Detecting Webshells Based On Random Forest With FastText

Extended Abstract†

|  |  |  |
| --- | --- | --- |
| Qiu Yaoyao Istituto Officina dei Materiali P.O. Box 1212 Italy gubbiotti@corporation.com | Huang Cheng Dipartimento di Fisica e Geologia P.O. Box 6221 Italy malago@affiliation.org | Steve Fin Dipartimento di Fisica e Scienze P.O. Box 5000 Italy fin@affiliation.org |
|  | Som Tacchi Dipartimento di Fisica e Geologia P.O. Box 6221 Italy tacchi@affiliation.org |

ABSTRACT

Web-based remote access Trojan (or webshell) is a kind of tool for network intrusion, which can be uploaded to a website to access web service management authority. Once attacker injected successfully, it can cause great damage so that it is crucial to detect webshell effectively. Webshells are flexible and changeable by using of obfuscation techniques, which compounds the difficulties of detecting. In this paper, we proposed a PHP webshell detection model based on a combination of fastText and random forest algorithm, which called FRF-WD. The PHP opcodes sequences as an important feature applied for webshell detection. The experimental results show that the model can provide high detection rate and low false alarm rate, which proved the feasibility and validity of the model.

CCS CONCEPTS

• **Security and privacy** → **Systems security**

KEYWORDS

webshell, webshell detection, fastText, random forest, opcode

1 INTRODUCTION

Nowadays, web applications and web services are ever increasingly replacing legacy applications, and as a consequence, widening attackers' exploitation opportunities [[1](#bib1)]. Webshell is a kind of backdoor program based on network, used by attackers with the intent to escalate and maintain persistent access on an already compromised web application. The common functionality includes but is not limited to shell command execution, code execution, database enumeration and file management. Web shells can be developed by different languages such as PHP, JAVA, ASP and PYTHON, etc. In this paper, we focused on PHP webshell because PHP is especially suited to web development so that made it the language of choice for web developers and web shells makers.

The traditional network-layer-only security controls such as firewalls and signature-based intrusion prevention and detection systems have little role to play in detecting webshells. Webshell as a script file has several static features and dynamic features. Neopi [[2](#bib1)] is a famous webshell detection tool, which calculates several static features such as longest string, index of coincidence, entropy, compression ratio and signature, and then gives a rank for all files and lists top ten with the highest rank. Afterwards, some researchers exploit these static features as inputs for machine learning models such like decision tree [[3](#bib1)] and support vector machine (SVM) [4]. In this paper, we investigated whether sequences of PHP opcodes are informative for detecting webshell. We focused on opcodes because opcodes (which are different from PHP opcodes but they are similar) are one of the most important malware features. Indeed, our experiments have shown that the sequences of PHP opcodes can represent a sort of signature of webshells, which are always available extractable from the file to be analyzed. Moreover, as a pseudo-static technique, the analysis has the advantage of fast and easy implementation.

We proposed a model based on the combination of fastText and random forest algorithm, which uses PHP opcode sequences and other statistical features of the webshells as the inputs. We experimentally applied the model to a dataset consisting of 6934 PHP benign files and 1587 PHP webshells. Our experimental results based on 10-fold cross validation showed that model performs very well, and accuracy stands at 99.23% with a recall of 97.65% and precision of 97.92%.

The rest of this paper is organized as follows: Section 2 provides the background of the problem, as well as a review of related work on webshell detection; Section 3 presents methods and techniques for PHP web shells detection; Section 4 details the experimental results of our model and we have a discussion; And finally, conclusions are presented in Section 5.

2 BACKGROUND

2.1 WebShells

Web-based remote access Trojans (or webshells) are small scripts designed to be uploaded onto production servers [[5](#bib1)]. Once infected, attackers could use the webshell to remotely access, perform privilege escalation attacks and pivot inside or outside the network. Based on webshells’ file size and functionality, they can be classified into three categories: one-sentence Trojan, mini Trojan, and full Trojan.

*2.1.1 One-sentence Trojans.* They are often just one line of code, most of them are obfuscated and deformed which are hard to be detected.

*2.1.2 Mini Trojan.* Compare with one-sentence Trojans, they have more functionalities such as file transfer and privilege escalation. Attackers use them as a springboard to bypass size restriction of upload file to upload full Trojan.

*2.1.3 Full Trojan.* A webshell with full intrusion functions, and it is much larger in size, has a friendly graphic user interface for file operations, command execution, and database connection. It normally calls system function so that it makes use of obfuscation techniques to hide features for escaping detection.

2.2 Webshell Detection

A webshell is a script that can be run on a web server, and it is also a plain text file essentially so that we can analyze its static and dynamic features. Up till the present moment, several detection methods have been proposed based on static features detections, dynamic features and web logs.

Tu et al. [[6](#bib1)] proposed a method based on the optimal threshold values of malicious functions and malicious signature samples to identify webshell, but their system may recognize a legal file which contains a few malicious functions for protection as a suspicious file. Wrench et al. [[7](#bib1)] determined the level of similarity within a collection of PHP malware samples by using four different measures of similarity to create representative similarity matrices. They decoded the samples, extracted the contents of user-defined function bodies, names of any user-defined functions and file fuzzy hashes for similarity analysis.

In recent years, some machine learning models are applied for webshell detection. Xin et al. [[8](#bib1)] proposed a webshell detection method based on matrix decomposition. However, the method is not confirmed whether is reliable to classify the page features. Ye et al. [[4](#bib1)] analyzed the HTML features of pages, proposed a method based on support vector machine (SVM). They compared two kernel functions in SVM, linear kernel and (Gaussian) radial basis function (RBF) kernel. The experiment showed that the former has a higher recall.

Shi et al. [[9](#bib1)] have proposed a new method of webshell detection based on web logs, which analyzes the file access path and the parameters, frequency of access to the file and page correlation to compare differences between webshells and normal web documents. Besides, they found the webshell is usually a solitary file. However, the method which only uses these features may have high false positive rates. Oleksii et al. [10] have used a combination of static and dynamic analysis techniques to uncover the shell’s visible and invisible features such as interface features, they also used honeypots to study how the attackers exploit the webshell.

2.3 PHP Opcodes

The opcode is a type of machine language instruction. It provides the computer with instructions indicating what to do with the data provided. So far, several related works showed that opcodes frequency can discriminate a malware software from a trusted one. In PHP, opcodes(or bytecodes) is a few difference from the opcodes used in malware identification, which are referred to a series of operation codes generated by Zend Engine 2 when parsing PHP files, representing the function of the code. They are similar in nature, as both tell the machine what to do. So we investigate whether PHP opcodes could be employed to detect PHP webshell.

PHP has the Vulcan Logic Disassembler (VLD) [11] extension. The VLD hooks into the Zend Engine and can help us to dump all the opcodes of script conveniently. An example listed below is the output of a typical webshell parsed by Zend Engine:

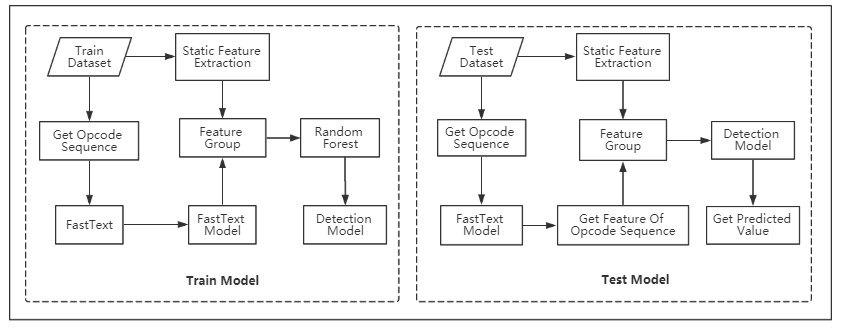
<?php eval($\_POST['a']); ?>

If running the above webshell through VLD, then we can see the following output in [Table 1](#tb1).

**Table 1:** Dump of all the opcodes from webshell

|  |  |
| --- | --- |
| # | OPCODE |
| 1 | EXT\_STMT |
| 2 | FETCH\_R |
| 3 | FETCH\_DIM\_R |
| 4 | EXT\_FCALL\_BEGIN |
| 5 | INCLUDE\_OR\_EVAL |
| 6 | EXT\_FCALL\_END |
| 7 | RETURN |

So we can get the PHP opcode sequence of a file: [‘EXT\_SMTM’,’F-ETCH\_R’, ‘FETCH\_DIM\_R’, ‘EXT\_FCALL\_BEGIN’, ‘INCLUDE\_OR\_ EVAL’, ‘EXT\_FCALL\_END’, ‘RETURN’].

 **Figure 1: The structure of FRF-WD**

3 PROPOSED APPROACH

The model we proposed which called FRF-WD, using PHP opcode sequences feature and static features to identify webshell. [Figure 1](#tb1) shows the structure of detection model and to assist understanding.

First, we extracted the features of the file, which are composed by two parts: static features and opcode sequences. On the one hand, we analyzed the static features of the file included longest string, entropy, index of coincidence and dangerous functions etc. On the other hand, we used PHP VLD extension to get opcode sequences of testing files, then using fastText to train the text classifier model. The predicted values of text classifier as the feature value of the PHP opcode sequences.

Afterwards, we use above features as inputs for random forest to train a webshell detection model.

3.1  Static Features

We identified a set of features that can be used for statistical analysis of PHP files:

A. Longest String

In order to escape signature-based matching, most of webshells will use obfuscation techniques. The idiomatic code obfuscation techniques used in webshell are encoding methods, such as base64 encoding. Encoder will produce a large string without space characters. However, typical text and script files are written in relatively short length of words, so the length of the longest uninterrupted string within a file is an effective feature. In addition, Tu et al. [[6](#bib1)] pointed out only and return back the longest string that starts and ends by header tags, which “<?php …?>” in PHP can reduce false positive rate because this kind of source codes contain many images, videos, rich-text-format, js and css also have long strings.

B. Information Entropy

It is defined as the average amount of information produced by a stochastic source of data. This test measure file’s uncertainty associated with the ASCII charset. Encryption can often increase information entropy into a text string so that measuring information entropy is useful in recognizing encrypted shellcode.

C. Index of coincidence(IC)

It provides a measure of how likely it is to draw two matching letters by randomly selecting two letters from a given text. It is useful both in the analysis of natural-language plaintext and in the analysis of ciphertext (cryptanalysis). If the value of IC is low, it indicates potential encryption or obfuscation in the file.

D. Signature

Although there are several techniques to thwart signature based detection or keyword searches, it is also useful to detect partial webshells. If one file calls dangerous PHP functions such as eval(), assert(), exec(), shell\_exec(), passthru(), system(), show\_source(), proc\_open() and pcntl\_exec(), then it is probably a suspicious file.

E. Blacklist keywords

When webshell developers write the code, the comment lines may contain suspicious keywords on the source codes, such as “webshell by \*”, “hack by \*”, “bypass AV”, “password is \*”, etc., which are rarely used in the normal file.

3.2 PHP Opcode Sequences Feature

By using the VLD extension of PHP, we can get opcodes sequence of a PHP file easily. Then we investigate whether could use text classifier techniques with opcodes to recognize the webshell. Each opcode local sequence exhibits correlations that are nearby opcodes in an opcode sequence are likely to be related to each other. These sequential patterns are important because they can be exploited to improve the performance of the predictors.

We chose the fastText to train a text classifier. The fastText model was proposed by Joulin et al. [[12](#bib1)]. It combines representing sentences with bag of words and bag of n-grams and uses the key features of rank constraint and fast loss approximation to improve the performance of basic linear classifiers.

The FastText model consists of a matrix A which is a look-up table over the words and a matrix B for classifier. The representations of the word are averaged into a text representation, which is in turn fed to the linear classifier. Using the softmax function f to obtain a probability distribution over pre-defined classes. For a set of N documents, the model leads to minimize:

|  |  |
| --- | --- |
|  | (1) |

Where is the normalized BoW of the n-th document, the label.

FastText is much faster but can achieve performance on par with the proposed methods based on deep learning. In particular, the length of the opcode sequence of different files are various, some may have only a few opcodes, some can have the opcodes over than thousands, while fastText is more suitable to process such dataset than some deep learning methods such as TextCNN [[13](#bib1)]

In this work, we first used VLD extension of PHP to get opcode sequence of a file, then we used the labeled samples on fastText to train a text classifier, then we can load the pre-trained classifier to predict the label of text which is generated from the PHP opcode sequences. The predicted value is used as the feature value of the file opcodes sequence

3.3 Classification

After the features are extracted, we use a random forest (RF) [[14](#bib1)] method to achieve a binary classifier. The random forest is an ensemble approach that can also be thought of as a form of nearest neighbor predictor. The main principle behind ensemble methods is that a group of “weak learners” can come together to form a “strong learner”. The result from an ensemble model is usually better than the result from one of the individual models. RF can run fast and efficiently with a high predictive accuracy. It also can generate an internal unbiased estimate of the generalization error as the forest building progress.

4 EXPERIMENTAL AND EVALUATION

4.1 Dataset

To train our model, a dataset of webshells and benign files was collected. In total, we collected 8521 PHP files which include 1587 available PHP webshells. The webshells came from several webshell collection projects on Github [[16-18](#bib1)]. Benign files are from several famous PHP frameworks such as Yii2, Wordpress, CodeIgniter, etc. The dataset was divided into two parts with a partition ratio of 3:7. Thirty percent of the dataset is used for building the text classifier of PHP opcode sequences, the rest for training and testing model of RF.

4.2 Experiment Design

Firstly, we also extracted static features of file which are discussed in Section 3.1.

Then, we use PHP VLD extension to get file’s opcode sequences, which would be used for training the text classifier by using fastText presented in Section 3.2. With this text classifier, we get the opcode sequences feature value. In the fastText model, an important argument is word-Ngram because the sequence of opcodes is more representative than single opcode. In the experiments, we would evaluate the influence of N-gram size from n=1 to n=6 in order to find the best performance.

Finally, we trained the random forest classifier with 100 decision trees by using these extracted features. We used a 10-fold cross-validation method to evaluate the performance of our model.

The performance evaluation is done using statistical measures of binary classification, as details are described as follow:

Detection accuracy is defined in (2) and is the correct classification of True Positive(TP) and True Negative(TN):

|  |  |
| --- | --- |
|  | (2) |

Where TP is the number of webshell files that are correctly identified, FN is the number of webshell files that are mistakenly classified as a benign file.

Recall and precision are defined in (3) and (4):

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |

F1-score is the harmonic mean of precision and recall and defined in (5):

|  |  |
| --- | --- |
|  | (5) |

4.3 Experiment Result

First, we tried to find the best N-gram size in detection model. Figure 2 presents the precision and recall of the model with different lengths of word N-grams. We can find the best n-gram of Opcodes was the 4-grams. Moreover, longer grams (n>4) would decrease the accuracy in the experiment, which possibly because fewer appearances in files, thus creating ineffectual representation vectors.

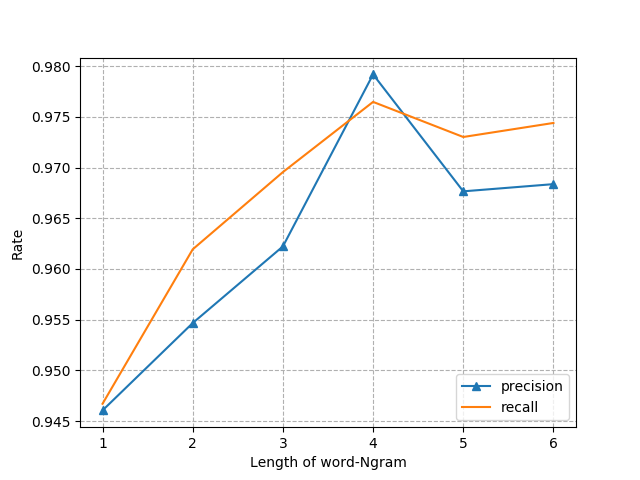


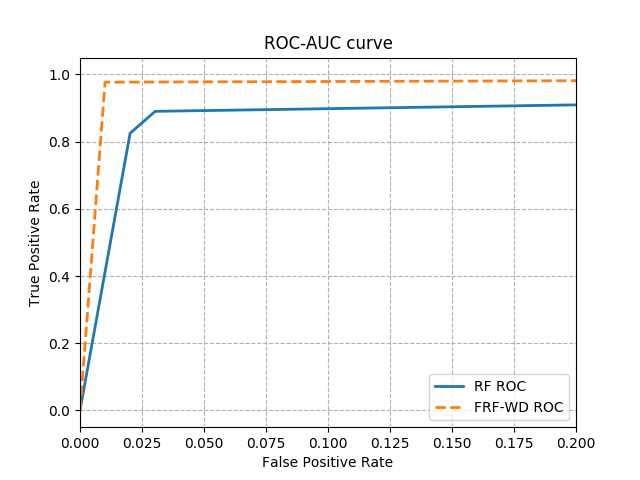
Figure 2: Evaluation by using different length of word-Ngram

In addition, we also compared the performance RF model (means without PHP opcode sequences feature) and fastText & RF model. The results are shown in Table 2.

**Table 2:** Result summary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | ACC. | Recall | Pre. | F1-score |
| RF | 0.9619 | 0.8823 | 0.8897 | 0.8860 |
| FRF-WD | 0.9923 | 0.9765 | 0.9792 | 0.9778 |

The RF model without the feature of the PHP opcodes sequences shows a recall of 88.23% and precision of 88.97%. When applying the opcode sequences feature value from fastText model to train the random forest classifier, the accuracy increased to over 99%, with a recall of 97.65% and precision of 97.92%. The ROC curves of the models are shown in Figure 3.

Figure 3: ROC curve of different models

5 CONCLUSION

In this paper, we explored the use of PHP opcode sequence on PHP webshell detection. From the experimental results, we can conclude that detecting PHP opcode sequence of files is an effective approach to predict webshells. Moreover, we combined statistical features of the file to train a random forest classifier, the accuracy increased to 99.23% with high precision and high recall, it shows that the model works well on webshell detection. Up till now, we only explored the PHP opcode sequence for PHP webshell detection. In the future, we will try to study the effectiveness of opcodes for detecting webshell developed by other languages.

REFERENCES

|  |  |
| --- | --- |
| [1] | Acunetix Web Application Vulnerability Report 2016[EB/OL]: https://d3eaqdewfg2crq.cloudfront.net/resources/acunetix-web-application-vulnerability-report-2016.pdf |
| [2] | Scott, B., & H. Ben. Web Shell Detection Using NeoPI [EB/OL]:[http://resources.infosecinstitute.com/web-shell-detection/](http://resources.infosecinstitute.com/web-shell-detection/(NEOPI) |
| [3] | Jiankang H U, Zhen X U, Duohe M A, et al. Research of Webshell Detection Based on Decision Tree[J]. Journal of Network New Media, 2012. |
| [4] | F Ye, J Gong, W Yang. Black box detection of webshell based on support vector machine[J]. Journal of Nanjing University of Aeronautics & Astronautics, 2015. |
| [5] | Wrench, P. M., & Irwin, B. V. (2015, August). Towards a PHP webshell taxonomy using deobfuscation-assisted similarity analysis. In Information Security for South Africa (ISSA), 2015(pp. 1-8). IEEE. |
| [6] | Tu T D, Guang C, Xiaojun G, et al. Webshell detection techniques in web applications[C]//Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on. IEEE, 2014: 1-7. |
| [7] | Wrench P M, Irwin B V W. Towards a PHP webshell taxonomy using deobfuscation-assisted similarity analysis[C]//Information Security for South Africa (ISSA), 2015. IEEE, 2015: 1-8. |
| [8] | Sun X, Lu X, Dai H. A Matrix Decomposition based Webshell Detection Method[C]//Proceedings of the 2017 International Conference on Cryptography, Security and Privacy. ACM, 2017: 66-70. |
| [9] | L Shi, Y Fang. Webshell Detection Method Research Based on Web Log[J].  Journal of Information Security Research, 2016. |
| [10] | Starov O, Dahse J, Ahmad S S, et al. No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells[C]// International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2016:1021-1032. |
| [11] | Package Information[EB/OL]: https://pecl.php.net/package/vld |
| [12] | Joulin A, Grave E, Bojanowski P, et al. Bag of tricks for efficient text classification[J]. arXiv preprint arXiv:1607.01759, 2016. |
| [13] | Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014. |
| [14] | Liaw A, Wiener M. Classification and regression by randomForest[J]. R news, 2002, 2(3): 18-22. |
| [15] | O’kane P, Sezer S, McLaughlin K. Detecting obfuscated malware using reduced opcode set and optimised runtime trace[J]. Security Informatics, 2016, 5(1): 2. |
| [16] | Webshell-sample[EB/OL]:https://github.com/ysrc/webshellsample/tree/master/-php |
| [17] | A webshell open source project[EB/OL]:https://github.com/tennc/webshell/tre-e/master/php |
| [18] | PHP-backdoors[EB/OL]: https://github.com/bartblaze/PHP-backdoors |