A Web Shell Detection Method Based on Multiview Feature Fusion

Tiantian Zhu 1, Zhengqiu Weng 2,3,* , Lei Fu 4,* and Linqi Ruan 5

1 College of Computer Science & Technology, Zhejiang University of Technology, Hangzhou 310023, China; ttzhu@zjut.edu.cn
2 Department of Information Technology, Wenzhou Polytechnic, Wenzhou 325035, China
3 Institute of Computer Software, Zhejiang University of Technology, Hangzhou 310023, China
4 College of Mechanical Engineering, Zhejiang University of Technology, Hangzhou 310023, China
5 Department of Computer Science and Information Security, Zhejiang Police College, Hangzhou 310053, China; ruanlinqi@zju.edu.cn
* Correspondence: derisweng@hotmail.com (Z.W.); fulei@zjut.edu.cn (L.F.); Tel.: +86-571-8795-3851 (L.F.)

Received: 9 July 2020; Accepted: 7 September 2020; Published: 9 September 2020

Abstract: Web shell is a malicious script file that can harm web servers. Web shell is often used by intruders to perform a series of malicious operations on website servers, such as privilege escalation and sensitive information leakage. Existing web shell detection methods have some shortcomings, such as viewing a single network traffic behavior, using simple signature comparisons, and adopting easily bypassed regex matches. In view of the above deficiencies, a web shell detection method based on multiview feature fusion is proposed based on the PHP language web shell. Firstly, lexical features, syntactic features, and abstract features that can effectively represent the internal meaning of web shells from multiple levels are integrated and extracted. Secondly, the Fisher score is utilized to rank and filter the most representative features, according to the importance of each feature. Finally, an optimized support vector machine (SVM) is used to establish a model that can effectively distinguish between web shell and normal script. In large-scale experiments, the final classification accuracy of the model on 1056 web shells and 1056 benign web scripts reached 92.18%. The results also surpassed well-known web shell detection tools such as VirusTotal, ClamAV, LOKI, and CloudWalker, as well as the state-of-the-art web shell detection methods.

Keywords: web shell detection; multiview feature fusion; feature selection; large-scale experiments; machine learning

1. Introduction

Web shell is a kind of malicious script uploaded by an attacker to a web server. Attackers often upload well-crafted web shell scripts through SQL injection, file inclusion/upload vulnerabilities, and cross-site scripting attacks. The purpose of web shells is to keep persistent access to infected computers/servers in order to perform a series of malicious exploits, such as system sensitive command execution, stealing and tampering with user data, and modifying the website’s homepage. Meanwhile, web shell is also an important part of advanced persistent threat (APT) attacks [1], which can cause significant damage to governments and large enterprises. Therefore, it is very important to design a reasonable and effective web shell detection method.

Although the academic and industrial communities have proposed a series of detection methods and defense solutions for web shell (i.e., auditing the source code before the program is released, real-time monitoring while the program is running), in the enterprise setting, the operating system or application program may not be updated in time, and the unknown software from third-parties can be
downloaded unconsciously. It is easy for attackers to use these vulnerabilities and upload web shells for their purposes. If the analyst can promptly find the web shell after the incident and delete/isolate it immediately, it can also effectively prevent the attacker from further harmful actions. However, existing web shell detection methods face a series of challenges. First, the behavior of an attacker using web shells for C&C (command & control) connection interaction can be disguised, and the attacker can easily bypass network-traffic-based detection using infrequent interactions [2–5]. For example, attackers try to lengthen the front of the attack and execute the required command only after a certain period of time, which makes the steps of these malicious acts in different periods look similar to normal ones. On the contrary, if a dynamic method is adopted for context detection, the storage of a massive amount of log data and the overhead of the required memory will be a big disaster. In this paper, the dynamic detection method will not be considered. Secondly, a web server often contains a large number of scripts, such as function scripts and plug-in scripts. Furthermore, the content of web scripts may change frequently over time (e.g., the file signature characteristics of a script file containing the current time will continue to change dynamically). In this case, the web shell detection method based on file signatures [6] will be inapplicable. Moreover, web shell is usually written in high-level languages such as PHP, ASP, JSP, and Python. Although the regular matching method [7] can identify some of the web shells, the lexical and syntactic features of high-level languages cannot be fully reflected by regular expressions.

In response to the above challenges, this paper proposes a web shell detection method based on multiview feature fusion. The main contribution of this paper is as follows:

1. A multiview feature fusion mechanism is presented, which can extract lexical features, syntactic features, and abstract features from PHP scripts. Features that can effectively represent web shells are integrated and extracted from multiple levels.
2. In order to determine the features that are reasonable for classification, a Fisher scoring mechanism is used to rank the importance of the features, and then the best features for subsequent classification are determined through experiments.
3. The use of optimized SVM for effective classification of malicious web shell has achieved satisfactory classification accuracy on large-scale data sets, and the overall performance is better than the state-of-the-art web shell detection tools and methods.

The remainder of this paper is organized as follows: Section 2 surveys the relevant work, Section 3 covers the methodology in detail, Section 4 presents the overall evaluation of our method, and Section 5 concludes our work.

2. Related Work

In order to effectively detect and combat web shell, the existing web server’s security response strategy can be divided into the following three types: pre-event detection strategy, mid-event detection strategy, and post-event detection strategy.

The pre-event detection strategy refers to the vulnerability scanning and vulnerability mining of the web server when the attack does not occur. Huang et al. [8] proposed a framework to generate test cases to reveal web server vulnerabilities. Son et al. [9] proposed a semantic-based method to identify PHP web applications with vulnerabilities (for example, the presence of infinite loop features, lack of authorization check functions). In addition, existing works [10–12] use static program analysis to detect web applications that are vulnerable to SQL injection and XSS (i.e., cross site scripting [13]) attacks. In addition to the above research, there is a series of web vulnerability scanning tools, such as Nikto2 [14] and WebInspect [15], which are widely used by the industry. Pre-event detection can only help the analyst understand the vulnerabilities in the web server system or application software. However, due to the complexity and diversity of the methods used by attackers, just scanning and repairing the vulnerabilities does not prevent the upload of suspicious code. It is also hard to reveal the hazards that will occur.
Correspondingly, the method of detection in the event (mid-event detection strategy) has been proposed by many researchers, that is, the real-time detection of behavior, namely, the attacker is uploading the web shell or using vulnerabilities or software vulnerabilities to cause harm to the web server. For example, Almgren et al. [16] developed a lightweight online attack detection tool to monitor the behavior of web servers in real-time. Kruegel et al. [17] established a statistical model to describe normal network requests and identified obvious deviations as abnormal. Robertson et al. [18] used multiple features to detect abnormal web behavior. The difficulty of anomaly detection technology lies in how to construct normal activity labels and how to design statistical algorithms so as to avoid marking normal operations as suspicious ones or ignoring real intrusion. In fact, anomaly detection technology will introduce a large number of false alarms while reducing false negatives, making it difficult to apply to real scenarios. In this regard, hybrid detection technology was proposed by Ko et al. [19], who developed a language framework through the analysis of program execution sequence and concurrent program grammatical events and finally customized event tracking strategy for real-time web intrusion detection. The experimental results of Prem et al. [20] proved that the strategy-based method is superior to a single feature detection or anomaly detection method. Hossain et al. [21] proposed an application-independent strategy to detect advanced persistent threat attacks represented by web shell by extracting origin information from audit data to improve detection level and used a custom IP address whitelist for intrusion screening; however, for a complex web server environment such as Linux/Windows, it seems to be powerless. Most of the mid-event detection strategies are based on prior knowledge, and there are problems such as high false positives and difficulty in customizing mining rules.

Although both pre-event detection strategy and mid-event detection strategy can avoid the harm of web shell to a certain extent, there is still no guarantee that all vulnerabilities and attacks will be completely avoided. Assuming that analysts can find web shell scripts in time and take further measures such as deletion and isolation after the incident, they can also effectively prevent attackers from further harmful actions. Therefore, it is necessary to design effective post-event detection strategies and mitigation measures. Starov et al. [22] analyzed a large number of real web shells to reveal their functions and studied how attackers use web shells by establishing “honeypots”. However, it may take a lot of time to configure the “honeypot” and obtain the corresponding attack intentions. Therefore, it is hard to ensure the real-time ability and effectiveness of web shell detection. In addition, some web shells have a “honeypot” detection function, which will automatically disable certain malicious behaviors in the “honeypot” to fight the detection engine. Tian et al. [5] proposed a web shell detection method based on network traffic, which uses convolutional neural networks to monitor and model the web shell runtime traffic. This method has two disadvantages. First, if the attacker stretches the front of the attack and executes the required command only once in a while, and the command self is proposed as normal behavior, this can undoubtedly bypass this detection scheme. Second, running web shells in real-time may cause key nodes/data of the system to be destroyed, and the overall risk is high. The web shell detection method proposed in this paper is based on the static detection of different characteristics of web shell, which can effectively avoid the above defects. Tu et al. [7] and Cui et al. [23] use regular matching and keyword feature matching to identify web shells. Although suspicious scripts can be effectively identified to a certain extent, web shells are usually written in high-level languages. The lexical and syntactic features cannot be fully reflected by regular expressions. The method proposed in this paper comprehensively considers the lexical, syntactic, and abstract features existing in such high-level languages (especially the PHP language) and proposes a heuristic ranking method of features based on the Fisher score, which can detect web shell accurately and efficiently.

3. Methodology

The overview of web shell detection methods is given in Section 3.1. The data preprocessing (including denoising, deredundancy, and clustering by various families) is introduced in Section 3.2.
Sections 3.3 and 3.4 describe how to extract and select representative features. The modeling and classification method is discussed in Section 3.5.

3.1. Overview

The method proposed in this paper includes the following four steps, as shown in Figure 1:

1. Use crawlers to collect a large number of web shell scripts on github [24–28]. Perform preprocessing such as denoising, deredundancy, and clustering by various families;
2. Analyze and extract lexical features, syntactic features, and abstract features from web shell;
3. Use Fisher score to rank the importance of the features, and then the best features for subsequent classification are determined through experiments;
4. Given the selected features, train a classifier that can distinguish web shells from benign web scripts by SVM.

![Figure 1. The overview of our method.](image)

3.2. Data Preprocessing

After collecting the relevant web shells from github, all the PHP scripts are extracted first; a total of 1489 PHP web shells were obtained. After that, various techniques for data preprocessing are adopted to improve the quality and formalization of the data samples. Data preprocessing mainly includes the following three categories:

1. **Denoise**.

   First, a script whose length exceeds a certain value is considered. Existing work [22] shows that scripts that are too large, often contain a lot of semantically irrelevant content and noise. We removed scripts with a size of more than 3 MB (it was found through extensive analysis that scripts exceeding this size generally cannot be effectively run) as noise, and a total of 1208 effective scripts were obtained. In addition, some PHP scripts will have a lot of obfuscation and BASE64 (a group of binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation [29]) encoding, which makes the detection method based on static analysis useless. In order to get a clean dataset, UnPHP technology [30] was used to deobfuscate the obfuscation code and BASE64-encoded web shells, and it has also been used in previous work for web shell analysis [21,31]. Finally, 1192 web shells that could be effectively run without timeout were retained.

2. **Deredundancy**.

   In order to remove duplicate web shell scripts, all the scripts are standardized first (i.e., all comments and blank lines are deleted). Secondly, the term frequency–inverse document frequency (TF-IDF) [32] model is used to calculate the text similarity, which is widely adopted in previous work [33]. Additionally, only one copy of all scripts, with a similarity exceeding 95%, is retained. In the end, 1056 effective and executable script files were retained.
(3) Clustering by various families.

Wu et al. [3] divided web shells into three categories according to the function and size of the scripting language: Big Trojan, One Word Trojan, and Small Trojan. However, in this paper, we only considered the Big Trojan for the following two reasons:

(1) Small Trojan usually has an upload function only. Attackers generally first obtain upload permission through the Small Trojan and then upload the Big Trojan to the website to perform key functions. The source of malicious behavior is Big Trojan.

(2) One Word Trojan needs to be used in conjunction with other tools to perform malicious behavior. Additionally, One Word Trojan is often used at the beginning of the attack. Unlike the Big Trojan, One Word Trojan is rarely used for persistence attacks.

There are a series of variations between web shells. For example, the c99 web shell is renamed c999 after adding the privilege escalation function. At the same time, some function modules in c99 can be found in the Fx29 web shell. It can be found that there is an inheritance relationship between the functional modules between them. Therefore, the idea of transitive closure [34] is used to cluster the similar web shells into one family. That is, if scripts m1 and m2 have a part of the same functional modules and scripts m2 and m3 have a part of the same functional modules, it can be inferred that m1, m2, and m3 are all subordinate to one family. This transfer feature makes it possible to get information about all members of the same family. At the same time, in this process, the corresponding function of the web shell in a family is gradually clear.

To ensure the utility of the provided method, 1056 effective web shells were divided into seven different families, with a clustering threshold of 30%. That is, if 30% of the functional modules are identical between the two web shells, they can be clustered into one family. The final clustering results are shown in Table 1.

Table 1. The final clustering results of 1056 web shells, which are based on a clustering threshold of 30%.

| Family Name | The Number of Web Shells |
|-------------|--------------------------|
| c99         | 326                      |
| r57         | 247                      |
| WSO         | 112                      |
| B374k       | 106                      |
| NST         | 98                       |
| NCC         | 84                       |
| Crystal     | 83                       |

3.3. Feature Extraction

Feature extraction will be performed after getting a clean data set. Lexical features, syntactic features, and abstract features are considered based on the semantic content as follows:

(1) Lexical features.

Lexical features refer to the way (or frequency) of certain words used in the script. We observe that in malicious web shells, since the attacker needs to accept various commands through the web shell to interact with the victim server, a large number of global variables are needed, and their parameters are combined to perform predefined activities. Since global variables are the main way to provide external data to PHP scripts, malicious web shells tend to use various highly parameterized global variables. Therefore, the number of global variables that can represent information that is received is extracted as features. These features contain $\_GET$ (F1), $\_POST$ (F2), $\_COOKIE$ (F3), $\_REQUEST$ (F4), $\_FILES$ (F5), and $\_SESSION$ (F6). Here, Fx is used to represent the feature.
(2) Syntactic features.

Syntactic features refer to the expressions used by attackers when writing web shell scripts. Before the attacker uploads the web shell, they cannot clearly know the detailed configuration information of the corresponding platform, such as which services are opened by the web server and what the password corresponding to a certain mailbox is. This requires the corresponding web shell to automatically adapt to various operating systems and automatically try to obtain the permissions of related software. For this, the following types of syntactic features are considered.

- The proportion of conditional statements, which represents the percentage of conditional statements in all statements in the script, including if (F7), elseif (F8), else (F9), and case (F10).
- The proportion of looping statements, which represents the percentage of loop statements in all statements in the script, including for (F11), while (F12), and foreach (F13).

(3) Abstract features.

Abstract features represent other relevant features besides lexical features and syntactic features. The most important part is the sensitivity function matching degree, which represents the usage of some keywords in the PHP language, such as the disguise execution function (eval), the file acquisition function (wget, curl, lynx, get, fetch), the reverse connection function (perl, python, gcc, chmod, nohup, nc), and the information collection function (uname, id, ver, sysctl, whoami, $OSTYPE, pwd). These keywords are often used by attackers in web shells to perform some suspicious behavior. This paper considers the existence of the following four functions as abstract features: disguise execution function (F14), file acquisition function (F15), reverse connection function (F16), and information collection function (F17).

In addition, three common features of the maximum length of words in the script (F18), the maximum length of lines in the PHP source code (F19), and information entropy (F20) are also added [35].

In summary, a total of 20 features related to web shell judgment were obtained through feature extraction.

3.4. Feature Selection

During the machine learning phase, features not only affect the convergence speed of the model but also determine the accuracy of the final classification. Further effective feature selection based on feature extraction can reduce the number of features to avoid overfitting and also allow the analyst to have an intuitive understanding of the pros and cons of each feature. The Fisher score has been proven to be a good method for feature selection in related work [36]. The theoretical basis is that the variance of a good feature and all corresponding values of the same class should be as small as possible.

Suppose \( u_f \) represents the average value of the \( i \)-th feature on the dataset, \( u_{k,i}^f \) represents the average value of the \( i \)-th feature in the \( k \)-th category, \( n_k \) represents the number of samples in the \( k \)-th category, \( f_{i,j,k} \) represents the value of the \( j \)-th position of the \( i \)-th feature in the \( k \)-th category. Then, the Fisher score of the \( i \)-th feature can be expressed as

\[
F(f_i) = \frac{\sum_{k=1}^{\xi} n_k (u_{k,i}^f - u_f)^2}{\sum_{k=1}^{\xi} \sum_{j=1}^{\gamma} f_{i,j,k}^k (f_{i,j,k}^k - u_{k,i}^f)^2}
\]  

(1)

The larger the \( F(f_i) \) is (it means that the variance of all corresponding values of the corresponding feature in the same class will be smaller), the better the feature is.
3.5. Modeling and Classification

After data preprocessing, feature extraction, and selection, it is necessary to construct a corresponding model for malicious web shell judgment. Aiming at the classification goal, selecting the appropriate classifier in combination with the actual data features will help to improve the discriminative ability of the model. Considering the nonlinear characteristics of the web script feature boundary, the multidimensional characteristics of the features, and the dependency between the web shell features in various dimensions, in this paper, support vector machine (SVM) is chosen as the classifier for the following reasons:

First, due to the complexity of web shell, many normal script files will also behave similarly to web shells, so malicious web shell is not linearly separable when using features to classify it. SVM has multiple kernel functions (e.g., polynomial kernel function and radial basis function) to handle nonlinear separable problems.

Second, 16-dimensional (described in Section 4.3) and multidimensional feature vectors are totally extracted, and related work [36] has shown that SVM has achieved satisfactory results in multidimensional feature classification.

4. Evaluation

The experimental setup and evaluation metric are proposed in Sections 4.1 and 4.2, respectively. The performance of feature selection is introduced in Section 4.3. Section 4.4 discusses how to pick up the best parameters for model training. Sections 4.5 and 4.6 summarize the overall performance and the advantages of the proposed methods.

4.1. Experimental Setup

The malicious web shell sample in this paper mainly comes from github [24–28], and after preprocessing and clustering, there were 1056 effective web shells (as detailed in Section 3.2) in total. For the purpose of binary classification, 1056 normal PHP scripts were downloaded directly from github, which are prestigious and of high rank, based on the number of stars (i.e., the number of stars is higher than 500 on github), for training and validation. The type of selected samples contained Wordpress, Joomla, VBB, Oscommerce, PhpNuke, and Phpbb.

4.2. Evaluation Metric

In terms of accuracy, the following evaluation matrixes are defined:

**True positive (TP)**, the malicious web shells are accurately marked.

**False positive (FP)**, the benign web scripts are marked as web shells.

**True negative (TN)**, the benign web scripts are accurately marked.

**False negative (FN)**, the malicious web shells are marked benign ones.

In terms of classification accuracy, the following indicators are used to judge the quality of the model. The indicators are defined based on experience, which has been widely used in previous work [37–39].

\[
\text{Pmalicious} = \frac{TP}{TP + FP} \\
\text{Rmalicious} = \frac{TPR = TP}{TP + FN} \\
\text{Fmalicious} = \frac{2 \times \text{Pmalicious} \times \text{Rmalicious}}{\text{Pmalicious} + \text{Rmalicious}} \\
\text{Pbenign} = \frac{TN}{TN + FN} \\
\text{Rbenign} = \frac{TN}{TN + FP} \\
\text{Fbenign} = \frac{2 \times \text{Pbenign} \times \text{Rbenign}}{\text{Pbenign} + \text{Rbenign}} \\
\text{FPR} = \frac{FP}{TN + FP}
\]
Accuracy = \( \frac{TP + TN}{TP + FP + FN + TN} \)

Pmalicious, Rmalicious, and Fmalicious represent the precision, recall, and F1-score of malicious web shells, respectively. Pbenign, Rbenign, and Fbenign represent the precision, recall, and F1-score of benign web scripts, respectively. In addition, the receiver operating characteristic (ROC) curve is used to represent the overall recognition performance of our method. The ROC curve shows the relationship between the true-positive rate (TPR) and the false-positive rate (FPR) at different classification thresholds, \( \theta \). In particular, the area under the curve (AUC) is used to represent the area enclosed by the ROC curve and the coordinate axis.

### 4.3. The Experiment of Feature Selection

In this paper, the Fisher score is used for feature selection. Before feature selection, all types of features are normalized to the interval of 0–1 to eliminate the impact between different dimensions. In this paper, the min–max normalization method is used, as follows, in Equation (2), where the \( Z_i \) is the normalized result, \( F_{\text{max}} \) is the maximum data of the target feature, and \( F_{\text{min}} \) is the minimum data of the target feature, respectively.

\[
Z_i = \frac{F_i - F_{\text{min}}}{F_{\text{max}} - F_{\text{min}}}
\]

For each type of normalized feature, the value of the Fisher score and the final classification accuracy (the ratio of the number of samples in the training set to that in the test set is 3:2.) is calculated. The relationship between the Fisher Score and the feature’s impact on the final classification accuracy for all features is shown in Figure 2.

![Figure 2](image-url)  
**Figure 2.** The relationship between the Fisher score and the feature’s impact on the final classification accuracy for all features.

It can be seen from Figure 2 that if a certain feature does not represent the user’s unique behavior well, then its Fisher score will be close to 0, and the accuracy when using this feature for classification alone will become very low. Conversely, if a feature is very representative, then its Fisher score will be much greater than 0, and the accuracy when using this feature for classification alone will become better. By comparison, it is found that the most important features include four types of functions (i.e., disguise execution function, file acquisition function, reverse connection function, and information collection function) and various highly parameterized global variables (i.e., \$_GET, \$_POST, and \$_COOKIE), while the useless features are “if” and “else” in the conditional sentences.

We also use SVM to demonstrate the effect of the number of selected features. The best parameters of SVM are adopted here (C = 0.8 and \( \gamma = 0.03 \), which are described in Section 4.4). The radial basis function (RBF) kernel is chosen in the experiment. The ratio of the number of samples in the training set
to that in the test set is 3:2 (in particular, 60% samples from each web shell family are selected randomly as malicious ones; similar methods are used to select benign samples). Each result is based on 100 independently repeated trials. Figure 3 shows the accuracy results by extracting different features through SVM. It was observed that the classification accuracy of the SVM classifier achieved 92.1% when the top 16 ranked features were selected by the Fisher score method. It was found that when the number of selected features was smaller than 16, the final accuracy increased as the number increased, and when the number of selected features was larger than 16, the final accuracy decreased as the number increased. Finally, the top 16 ranked features were chosen for our web shell detection.

![Figure 3. Classification accuracy results with different numbers of selected features. Training result by SVM model. Each result is based on 100 independently repeated trials.](image)

To further investigate the effectiveness of feature selection, the Fisher score method was evaluated by adopting different ratios of training and testing sets. The proportions of the training set are 20%, 30%, 40%, 50%, 60%, and 70% (in particular, samples of the above ratio from each web shell family were randomly selected as malicious samples). After determining the size of the training set, the remaining data were used for testing. In order to eliminate accidental factors, 100 independent trials were carried out for different ratios. Figure 4 depicts the relationship between different percentages of samples for training and classification accuracy. It was observed that classification accuracy with the Fisher score feature selection was better than that without feature selection. In Figure 4, when the training percentage was selected as 20%, the classification accuracy was less than 80%, which demonstrated that our SVM model (the experiment of SVM training is discussed in Section 4.4) was not well trained. When the proportion of the training set was between 20% and 60%, it was obvious that the classification accuracy was enhanced with the increased percentage of the training sets. When the percentage was set to 70% (larger than 60%), the final classification accuracy tended to be stable, which means that 60% of the training data is enough to build a robust model.

Moreover, in order to further illustrate that the selected features can accurately distinguish between normal scripts and malicious scripts, another experiment was conducted by using the dataset of 2112 samples (1056 normal samples and 1056 malicious web shell samples) and 16 features selected by the Fisher score.

For each dimension data of the feature vector, the center point of various scripts (normal ones and web shells) was calculated first, which were denoted as $c_n$ and $c_m$, respectively. Then, the average radius of the Euclidean distance from the corresponding dimension data to the center points $c_n$ and $c_m$ in the data samples of the normal and malicious scripts were recorded as $r_n$ and $r_m$, respectively. Finally, the Euclidean distance between the center points of the normal and malicious scripts was calculated and recorded as $d_{cm}$. For the 16 dimensions (features), the final results are shown in Table 2. It can be seen from Table 2 that the values of $r_n$ and $r_m$ are much smaller than $d_{cm}$, which shows that the features extracted in this paper can distinguish between normal scripts and web shells.
Figure 4. The relationship between different percentages of samples for training and classification accuracy, with/without feature selection (Fisher score).

Table 2. Cluster separability.

| Parameter | Value  |
|-----------|--------|
| $r_n$     | 2.604  |
| $r_m$     | 3.025  |
| $d_{cm,n}$| 17.531 |

4.4. Model Optimization

In Section 4.3, it can be found that 60% of the training data is enough to build a robust model. In this paper, the dataset is divided into 6:4; that is, 60% of the web shell and normal scripts is used to train the model, and the remaining 40% is used for testing. In order not to lose generality, this paper randomly extracts 60% of the scripts in all family web shells and mixes them with the random samples of 60% of the normal samples as a labeled training set. The advantage of doing this is to ensure that the characteristics of each family can be learned by the classifier. During training, SVM uses the radial basis kernel function because the web shell classification in this paper is a multidimensional nonlinear problem; the use of the radial basis kernel function can quickly and efficiently build a model.

In order to optimize the model, the grid search is used to determine the best parameters for SVM training. In the SVM training phase, the two main parameters that need to be determined are $C$ and $\gamma$, where parameter $C$ is the penalty coefficient. If $C$ becomes larger, it means that the intensity of the punishment becomes larger, resulting in a situation where the model is not flexible enough and the generalization ability becomes weak. Conversely, if $C$ becomes smaller, then the punishment strength will naturally become smaller, and the model is prone to underfitting. Another parameter $\gamma$ determines the distribution of the data after mapping into a new feature space. Similar to the case of parameter $C$, if $\gamma$ is too large, the model does not learn the content of the vector, but only remembers the support vector itself, which leads to a decline in the generalization ability and is prone to overfitting. If $\gamma$ is too small, the data distribution in the new feature space will be too smooth, resulting in underfitting. In this paper, the SVM convergence coefficient is fixed at 0.01, the parameter $\gamma$ is fixed at 0.01, and the change of parameter $C$ and the final classification accuracy of the model is shown in Figure 5. It can be seen from Figure 5 that when the value of the parameter $C$ is 0.8, the classification effect is the best. Similarly, when the SVM convergence coefficient is fixed at 0.01 and the parameter $C$ is fixed at 0.8, the change in parameter $\gamma$ and the final classification accuracy of the model is shown in Figure 6. It can
be seen from Figure 6 that when the value of the parameter $\gamma$ is 0.03, the classification effect is the best. In summary, $C = 0.8$ and $\gamma = 0.03$ are used in this paper.

![Figure 5](image_url)  
**Figure 5.** The relationship between the parameter $C$ and the final classification accuracy of the model.

![Figure 6](image_url)  
**Figure 6.** The relationship between the parameter $\gamma$ and the final classification accuracy of the model.

### 4.5. Overall Performance

Under the optimal parameters, 100 independently repeated experiments were conducted, the average values of TP, FP, TN, and FN finally obtained are shown in Table 3, and the final average values of $P_{\text{malicious}}$, $R_{\text{malicious}}$, $F_{\text{malicious}}$, $P_{\text{benign}}$, $R_{\text{benign}}$, and $F_{\text{benign}}$ are shown in Table 4.

**Table 3.** The average values (with feature selection) of TP, FP, TN, and FN under 100 independently repeated experiments.

|                     | Web Shells | Benign Scripts |
|---------------------|------------|----------------|
| **Web shells**      | TP = 402   | FN = 20        |
| **Benign scripts**  | FP = 46    | TN = 376       |

**Table 4.** The average classification results of 100 independently repeated experiments.

| Categories           | $P_{\text{malicious}}$ | $R_{\text{malicious}}$ | $F_{\text{malicious}}$ | $P_{\text{benign}}$ | $R_{\text{benign}}$ | $F_{\text{benign}}$ |
|----------------------|-------------------------|------------------------|------------------------|---------------------|---------------------|---------------------|
| With feature selection| 89.73%                  | 95.26%                 | 92.41%                 | 94.05%              | 89.10%              | 91.51%              |
| Without feature selection | 85.52%              | 89.57%                 | 87.50%                 | 89.05%              | 84.83%              | 87.48%              |
As can be seen from the above table, after the 16 most important features were selected, the final recognition results were significantly improved. By deploying the Fisher score, the average $P_{\text{malicious}}$, $R_{\text{malicious}}$, $F_{\text{malicious}}$, $P_{\text{benign}}$, $R_{\text{benign}}$, and $F_{\text{benign}}$ from 85.52%, 89.57%, 87.50%, 89.05%, 84.83%, and 87.48%, were increased to 89.73%, 95.26%, 92.41%, 94.05%, 89.10%, and 91.51%. It can be seen from Table 4 that the recognition rate of web shell ($R_{\text{malicious}}$) is 95.26%, and the recognition rate of normal script ($R_{\text{benign}}$) is 89.10%. The final classification accuracy of the model is 92.18%. Considering that FN (the malicious web shells marked benign ones) in the web shell detection system is often more critical than FP (that is, the benign web scripts marked as web shells), in practical applications, the classification threshold can be adjusted. For example, on a web server with higher security, the classification threshold can be lowered so that the recognition rate of benign scripts can be appropriately reduced to improve the detection rate of web shells.

To get the ROC curve, we set the threshold value $\theta$ from 0 to 1; the step size is 0.01. Under different classification thresholds $\theta$, the average values of TPR and FPR of both web shells and benign scripts were calculated, and the resulting ROC curves with/without feature selection are shown in Figure 7. The resulting AUC values are 0.9257 and 0.8940, respectively. From the usability point of view, the increase in FPR will greatly increase the workload of the web administrator (because the administrator needs to conduct a second screening of real web shells based on a large number of alarms). When it comes to security, the reduction in TPR will cause a greater potential threat to the web platform. Due to the high AUC (with feature selection), our method has sufficient optimization space to balance security and usability in a real-world scenario.

![Figure 7. Receiver operating characteristic (ROC) curve of the system.](image)

### 4.6. Comparison with Existing Detectors

In order to reflect the advanced nature of our proposed web shell detection method, it is necessary to compare this method with other existing tools, such as VirusTotal [40], ClamAV [41], LOKI [42], and CloudWalker [43].

VirusTotal can analyze suspicious files to detect types of malware and automatically share them with the security community. It contains, in total, sixty-nine detection engines. In the experiment, as long as any one of the sixty-nine detection engines issues an alarm, it will be considered that the web shell was successfully detected. ClamAV is an open-source antivirus engine for detecting trojans, viruses, malware, and other malicious threats. For any file that needs to be detected, there are two kinds of detection results: one is benign and the other is infected. We extracted all web shells that ClamAV judged to be infected as TP. LOKI offers a simple way to scan malicious files such as MD5/SHA1/SHA256 hashes. The detection results of LOKI include the following four types (arranged according to the degree of maliciousness): benign, notices, warnings, and alerts. We regard notices,
warnings, and alerts as successful detections of web shell. CloudWalker is an open source server security management platform launched by Changting. According to the project plan, it will gradually cover various functions such as server asset management, threat scanning, and web shell detection. At present, the detection results are divided into five levels (one to five). The higher the level, the higher the probability of detecting web shells. In the experiment, the web shell will be judged as a score greater than two.

Furthermore, this paper compares the other two state-of-the-art methods (Tu et al. [7] and Fang et al. [44]) to show the advantage of our detector. Tu et al. [7] utilized optimal threshold values to identify web shells. In the experiment, the best parameters (the malicious signature is set to 0, the malicious function is set to 13, and the longest word is set to 83) defined in [7] are used. Fang et al. [44] considered both the static features (longest string, information entropy, index of coincidence, signature, and blacklist keywords) and PHP opcode sequence features (using the PHP VLD extension) to detect web shells. The above two state-of-the-art methods were reimplemented by using their tools and parameters. Table 5 shows the detection rate (TPR) of various advanced web shell detectors for web shells of different families. All the results are based on the average of 100 independently repeated experiments. It can be seen from the table that the overall detection rate of our method is better than other detectors. The detection rates of our method for web shells of c99, r57, WSO, B374K, NST, NCC, and Crystal families are 96.92%, 98.99%, 88.89%, 100%, 97.44%, 85.29%, and 87.88%, respectively. For the existing tools, the detection rates of VirusTotal (second place) are only 92.31%, 93.94%, 88.89%, 97.62%, 92.31%, 52.94%, and 48.48%, respectively. Among them, the detection rate of web shells of c99 and r57 families by all types of detectors is generally high because these two types of web shells are the most common. For this reason, most detectors contain a prior knowledge of these two types of web shells (such as signatures and regular features) so as to achieve higher detection rates. At the same time, it was found that VirusTotal and CloudWalker can have a higher detection rate on the families of WSO and B374K than that of ClamAV and LOKI. The main reason is CloudWalker combines four different detection engines and VirusTotal is composed of sixty-nine different detection engines. The coverage of these two detectors is somewhat higher than that of other detectors. In addition, through comparison, it can be found that ClamAV, LOKI, and CloudWalker have a lower detection rate for web shells of the three families NST, NCC, and Crystal; the reason is that the web shells of these three families are not as popular as the previous ones, and the features are not well included in these detectors, which result in a low detection rate. Overall, the detection rates of VirusTotal, ClamAV, LOKI, and CloudWalker are 86.26%, 55.69%, 62.56%, and 73.46%, respectively. For the state-of-the-art methods, it can be found that the method proposed by Fang et al. is much better than that of Tu et al. The main reason is that (1) PHP opcode sequence features can represent the malicious web scripts effectively, and (2) the threshold defined by Tu et al. is easily compromised by sophisticated attackers.

Our method does not use the opcode sequence features because extra costs may be incurred to adopt the transformation tool (i.e., the PHP VLD extension used in [44]). Additionally, the detection rate of our method reaches 95.26%, which is much higher than other detectors.

Table 5. Comparison average in TPR results of various advanced web shell detectors and different web shell families. All the results are based on the average of 100 independently repeated experiments.

|        | c99    | r57    | WSO    | B374K  | NST    | NCC    | Crystal | Overall |
|--------|--------|--------|--------|--------|--------|--------|---------|---------|
| Our method | 96.92% | 98.99% | 88.89% | 100%   | 97.44% | 85.29% | 87.88%  | 95.26%  |
| VirusTotal | 92.31% | 93.94% | 88.89% | 97.62% | 92.31% | 52.94% | 48.48%  | 86.26%  |
| ClamAV   | 83.08% | 83.84% | 37.78% | 26.19% | 20.51% | 17.65% | 6.06%   | 55.69%  |
| LOKI     | 86.15% | 87.88% | 51.11% | 52.38% | 17.95% | 26.47% | 12.12%  | 62.56%  |
| CloudWalker | 95.38% | 93.94% | 91.11% | 90.48% | 15.38% | 17.65% | 6.06%   | 73.46%  |
| Tu et al. | 94.17% | 96.76% | 62.50% | 54.72% | 72.45% | 54.76% | 63.86%  | 79.92%  |
| Fang et al. | 96.31% | 87.57% | 94.64% | 93.40% | 83.67% | 82.14% | 85.54%  | 92.99%  |
5. Conclusions

Nowadays, web shell detection is important in the context of increasing levels of malicious attacks on web servers around the world. This paper proposes a web shell detection method based on multiview feature fusion.

The advantages of our work: First, lexical features, syntactic features, and abstract features that can effectively represent PHP web shells are integrated and extracted from multiple levels. Second, in order to determine the features that are reasonable for classification, a Fisher scoring mechanism is used to rank the importance of the features, and then the sixteen best features for subsequent classification are determined through experiments. Finally, the optimized SVM (C = 0.8 and γ = 0.03) is used to distinguish web shells from benign web scripts. Our method has achieved satisfactory classification accuracy on large-scale data sets. The total accuracy (combining all the web shells and benign scripts) is 92.18%, and 95.26% of web shells can be detected successfully, which is better than the state-of-the-art web shell detection tools and methods.

The disadvantages and future work: First, the coverage of our method needs to be improved. Our future work is to study the mechanism of web shells with other languages, such as JSP, ASP, and Python. Our goal is to find a general way to detect different types of web shells, e.g., using a system log [45] to detect web shells in real-time. Second, the feature engineering can be improved. The features used in this paper are mostly statistical-based. Our future work is to use deep neural networks to extract and select features automatically, e.g., using auto-encoding and decoding [46].

Author Contributions: Each author contributed extensively to the preparation of this manuscript. T.Z. and Z.W. designed the experiment; T.Z. and L.F. performed the experiments; L.R., T.Z. and Z.W. analyzed the data; T.Z. and Z.W. wrote the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This paper is sponsored by Wenzhou scientific research projects for underdeveloped areas (WenRenSheFa [2020] 61 (No.5)) and is also supported by major scientific research projects of Wenzhou Polytechnic (No.WZY2020001).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Compromised Web Servers and Web Shells. Available online: https://www.us-cert.gov/ncas/alerts/TA15-314A (accessed on 2 July 2020).
2. Yang, W.; Sun, B.; Cui, B. A Webshell Detection Technology Based on Http Traffic Analysis. In Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing; Kunibiki Messe, Matsue, Japan, 4–6 July 2018; Springer: New York, NY, USA, 2018; pp. 336–342. [CrossRef]
3. Wu, Y.; Sun, Y.; Huang, C.; Jia, P.; Liu, L. Session-Based Webshell Detection Using Machine Learning in Web Logs. Secur. Commun. Netw. 2019, 2019, 3093809. [CrossRef]
4. Zhang, H.; Guan, H.; Yan, H.; Li, W.; Yu, Y.; Zhou, H.; Zeng, X. Webshell traffic detection with character-level features based on deep learning. IEEE Access 2018, 6, 75268–75277. [CrossRef]
5. Tian, Y.; Wang, J.; Zhou, Z.; Zhou, S. CNN-Webshell: Malicious Web Shell Detection with Convolutional Neural Network. In Proceedings of the VI International Conference on Network, Kunming, China, 8–10 December 2017; Communication and Computing: New York, NY, USA, 2017; pp. 75–79. [CrossRef]
6. Kim, J.; Yoo, D.; Jang, H.; Jeong, K. WebSHArk 1.0: A Benchmark Collection for Malicious Web Shell Detection. JIPS 2015, 11, 229–238. [CrossRef]
7. Tu, T.D.; Guang, C.; Xiaojun, G.; Wubin, P. Webshell Detection Techniques in Web Applications. In Proceeding of the Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Hefei, China, 11–14 July 2014; pp. 1–7. [CrossRef]
8. Huang, Y.; Tsai, C.; Lin, T.; Huang, S.; Lee, D.T.; Kuo, S. A testing framework for Web application security assessment. Comput. Netw. 2005, 48, 739–761. [CrossRef]
9. Son, S.; Shmatikov, V. SAFERPHP: Finding Semantic Vulnerabilities in PHP Applications. In Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security, San Jose, CA, USA, 5 June 2011; pp. 1–13. [CrossRef]
10. Wassermann, G.; Su, Z. Sound and Precise Analysis of Web Applications for Injection Vulnerabilities. In Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, San Diego, CA, USA, 10–13 June 2007; pp. 32–41. [CrossRef]

11. Wassermann, G.; Su, Z. Static Detection of Cross-Site Scripting Vulnerabilities. In Proceedings of the 30th International Conference on Software Engineering (ACM/IEEE), Leipzig, Germany, 10–18 May 2008; pp. 171–180. [CrossRef]

12. Xie, Y.; Aiken, A. Static Detection of Security Vulnerabilities in Scripting Languages. In Proceedings of the USENIX Security Symposium, Vancouver, BC, Canada, 31 July–4 August 2006; pp. 179–192.

13. Cross Site Scripting. Available online: https://en.wikipedia.org/wiki/Cross-site_scripting (accessed on 2 July 2020).

14. Nikto 2. Available online: https://cirt.net/Nikto2 (accessed on 2 July 2020).

15. WebInspect. Available online: https://webinspect.updatestar.com/ (accessed on 2 July 2020).

16. Almgren, M.; Debar, H.; Dacier, M. A Lightweight Tool for Detecting Web Server Attacks; NDSS: San Diego, CA, USA, 2000. [CrossRef]

17. Kruegel, C.; Vigna, G. Anomaly Detection of Web-Based Attacks. In Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington, DC, USA, 27–30 October 2003; pp. 251–261. [CrossRef]

18. Robertson, W.; Vigna, G.; Kruegel, C.; Kemmerer, R.A. Using Generalization and Characterization Techniques in the Anomaly-Based Detection of Web Attacks; NDSS: San Diego, CA, USA, 2006.

19. Ko, C.; Ruschitzka, M.; Levitt, K. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-Based Approach. In Proceedings of the Symposium on Security and Privacy (Cat. No. 97CB36097), Oakland, CA, USA, 4–7 May 1997; pp. 175–187. [CrossRef]

20. Uppuluri, P.; Sekar, R. Experiences with Specification-Based Intrusion Detection. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Davis, CA, USA, 10–12 October 2001; pp. 172–189. [CrossRef]

21. Hossain, M.N.; Milajerdi, S.M.; Wang, J.; Eshete, B.; Gjomeno, R.; Sekar, R.; Stoller, S.; Venkatakrishnan, V.N. [SLEUTH]: Real-Time Attack Scenario Reconstruction from [COTS] Audit Data. In Proceedings of the 26th [USENIX] Security Symposium, Vancouver, BC, Canada, 16–18 August 2017; pp. 487–504. [CrossRef]

22. Starov, O.; Dahse, J.; Ahmad, S.S.; Holz, T.; Nikiforakis, N. No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells. In Proceedings of the 25th International Conference on World Wide Web, Montreal, Canada, 11–15 April 2016; pp. 1021–1032. [CrossRef]

23. Cui, H.; Huang, D.; Fang, Y.; Liu, L.; Huang, C. Webshell detection based on random forest–gradient boosting decision tree algorithm. In In Proceedings of the Third International Conference on Data Science in Cyberspace (DSC), Guangzhou, China, 18–21 June 2018; pp. 153–160. [CrossRef]

24. JohnTroony’s php-Web Shells Repository. Available online: https://github.com/JohnTroony/php-webshells (accessed on 2 July 2020).

25. Nikicat’s Web-Malware-Collection Repository. Available online: https://github.com/nikicat/web-malware-collection/tree/master/Backdoors/PHP (accessed on 2 July 2020).

26. Tennic’s Webshell Repository. Available online: https://github.com/tennc/webshell/ (accessed on 2 July 2020).

27. Webshell-Sample. Available online: https://github.com/ysrc/webshell-sample/ (accessed on 2 July 2020).

28. PHP-Backdoors. Available online: https://github.com/bartblaze/PHP-backdoors (accessed on 2 July 2020).

29. Base64. Available online: https://en.wikipedia.org/wiki/Base64 (accessed on 2 July 2020).

30. UnPHP, the Online PHP Decoder. Available online: http://www.unphp.net/ (accessed on 2 July 2020).

31. Zhang, Z.; Li, M.; Zhu, L.; Li, X. SmartDetect: A smart Detection Scheme for Malicious Web Shell Codes Via Ensemble. In Proceeding of the Learning, International Conference on Smart Computing and Communication, Tokyo, Japan, 10–12 December 2018; Springer: New York, NY, USA, 2018; pp. 196–205. [CrossRef]

32. TF-IDF. Available online: http://www.tfidf.com/ (accessed on 2 July 2020).

33. Zhu, J.; Yin, J.; Hou, F. A text similarity measurement combining word semantic information with TF-IDF method. Jisuanji Xuebao 2011, 34, 856–864. [CrossRef]

34. Zhu, T.; Gao, H.; Yang, Y.; Bu, K.; Chen, Y.; Downey, D.; Lee, K.; Choudhary, A.N. Beating the artificial chaos: Fighting OSN spam using its own templates. IEEE ACM Trans. Netw. 2016, 24, 3856–3869. [CrossRef]

35. Webshell Detector. Available online: https://github.com/chaitin/cloudwalker (accessed on 2 July 2020).
36. Zhu, T.; Qu, Z.; Xu, H.; Zhang, J.; Shao, Z.; Chen, Y.; Prabhakar, S.; Yang, J. RiskCog: Unobtrusive real-time user authentication on mobile devices in the wild. *IEEE Trans. Mobile Comput.* 2019, 19, 466–483. [CrossRef]

37. Zhu, T.; Weng, Z.; Song, Q.; Chen, Y.; Liu, Q.; Chen, Y.; Lv, M.; Chen, T. ESPIALCOG: General, Efficient and Robust Mobile User Implicit Authentication in Noisy Environment. *IEEE Trans. Mobile Comput.* 2020. [CrossRef]

38. Zhu, T.; Fu, L.; Liu, Q.; Lin, Z.; Chen, Y.; Chen, T. One Cycle Attack: Fool Sensor-based Personal Gait Authentication with Clustering. *IEEE Trans. Inf. Foren. Sec.* 2020. [CrossRef]

39. Zhu, T.; Weng, Z.; Chen, G.; Fu, L. A Hybrid Deep Learning System for Real-World Mobile User Authentication Using Motion Sensors. *Sensors* 2020, 20, 3876. [CrossRef] [PubMed]

40. Virustotal. Available online: https://www.virustotal.com/ (accessed on 2 July 2020).

41. ClamAV. Available online: https://www.clamav.net/ (accessed on 2 July 2020).

42. LOKI. Available online: https://www.nextron-systems.com/loki/ (accessed on 2 July 2020).

43. CloudWalker. Available online: https://github.com/chaitin/cloudwalker (accessed on 2 July 2020).

44. Fang, Y.; Qiu, Y.; Liu, L.; Huang, C. Detecting Webshell Based on Random Forest with Fasttext. In Proceedings of the International Conference on Computing and Artificial Intelligence, London, UK, 10–12 July 2018; pp. 52–56. [CrossRef]

45. Xiong, C.; Zhu, T.; Dong, W.; Ruan, L.; Yang, R.; Chen, Y.; Cheng, Y.; Cheng, S.; Chen, X. CONAN: A Practical Real-time APT Detection System with High Accuracy and Efficiency. *IEEE Trans. Depend. Secur.* 2020. [CrossRef]

46. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. *Stat* 2014, 1050, 10. [CrossRef]