9874*
	Multilayer perceptron	0.9953*	0.9863*	0.9931*	0.9897*
Model 2	Random forest	0.9926	0.9896	0.9782	0.9839
	Support vector machine	0.9792	0.9853*	0.9233	0.9533
	Multilayer perceptron	0.9871	0.9725	0.9714	0.9719
Model 3	Random forest	0.9942	0.9819	0.9931	0.9875
	Support vector machine	0.9884	0.9621	0.9885	0.9751
	Multilayer perceptron	0.9918	0.9806	0.9840	0.9823

*Comparing the same algorithm in different models, the maximum value of the indicator.

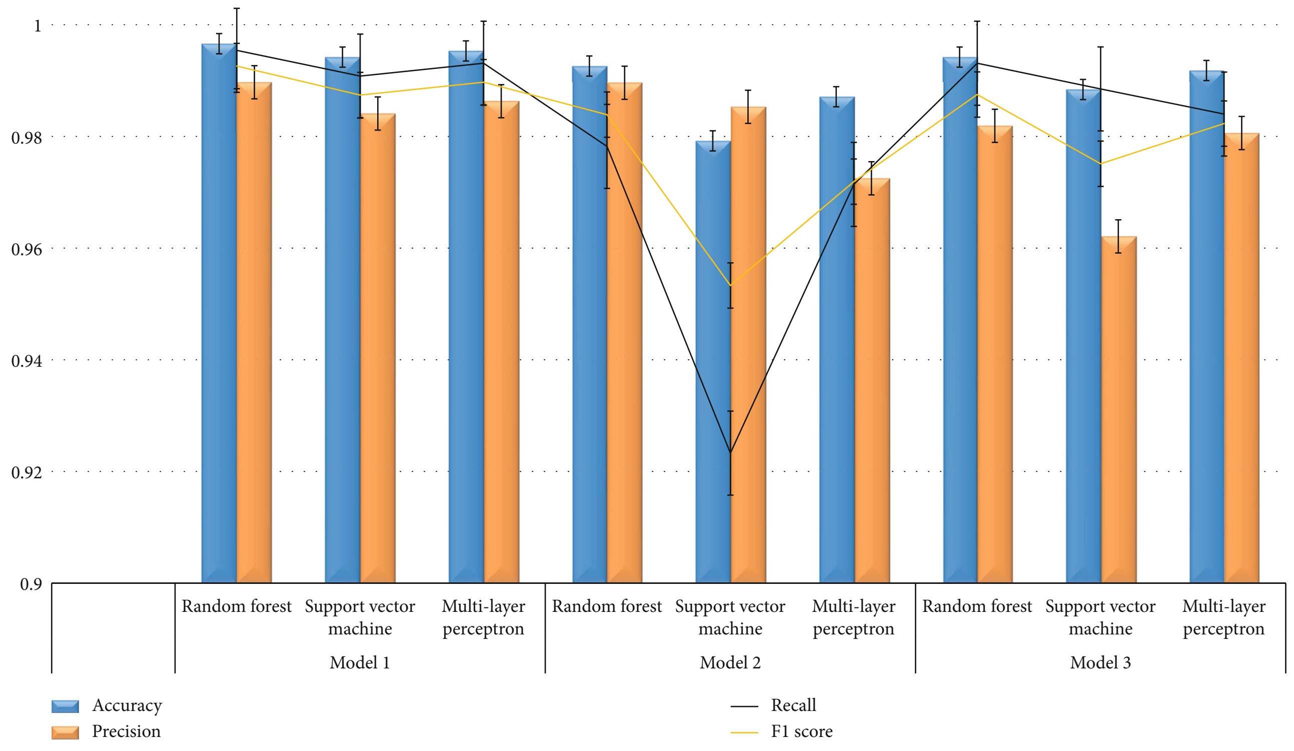

Figure 4: Data comparison of the three models.

opcode text vector library and sample static statistics. By comparing the experimental data of model 1 and model 2, we can see model 1 performs well than model 2. The evaluation indicators have been improved a lot, among which the accuracy of the three algorithms has reached more than 99.40%, the highest reached 99.66%, the recall rate has been increased by at least 1.8%, the most increased by 6.75%, and the F1 value has increased by 2.02% on average. The average value of precision has not changed much. The data fully shows that the executable data characteristics of PHP code can effectively improve the distinguishing ability of the model.

By comparing the experimental data of model 2 and model 3, results can be reached out. Specifically, the performance of the model constructed using the feature matrix of the combination of the opcode text vector library and the executable data characteristics of the PHP code is slightly better than the performance of the model built by model 2. Among them, the accuracy rate, recall rate, and F1 value are improved by 0.52%, 3.07%, and 1.19%, respectively. The data fully indicates that the executable data characteristics of PHP code can describe webshell better than static statistical features and have better discrimination ability. We can intuitively see the difference in the experimental results of the three models from Figure 4.

Figures 5–7 show the ROC curves of the three groups of experiments, respectively. And its horizontal coordinates are false-positive rate (false-positive rate (FPR)) which

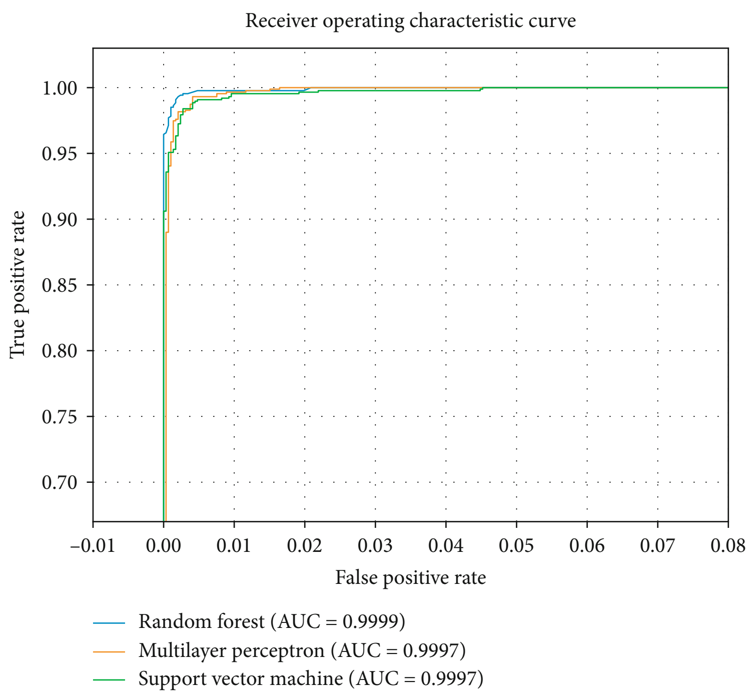

FIGURE 5: Model 1 ROC curve.

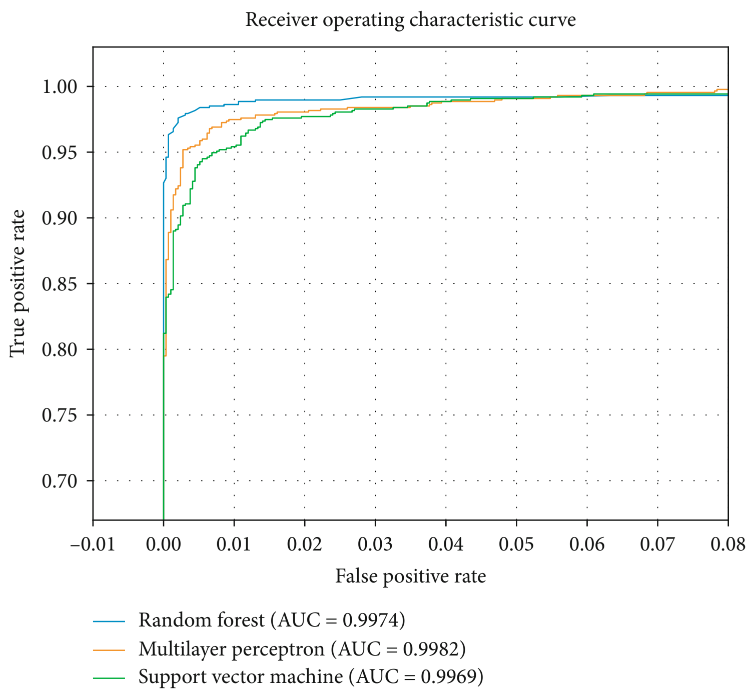

FIGURE 6: Model 2 ROC curve.

represents the false alarm rate of web normal page file. The ordinate stands for true-positive rate (true-positive rate (TPR)) which describes the accuracy of the classification of webshell documents. The ideal test model is supposed to fully identify webshell and normal web files when the value of TPR is 1 and the value of FPR is 0. In other words, the closer the area value under the ROC curve to 1, the higher the recognition accuracy of the detection will be. A comparison between the ROC curves has validated the performance of our detection model.

By analyzing the above data, it fully shows that the data executable feature of PHP code is an important grammatical feature of PHP language webshell. This grammatical feature can describe webshell better than traditional statistical-based text features, and has a better ability to distinguish between webshell files and normal web pages. By adding

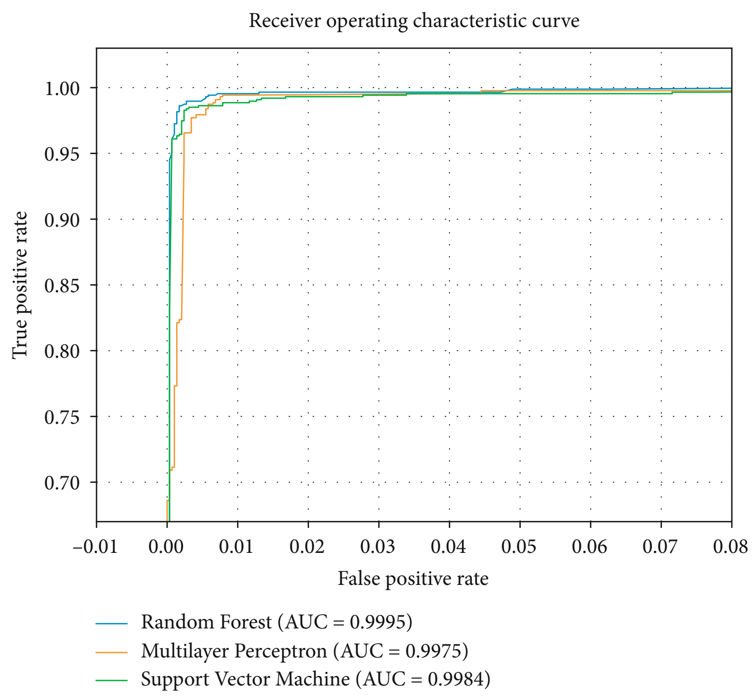

Figure 7: Model 3 ROC curve.

the data executable feature of the PHP code, the performance of the webshell detection model has been significantly improved.

Besides, we also randomly divide the training set and the test set at a ratio of 8 : 2; that is, the training set consists of 7,789 normal web page samples and 2,334 webshell samples, and the test set consists of 1,947 normal web page samples and 583 webshell samples. By comparing to model 2, the highest accuracy rate in model 1 has reached 99.64%, the most increased by 1.46%, and the least increased by 0.31%. The recall rate increased by at least 0.86%; the most increased by 6.35%. The F1 value increased by 2.33% on average, and the average precision increased by 1.16%. By comparing the experimental data of model 2 and model 3, results can be reached out. Specifically, model 3 is slightly better than the performance of the model built by model 2. Among them, the accuracy rate, precision rate, recall rate, and F1 value are improved by 0.94%, 0.83%, 3.37%, and 2.10%. From all the experimental results, the result of the random division according to the ratio of 8 : 2 and 7 : 3 is similar. Further confirmed the executable data characteristics of the PHP code can significantly improve the detection performance of the model, so, we do not describe in detail the experimental results of randomly dividing the data set and the test set according to the 8 : 2 ratio.

4.4. Comparison with Well-Known Webshell Detection Tools. We selected 4 of the current most popular webshell detection tools to compare with the module in this paper, which are D shield [28], Baidu WEBDIR+ [29], PHP Malware Finder [30], and SHELLPUB [31]. By using these tools to scan and detect all the sample files in our test set, we get results which are shown in Table 7.

Table 7: The evaluation index results.

Webshell detection tools	Version	Accuracy	Recall
D shield	V2.0.9	0.9863	0.9633
WEBDIR+	2020-0423-1800	0.9797	0.9118
PHP Malware Finder	—	0.8996	0.7904
SHELLPUB	V 1.7.0	0.9027	0.6277
Our model	—	0.9966	0.9954

It can be seen from the table that though the accuracy rate of the D shield can be as high as 98.63% and the Recall as high as 96.33%, the accuracy rate and the recall rate of our model detection can reach 99.66% and 99.54%. Obviously, the detection ability of our model is significantly better than these webshell detection tools. It is worth mentioning that the false-positive rate of Webdir+ for whitelist detection in the test data is 0%, which is also the goal we expect to achieve in subsequent work.

5. Conclusion

In this paper, we propose a webshell detection model based on static features of PHP codes. The key idea is to leverage PHP code executable data characteristics from the perspective of PHP code syntax features for webshell detection. To systematically evaluate our model, we firstly construct a cleaned data set of webshell consisting of 2,917 samples from 17 webshell collection projects and then conduct extensive experiments. The experimental results have verified the efficiency of our model by achieving 99.66% detection accuracy, without exploring the optimization of the machine learning algorithm. Moreover, our detection model outperforms several popular webshell detection tools in terms of accuracy, precision, recall rate, F1 value, and ROC curve. It is confirmed that the executable data characteristics of PHP code are significant grammatical features of webshell and can effectively improve the performance of the detection model.

Data Availability

We uploaded the data set to Github at https://github.com/Cyc1e183/PHP-Webshell-Dataset.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

We are very thankful to Hui Guo, Zhijie Xie, and Zhihao Hu for their help in the preparation of the experiment and paper review. This research was funded by the National Key R&D Program “Cyberspace Security” (2017YFB0802900).

References

[1] J. Kim, D.-H. Yoo, H. Jang, and K. Jeong, “WebSHArk 1.0: a benchmark collection for malicious web shell detection,” *Journal of Information Processing Systems*, vol. 11, no. 2, pp. 229–238, 2015.

[2] T. D. Tu, C. Guang, G. Xiaojun, and P. Wubin, “Webshell detection techniques in web applications,” in *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pp. 1–7, Hefei, China, 2014.

[3] B. Yong, X. Liu, Y. Liu, H. Yin, L. Huang, and Q. Zhou, “Web behavior detection based on deep neural network,” in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pp. 1911–1916, 2018.

[4] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, “CNN-webshell: malicious web shell detection with convolutional neural network,” in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing - ICNCC 2017*, pp. 75–79, Kunming, China, 2017.

[5] H. Zhang, H. Guan, H. Yan et al., “Webshell traffic detection with character-level features based on deep learning,” *IEEE Access*, vol. 6, pp. 75268–75277, 2018.

[6] W. Yang, B. Sun, and B. Cui, *Innovative Mobile and Internet Services in Ubiquitous Computing. IMIS 2018*, Springer, 2018.

[7] J. Riordan, A. Wespi, and D. Zamboni, “How to hook worms [computer network security],” *IEEE Spectrum*, vol. 42, no. 5, pp. 32–36, 2005.

[8] J. Shukla, “Application sandbox to detect, remove, and prevent malware,” US Patent 11/769,297, 2008.

[9] S. Liuyang and F. Yong, “Webshell detection method research based on web log,” *Journal of Information Security Research*, vol. 1, p. 11, 2016.

[10] 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, 11 pages, 2019.

[11] K. Thompson, “Programming techniques: regular expression search algorithm,” *Communications of the ACM*, vol. 11, no. 6, pp. 419–422, 1968.

[12] J. B. Fraley and J. Cannady, “The promise of machine learning in cybersecurity,” in *SoutheastCon 2017*, pp. 1–6, Charlotte, NC, USA, 2017.

[13] J. Hu, Z. Xu, D. Ma, and J. Yang, “Research of webshell detection based on decision tree,” *Journal of Network New Media*, vol. 6, 2012.

[14] Z. Meng, R. Mei, T. Zhang, and W. P. Wen, “Research of Linux WebShell detection based on SVM classifier,” *Netinfo Security*, vol. 5, pp. 5–9, 2014.

[15] B. Hu, “Research on webshell detection method based on Bayesian theory,” *Science Mosaic*, vol. 6, pp. 66–70, 2016.

[16] Y. Fang, Y. Qiu, L. Liu, and C. Huang, “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, Chengdu, China, 2018.

[17] H. Cui, D. Huang, Y. Fang, L. Liu, and C. Huang, “Webshell detection based on random forest–gradient boosting decision tree algorithm,” in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pp. 153–160, Guangzhou, China, 2018.

[18] J. Ramos, “Using TF-IDF to determine word relevance in document queries,” in *Proceedings of the first instructional conference on machine learning*, pp. 29–48, 2003.

[19] T. Li, C. Ren, Y. Fu, J. Xu, J. Guo, and X. Chen, “Webshell detection based on the word attention mechanism,” *IEEE Access*, vol. 7, pp. 185140–185147, 2019.

[20] “PECL:: package:: vld,” 2020, http://pecl.PHP.net/package/vld.

[21] “PHP: PHPdbg-Manual,” 2020, http://www.PHP.net/PHPdbg.

[22] W. B. Cavnar and J. M. Trenkle, “N-gram-based text categorization,” in *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, vol. 161175, 1994.

[23] I. Neamtiu, J. S. Foster, and M. Hicks, “Understanding source code evolution using abstract syntax tree matching,” in *Proceedings of the 2005 international workshop on Mining software repositories - MSR '05*, pp. 1–5, 2005.

[24] “nikic/PHP-Parser: a PHP parser written in PHP,” 2020, https://github.com/nikic/PHP-Parser.

[25] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, pp. 278–282, Montreal, QC, Canada, 1995.

[26] 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, no. 1, 2019.

[27] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (ROC) curve,” *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[28] “D shield,” http://www.d99net.net/.

[29] BAIDU, “WEBDIR+webshell detector,” https://scanner.baidu.com.

[30] “php-malware-finder,” https://github.com/jvoisin/php-malware-finder.

[31] “SHELLPUB,” https://ml.shellpub.com/.