SmartDetect: A Smart Detection Scheme for Malicious Web Shell Codes via Ensemble Learning

Zijian Zhang, Meng Li, Liehuang Zhu(✉), and Xinyi Li

School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China
{zhangzijian,menglibit,liehuangz,2120171114}@bit.edu.cn

Abstract. The rapid global spread of the web technology has led to an increase in unauthorized intrusions into computers and networks. Malicious web shell codes used by hackers can often cause extremely harmful consequences. However, the existing detection methods cannot precisely distinguish between the bad codes and the good codes. To solve this problem, we first detected the malicious web shell codes by applying the traditional data mining algorithms: Support Vector Machine, K-Nearest Neighbor, Naive Bayes, Decision Tree, and Convolutional Neural Network. Then, we designed an ensemble learning classifier to further improve the accuracy. Our experimental analysis proved that the accuracy of SmartDetect—our proposed smart detection scheme for malicious web shell codes—was higher than the accuracy of Shell Detector and NeoPI on the dataset collected from Github. Also, the equal-error rate of the detection result of SmartDetect was lower than those of Shell Detector and NeoPI.

Keywords: Smart detection · Malicious web shell code · Data mining

1 Introduction

High-speed Internet access provided by web technologies has enhanced web experience for users. However, user data remains susceptible to security attacks because these technologies do not provide security to users. Therefore, web server attacks have become one of the most dangerous Internet problems. Canali and Balzarotti found that almost half of the hackers use web shells for their attacks [1].

To create backdoors in servers, hackers often upload web shells that enable remote control and administration of these servers. Attackers explore and discover web-specific vulnerabilities and configuration loopholes, such as Cross-Site Scripting and SQL injection, to expose the administrator's interfaces [2]. After hackers gain control of the server, they can arbitrarily upload and download files, view databases, execute commands, elevate their privileges, and subsequently control the entire network of the enterprise. This can have devastating

M. Qiu (Ed.): SmartCom 2018, LNCS 11344, pp. 196–205, 2018.
https://doi.org/10.1007/978-3-030-05755-8_20

consequences. Therefore, it is very important to develop a precise method for detecting web shells.

The commonly used methods for the detection of web shell codes can be categorized into three main groups: static detection, dynamic detection, and statistical detection. The static detection method uses the regularization feature to match the malicious commands. Dynamic detection determines the threat by using system commands and abnormal network traffic [3]. Statistical detection uses entropy, longest word, index of coincidence, signature, and the compression rate to detect the web shell codes.

Moreover, several automatic detection tools have been developed that have better efficiency than the manual detection of web shell codes. For instance, Shell Detector [4] is a static detection tool. It can find both the original and encoded web shells. NeoPI is a statistical detection tool to detect obfuscated and hidden web shell codes. Unfortunately, both tools have low precision and high equal-error rates.

This research proposes a model that overcomes these concerns to a large extent. Our main contributions are as follows:

- We compared SmartDetect with five existing data mining algorithms: Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naive Bayes (NB), Decision Tree (DT), and Convolutional Neural Network (CNN), for web shell detection. To improve the detection accuracy, we propose SmartDetect, a smart scheme for detecting malicious web shell codes.
- We applied ensemble learning to improve the accuracy of the proposed scheme. To the best of our knowledge, this is the first study that constructs an ensemble classifier for detecting malicious web shell codes.
- Our experimental analysis showed that the precision of the ensemble classifier was higher than that of the existing classifiers such as Shell Detector and NeoPI. In addition, the equal-error rate of the ensemble classifier was lower than that of the Shell Detector and NeoPI.

2 Related Work

In this section, we have investigated several studies from the last decade on malicious web shell detection. These studies can be divided into four categories.

Starov et al. [2] performed a comprehensive study on web shells. They used several static and dynamic detection methods, and they checked the visible and invisible features generated by popular malicious shells. The result demonstrated that 25% shells were missed in the sample set even for the best detection system at that time.

Tu et al. [5] proposed a detection method to identify malicious codes by using the optimal threshold values. They calculated the score for malicious signatures and malicious functions of the source code. An optimal threshold value was determined to select the suspicious files whose signature total score exceeded the threshold value. The system received a true positive rate of 79.9% and a false positive rate of 2.0%.

Yi Nan et al. [6] proposed a detection scheme for semantics-based web shells by using an abstract syntax subtree extraction algorithm. This algorithm could locate the malicious behavior of web shell files by using the risk assessment table of nodes. However, it could not detect certain specific shells.

Wrench et al. [7] proposed a shell classification approach based on the similarities of the files to known malwares. They proposed four ways of classifying shells based on their similarities: their aim was to produce representative similarity matrices. Then, the matrices were visualized and interpreted graphically. However, the existing web shells that detected malicious shells were not flexible enough. Our proposed method, SmartDetect, can be used to manage the deduplication and deobfuscation of web shell data sets.

3 Preliminaries

3.1 Web Shell

Based on their complexity and available functions, web shells can be basically divided into three categories: complex web shells, simple web shells, and web shells that contain one command line [8]. The complex web shells contain all the features of Trojans. They usually need to call the system's key commands such as exec and system. They are concealed using code encryption. Simple web shells include only the function of uploading files. Their file sizes are small. The web shells that contain one command line can be used flexibly because they can be used as separate files or as parts of a normal file. Therefore, it is difficult to detect these web shells by using ordinary methods.

For identification, web shells can be basically divided into two categories. One category is non-encoded web shells, and the other is encoded web shells. Non-encoded web shells store their source code without being encoded. Thus, we can detect the commands directly. Encoded web shells contain commands that are obfuscated. For example, hackers always use base64_encode to encode the commands [9].

3.2 Ensemble Classifier

Dietterich [10] theoretically proved that ensemble classifiers have better classification accuracy than single-component classifiers. Miranda Dos Santos [11] proved this fact empirically. The error rate of ensemble classifiers could be reduced by implementing multiple base classifiers if the accuracy of each base classifier was over 50%. For the detection of web shells, we used ensemble classifiers to improve the detection accuracy because web shells are of different types, and we can detect these web shells using different basic detectors. Our study uses bagging to realize the ensemble classifier. Bagging, namely Breiman's bootstrap aggregating method, is a straightforward way to obtain high efficiency [12]. A number of sampling sets are produced by using bootstrap sampling. We used each sampling set to train a distinct classifier; then, we combined these classifiers by using the simple voting method.

4 SmartDetect

We propose SmartDetect: a smart code detection scheme for malicious web shells. The concrete scheme consists of three steps. The first step was to preprocess the collected data. Second, we used the bootstrap sampling measure to generate five sample sets and train five classifiers separately. Finally, we used bagging to design an ensemble learning algorithm by which the malicious web shell codes could be precisely detected.

4.1 Data Preprocessing

We obtained web shell samples from some publicly available data sets on the web [13–16]. Given the complexity and diversity of web shells, there were a number of data sets having entirely different structures; all of these could not be used directly for our scheme. However, we preprocessed the collected data by identifying a number of similar web shell codes. In this paper, we mainly deal with the malicious web shell codes with the PHP script language. We performed the following steps:

- First, we filtered the web shell codes by checking the PHP tags. Then, the web shell codes with non-PHP script languages were deleted. Subsequently, we obtained 1404 PHP files [13–15] with web shell codes.
- Second, we removed the web shells with syntactic meaning, whitespaces, and comments. After this procedure, we had 826 shells.
- Code obfuscation was used to prevent reverse engineering [17]. Although the obfuscation technique could protect the proprietary code, it could also be used by web shell authors to hide the malicious code. We used the UnPHP deobfuscation service [18,19] for automatically deobfuscating the shells.

The normal PHP files were collected from an open source software developed by multiple PHP projects [20–25]. In fact, we eventually collected a total of 8045 PHP files.

4.2 Feature Extraction

We used the Opcode bi-gram model to extract the features. Here, Opcode [26] is part of the instructions that specify the operation to be performed. The content of the instructions were determined by the instruction specifications of the previous step. In addition, certain operands are usually required by the instruction. The possible instructions do not require explicit operands. These operands may be the values in the registers, the values in a stack, the values in a memory block, or the values in an input or an output port. We used PHP's VLD [27] extension to view the Opcode of the PHP file. The N-Gram model is based on the assumption that the occurrence of the n^{th} word is only related to the preceding $N-1$ words. The probability of the entire sentence is the product of the probabilities of the occurrences of the words that make up the sentence.

The flowchart for the feature extraction model is as follows:

Algorithm 1. Feature Extraction Algorithm

Input:
PHP Files F.
Output:
Features $\mathcal{F}$.
1: *Put F into VLD extension module.*
2: *Obtain Opcode O.*
3: *Put O into Bi-Gram model.*
4: **return** $\mathcal{F}$.

4.3 Model Training

We first used SVM, KNN, NB, DT, and CNN to train fives classifiers by using the above-mentioned training sets to obtain a well-performing classifier. The architecture for web shells classification is shown in Fig. 1.

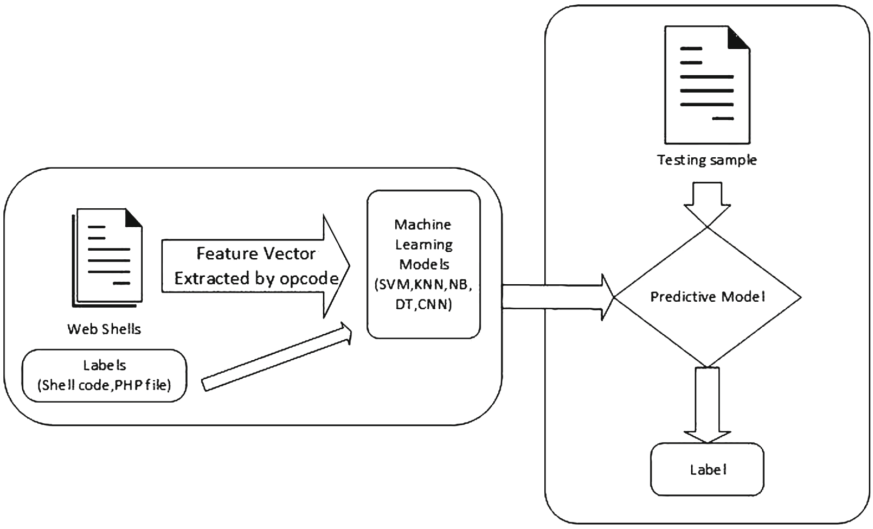

Fig. 1. Architecture for web shell codes' classification

We illustrate the technical details by using the SVM-based malicious web shell codes as an example. The model uses the following formula.

$$min\frac{1}{2}\left\|\omega\right\|$$

$$s.t.\ y_i(\omega^T x_i + b) \geq 1, i = 1, 2, ..., m$$

Then, the RBF kernel is defined as follows:

$$\kappa(x_i, x_j) = exp(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}).$$

4.4 Ensemble Learning

We continue to apply bagging to design the ensemble learning algorithm to further improve accuracy. Bagging can be applied to a parallel model where there is no strong dependency between the individual learners and can be generated simultaneously.

The algorithm for building a bagging model is shown in Fig. 2.

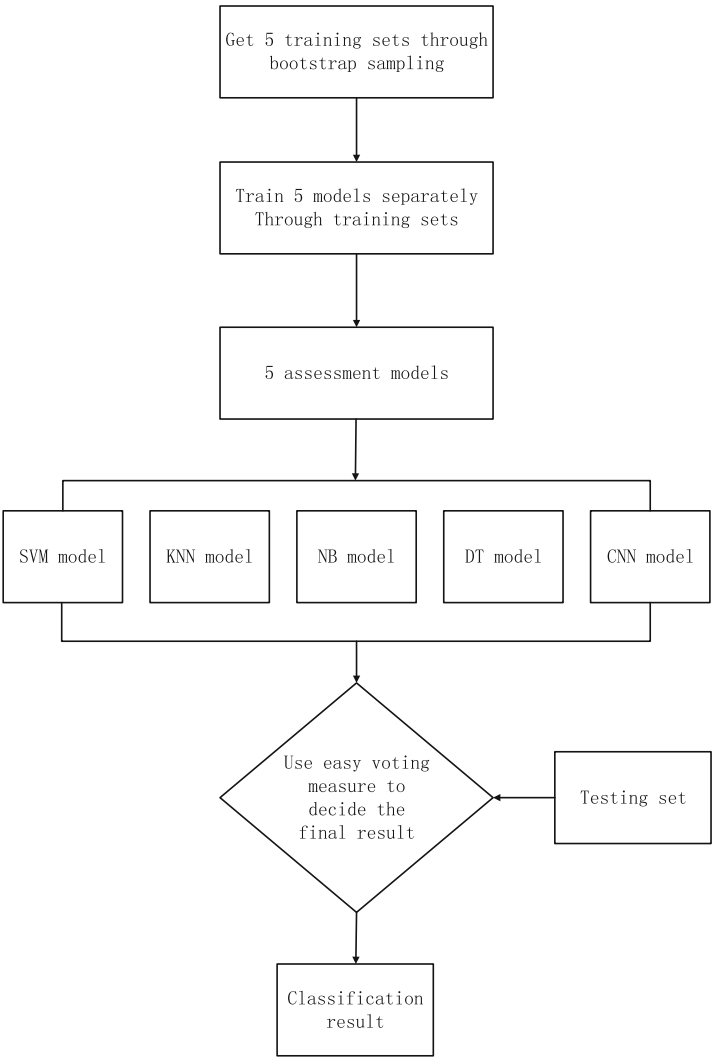

Fig. 2. Experimental flowchart for bagging

Algorithm 2. Ensemble-Learning-Based Detection Algorithm

Input:

1. Basic classifier: C,
2. The iteration of training: t. The total iteration T is 5,
3. Training set: S.

Output:

Ensemble of classifiers: C_t.

1: $t = 1$
2: *while* $t < T$
3: $S_t \leftarrow$ *the subset of training set generated through bootstrap sampling*
4: *Using* S_t *to create basic classifier* C_t
5: $t{+}{+}$
6: **return** *C(Ensemble of classifiers* C_t *by simple voting measure)*

5 Performance Analysis

We compared the accuracy of our proposed scheme with the existing detection tools. The experimental flowchart is shown in Fig. 2.

We first compared the accuracy of the proposed scheme. For this, we considered five data mining algorithms: SVM, KNN, NB, DT, and CNN. The results are shown in Fig. 4.

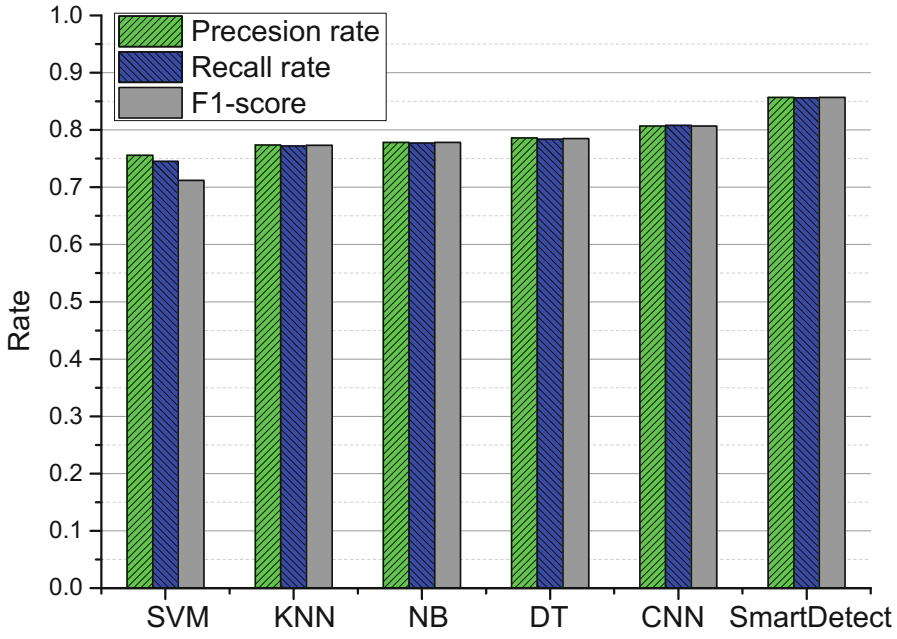

Fig. 3. Architecture for web shells' classification

From the table, we can see that the best basic classifier is from the CNN-based detection scheme. Its precision reaches 80.7%, which is higher than that of other algorithms. Our ensemble classifier achieved the highest precision of 85.7% (Fig. 3).

Then, we compared the error rate. The ROC curve is shown in Fig. 4.

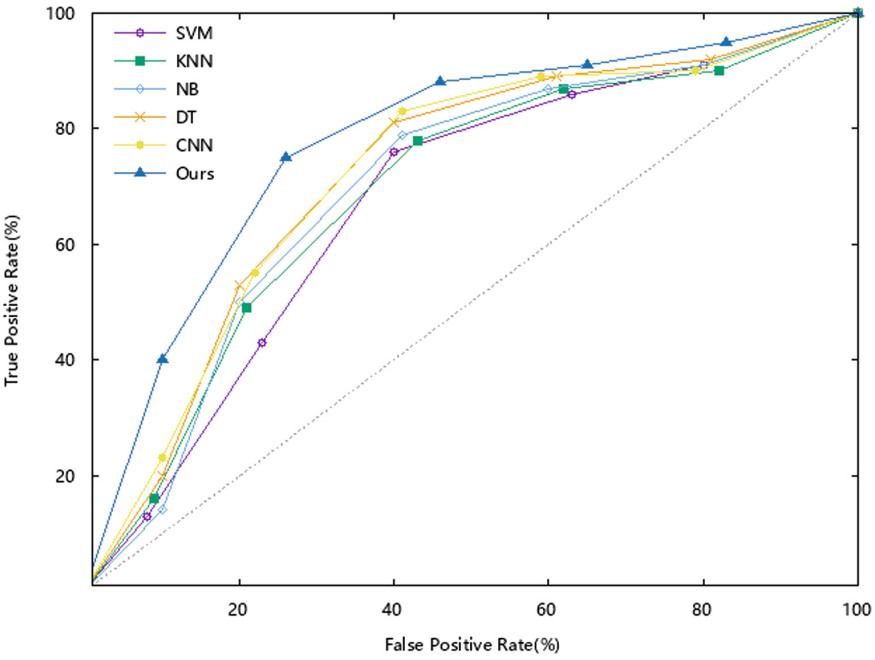

Fig. 4. Architecture for web shells classification

From the figure, we can see that the error rate of the proposed detection scheme is far superior to that of existing detection schemes.

6 Conclusion

The damage caused by web shells can be extremely serious. Unfortunately, the existing detection methods are not very effective. In this study, we developed SmartDetect, an ensemble learning model to detect web shells. Five learning algorithms were applied. Our experimental results proved that the precision of our ensemble learning algorithm increased by at least 5% as compared with the basic learning algorithm. It is difficult to detect web shells using a single classifier because there are various kinds of web shells; therefore, we used the ensemble classifier.

Our experimental analysis also showed that the precision of the ensemble classifier was higher than that of the existing classifiers such as Shell Detector

and NeoPI, whereas the equal-error rate of the ensemble classifier was lower than that of NeoPI.

With the advent of the intelligent age [28,29], the use of machine learning methods to detect malicious code is becoming increasingly significant. Moreover, the methods used in our research can be applied to other malicious-code detection fields. In our future studies, we could investigate web shells that are hidden in test images.

Acknowledgment. This work is partially supported by China National Key Research and Development Program No. 2016YFB0800301 and National Natural Science Foundation of China No. 61872041.

References

1. Canali, D., Balzarotti, D.: Behind the scenes of online attacks: an analysis of exploitation behaviors on the web. In: NDSS 2013, 20th Annual Network and Distributed System Security Symposium, San Diego, CA, United States, 24–27 February 2013 (2011)
2. 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, International World Wide Web Conferences Steering Committee, pp. 1021–1032 (2016)
3. Xue, L., Ma, X., Luo, X., Chan, E.W.W., Miu, T.T.N., Gu, G.: LinkScope: toward detecting target link flooding attacks. IEEE Trans. Inf. Forensics Secur. **13**(10), 2423–2438 (2018)
4. http://www.shelldetector.com
5. Tu, T.D., Guang, C., Xiaojun, G., Wubin, P.: Webshell detection techniques in web applications. In: Proceedings of the International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1-7 (2014)
6. Yi Nan, H.C.L.L., Yong, F.: Semantics-based webshell detection method research. Res. Inf. Secur. **3**(2), 145–150 (2017)
7. Wrench, P.M., Irwin, B.V.: Towards a PHP webshell taxonomy using deobfuscation-assisted similarity analysis. In: Proceedings of the Information Security for South Africa (ISSA), pp. 1-8 (2015)
8. Kolbitsch, C., Livshits, B., Zorn, B., Seifert, C.: Rozzle: de-cloaking internet malware. In: Proceedings of the IEEE Symposium on Security and Privacy 2012, pp. 443-457 (2012)
9. Exploitable PHP functions. https://stackoverflow.com/questions/3115559/exploitable-php-functions
10. Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45014-9_1
11. Miranda Dos Santos, E.: Static and dynamic overproduction and selection of classifier ensembles with genetic algorithms. Ph.D. thesis, École de technologie supérieure (2008)
12. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
13. Webshell open source project. https://github.com/tennc/webshell
14. Common PHP webshells. https://github.com/JohnTroony/php-webshells

15. Nikicat's webshells collection project. https://github.com/nikicat/web-malware-collection
16. Gai, K., Qiu, M.: Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers. IEEE Trans. Ind. Inform. **4**(8), 3590–3598 (2018)
17. Wrench, P.M., Irwin, B.V.: Towards a sandbox for the deobfuscation and dissection of PHP malware. In: Proceedings of the Information Security for South Africa (ISSA), pp. 1–8 (2014)
18. UnPHP - the online PHP decoder. https://stackoverflow.com/questions/3115559/exploitable-php-functions
19. Gai, K., Choo, K.-K.R., Qiu, M., Zhu, L.: Privacy-preserving content-oriented wireless communication in internet-of-things. IEEE Internet Things J. **5**(4), 3059–3067 (2018)
20. Wordpress project. https://github.com/WordPress/WordPress
21. A PHP blogging platform. https://github.com/typecho/typecho
22. A web interface for MySQL and MariaDB. https://github.com/phpmyadmin/phpmyadmin
23. A PHP framework for web artisans. https://github.com/laravel/laravel
24. The symfony PHP framework. https://github.com/symfony/symfony
25. Yii 2: the fast, secure and professional PHP framework. https://github.com/yiisoft/yii2
26. Opcode. http://www.php-internals.com/book/?p=chapt02/02-03-02-opcode
27. Visual leak detector. https://github.com/KindDragon/vld
28. Gai, K., Qiu, M., Xiong, Z., Liu, M.: Privacy-preserving multi-channel communication in edge-of-things. Futur. Gener. Comput. Syst. **85**, 190–200 (2018)
29. Zhu, L., Li, M., Zhang, Z., Zhan, Q.: ASAP: an anonymous smart-parking and payment scheme in vehicular networks. IEEE Trans. Dependable Secur. Comput. (TDSC) **PP**(99) (2018)