ICIC Express Letters
Volume **13**, Number **6**, June **2019**

ICIC International ©2019 ISSN 1881-803X
pp. **445–452**

CLASSIFICATION OF WEB BACKDOOR MALWARE BASED ON FUNCTION CALL EXECUTION OF STATIC ANALYSIS

Aditya Kurniawan[1,2], Bahtiar Saleh Abbas[2,4], Agung Trisetyarso[2] and Sani Muhammad Isa[2,3]

[1]Computer Science Department, School of Computer Science
[2]Doctor of Computer Science, Binus Graduate Program
[3]Magister of Information Technology, Binus Graduate Program
[4]Industrial Engineering Department, Faculty of Engineering
Bina Nusantara University
Jl. K. H. Syahdan No. 9, Kemanggisan, Palmerah, Jakarta 11480, Indonesia
{ adkurniawan; bahtiars; atrisetyarso; sisa }@binus.edu

Received October 2018; accepted January 2019

Abstract. *We introduce the classification of the web shell backdoor, or web malware, by using a sensitive function call analysis. Our result illustrates that the support vector machine has a 0.92 (92%) accuracy value and a 0.927 (92.7%) precision value. The support vector machine has the highest result, when compared with other classification algorithms. Our novel method uses a static analysis to extract the function call based sensitive function list correctly and not just as a keyword. The source codes that read as a bag of words or read with neuro linguistic programming are prone to a false positive detection. This is because the source code language has different structures compared with the regular language. The source code uses a static analysis to extract its abstract syntax tree structures.*
Keywords: Web shell backdoor, Classification, Function call, Static analysis

1. **Introduction.** In January of 2018, there were more than 1,545 million websites [12] active, on a daily basis, for business and personal purposes. The tremendous number of websites makes them an attack target by malicious hackers, known as black-hat hackers. A web shell backdoor can be uploaded to the target website to control and exploit the web server. More specifically, the web shell backdoor is a malicious code that is crafted and used by black-hat hackers to escalate their privilege access and maintain secret access on a compromised web application.

The web shell is a common type of web backdoor that is usually written in a scripting language (e.g., PHP, ASP, and JSP) [10]. The web shell has the ability to crawl a file and folder directory after this backdoor is injected into the server. The web shell can also execute terminal commands, connect to a database (e.g., MySQL), and execute an sql query command.

The web shell backdoor has a remote shell server administration capability. The injection of a web shell backdoor is a post-exploitation step that occurs after the hacker successfully exploited an injection vulnerability (e.g., remote file inclusion, and injection). File upload features also have a file injection vulnerability without a constraint and validation. Therefore, the shell backdoor has to be inside the web server. An injected shell backdoor is evidence that a file injection vulnerability has been exploited.

There are three types of shell backdoors; they differ in how the backdoors are injected. The first type is a web shell backdoor that has a script or source code form. These types are common. There are thousands of shell backdoors available publicly on the Internet.

DOI: 10.24507/icicel.13.06.445

These source code backdoors are usually injected into a web server via a file upload or URL file injection. The second type is a web shell backdoor where the source code is inserted into the other media like images. These types are usually injected via image upload features. The third type is a web shell backdoor where the source code has been encrypted or encoded [17, 18]. These types are usually injected via a file upload or file inclusion. The source code is encrypted to avoid any server administrator detection.

A black-hat hacker can easily craft a shell backdoor. The shell backdoor is created to control a web server with a simple configuration setting in its server administration. The shell administration functionality that the shell backdoor is capable of includes, but is not limited to: shell command function, file management, database enumeration, code execution, traversing directories, viewing file content, download file, delete file, execution of SQL query, and bypass mod security [3, 10, 11]. Code Listing 1 provides an example of a simple backdoor PHP code.

Code Listing 1. Example of simple backdoor PHP code

```
1  if(isset($_REQUEST['cmd'])){
2  echo "<br><br>";
3  $cmd = ($_REQUEST['cmd']);
4  system($cmd);
5  echo "<br><br>";
6  }
```

The capability of the simple shell backdoor in Code Listing 1 allows the black-hat hacker to insert a shell command execution into a fabricated URL (http://exploitedsystem.com/simplebackdoor.php?cmd=cat+/etc/shadow). A shell backdoor can have more sophisticated features for interactions (e.g., console system, edit files, and sql manager).

This paper is arranged as follows. Section 1 explains problem background of backdoor malware detection. Section 2 explains current research in backdoor malware detection. Section 3 explains data collection and analysis and describes the proposed methodology. Section 4 describes result and discussion. Section 5 represents the conclusion.

2. Related Work.

2.1. False positive of web shell detection based on keywords.

A web shell backdoor is difficult to detect with an antivirus or an intrusion detection system (IDS). The antivirus and IDS find it difficult to distinguish between the normal web application or the web shell source code. There are many web shell detections currently available (e.g., the famous Linux Malware Detect (LMD) [7], NeoPI [2], and PHP Web Shell Backdoor [6]).

The web shell detection tools are signatures or patterns based on keywords. Hagen and Behrens tested the LMD against 90 shell backdoors. The LMD was capable of detecting 37 shell backdoors out of 90 [2]. These tools failed to detect some obfuscated or encoded web shell backdoors. The PHP shell detector used 141 web shells to construct their signature pattern database [6]. Tian et al. developed a web shell detection approach based on word to vector (word2vec) representation and a convolutional neural network (CNN) [15]. All of the previous work's tools and approaches read the web shell source code as a bag of words, not compiled as source code structures.

It is important to treat the web shell source code as a structure, not as a bag of words for content purposes. Figure 1 provides an example. The funccall.php file in Figure 1(a) shows the execution of the shell_exec function call. Figure 1(b) illustrates the definition of a shell_exec function by a developer. The content.docx file is a regular word document that has a shell_exec description illustrated in Figure 1(c). If the three files are in one folder and are read as a signature keyword pattern, or with a word to vector, and are

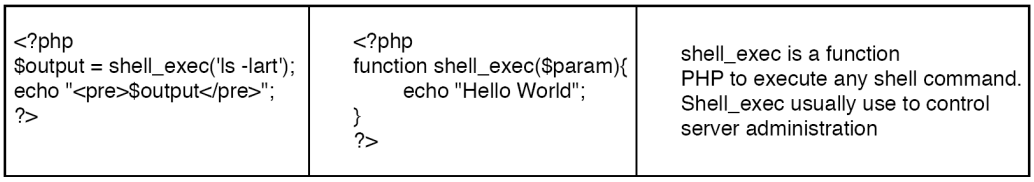

FIGURE 1. Comparison of source code structure and content

applied to machine learning, then Figures 1(b) and 1(c) will detect a false positive. This is because there are no shell_exec function call executions in both files.

2.2. **Static analysis to identify web shell source code structure.** The technique that reads the source code and analyzes the structure is known as a static analysis. A static analysis is a process that capitulates the source code, builds an intermediate model that represents the program through the model extraction process, analyzes the model based on the requirements, and uses the results for other analyses or computations [4].

The model extraction is a lexical analysis process used to transform the source code into a specific token. The intermediate representation is a set of data structures that represents the source code. The source code tokenization uses a set of context-free grammar to match the token stream and construct an intermediate representation (e.g., abstract syntax tree [8]). An abstract syntax tree is built by associating a parse tree with the grammar's production rules. Intermediate representation is created to analyze a source code pattern structure [5].

The programming language parser transforms the PHP source code into an abstract syntax tree by using the PHP grammar, which is already defined. The PHP grammar is mapped into nodes that represent every attribute for each line of the source code. Our algorithm will traverse all nodes with a deep-first search algorithm that searches any function call object node in the abstract syntax tree. The static analysis process will detect every function call in the source code correctly.

3. Methodology.

3.1. **Data collection and analysis.** Our novel method combines the static analysis method, used to detect any sensitive function call and classify the web shell backdoor based on those sensitive function calls, with a machine learning algorithm. There are 50 sensitive function names gathered that are related to the web shell. Table 1 illustrates all of the function names that can be used as machine learning data features.

This paper collected 100 web shells from a primary data source, the server of a hosting company. The web shell backdoor sample was collected carefully and tested in a virtual server sandbox environment. One hundred web shells were selected from the 1200 PHP files that indicated that they were a web shell.

3.2. **Static analysis algorithm.** The static analysis algorithm of our method used a modified Popov's Library[1] that mapped the grammar language into object types. Our method efficiently traverses an abstract syntax tree with a 52% grammar reduction from 140 grammar [19]. Our method only filters the function call that will check the function name.

Algorithm 1 prepared the sensitive function call that has been mapped to code the unstructured object database. This paper uses mongodb as the database, because the algorithm needs to be query based on a node object that maps to a symbol code. The algorithm is combining result between our efficient traverses an abstract syntax tree and sensitive function call.

[1]https://wiki.php.net/rfc/abstract_syntax_tree.

Table 1. List of web shell sensitive function

Web Shell Sensitive Function		
set_time_limit	fopen	eval
mysql_query	curl_exec	htmlspecialchars
fwrite	system	mysqli_query
curl_close	ini_set	fputs
exec	sqlsrv_query	curl_setopt
base64_decode	rename	passthru
mssql_query	gzopen	gzinflate
unlink	shell_exec	pg_query
gzclose	str_rot13	touch
proc_open	oci_parse	gzencode
rmdir	chmod	proc_close
oci_execute	bzopen	copy
file_put_content	popen	sqlite_query
bzwrite	rename	pclose
fclose	pclose	odbc_exec
bzclose	file_get_contents	win_shell_execute
curl_init	bzcompress	

Algorithm 1 involves two primary steps. Firstly, the algorithm chooses the node of the abstract syntax tree and maps it with a sensitive function call. The algorithm will calculate the number of occurrences of the sensitive function call. Secondly, the name of the function will be compared with the function names that have been saved in the mongodb object set. If both names are equal, then the value occurrence of the function will be added.

Figure 2 presents an example of the algorithm process. The exec function call is detected in the line of the source code. The name of the exec function will be compared with the mongodb object set on the right side. The compared result will add the new occurrence of the exec function.

After all sensitive function call occurrences are counted in a PHP file, the result will be saved in the mongodb object set. The data will then be divided into two classes: positive and negative. The positive class represents all of the calculated sensitive function calls in the malware type. The negative class represents all of the calculated sensitive function calls in the non-malware type.

The mongodb object data structure is represented in Code Listing 2. This object data set will be used in the training and classification process.

3.3. **Four classification algorithms.** Four classification algorithms are used in this investigation: neural network [9], support vector machine [1], decision tree [14], and naive Bayes [13]. These four classification algorithms were selected because this paper uses pre-classified or supervised data. In this way, the supervised data can proceed with the four supervised learning algorithms. They also have different precision and recall result values. Therefore, those algorithms precision will be compared in Figure 4.

Our web shell classification consists of two classes, malware and non-malware. This paper uses 100 of PHP files that consist of 50 web shell as malware class and other 50 regular PHP files that contain some or less sensitive function call numbers as non-malware class. The numbers of all fifty sensitive function in Table 1 will be counted in each PHP file with static analysis process. The static analysis process can be seen at Algorithm 1.

Classification web shell consists of two steps: training and testing. Calculation of sensitive function call numbers has to be done before training step. Sample of web shell

backdoor will be tested after training process. The testing result will be evaluated by using confusion matrix [16]. Confusion matrix is used to calculate accuration, precision, recall, and F-1 score that can be seen in Table 2.

Algorithm 1: Shell backdoor extract sensitive function call node

input : node = $\{v_1, v_2, \ldots, v_n \mid$ node a block set of tree node $v\}$
output: Sensitive function node based on MongoDB object

```
Function extractFunctionNode(node)
    extractedNode = node.extract() //Extract args function to filter in MongoDB object
    resultNode = extractRecursiveArg(node.extract())
    filter = {resultNode['type'], resultNode['name']}
    searchResult = searchMongoDBObject(filter)
    // filter by function's arguments
    foreach result as searchResult do
        // if the db's data has "regexArgs" key, then compare by the regex only
        if key_exists('regexArgs', result) then
            //create regexArgs for extracted node from regex type
            regexArgsExtracted = getRegexArgument(extractedNode['args'])
            if regexArgsExtracted equals result['regexArgs'] then
                if key_exists(result['regexArgs'], regexCounter) then
                    regexCounter[result['regexArgs']]++
                break
        else
            //compare the values
            match = true
            //search type args in database args
            foreach result['args'] as dbArgIndex=>dbArgs do
                if dbArg['type'] equals resultNode['args'][dbArgIndex]['type'] and
                  dbArg['value'] not equals resultNode['args'][dbArgIndex]['type'] then
                    match = false
                    break
                else
                    match = false
                    break
            if match equals true then
                if key_exists(result['regex'], regexCounter) then
                    regexCounter[result['regex']]++
                break
```

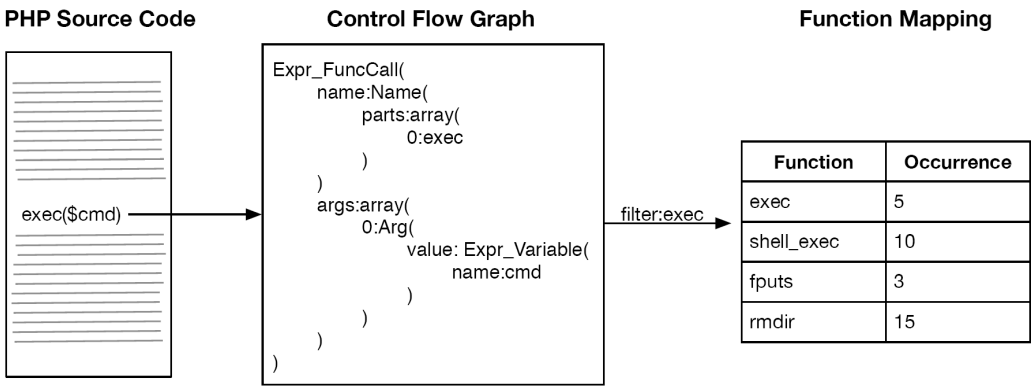

FIGURE 2. Process of sensitive function occurrence counting based on function call execution

CODE LISTING 2. Mongodb object data structure for sensitive function call

```
array(1){
[0]=>
array(4){
["positive"]=>
  array(5){
    ["exec"]=>int(1)
    ["fputs"]=>int(25)
    ["fclose"]=>int(12)
    ["rmdir"]=>int(0)
    ["fopen"]=>int(0)
  },
["negative"]=>
  array(5){
    ["exec"]=>int(0)
    ["fputs"]=>int(0)
    ["fclose"]=>int(0)
    ["rmdir"]=>int(0)
    ["fopen"]=>int(0)
  },
["positiveFileCount"]=>int(9),
["negativeFileCount"]=>int(6)
}
}
```

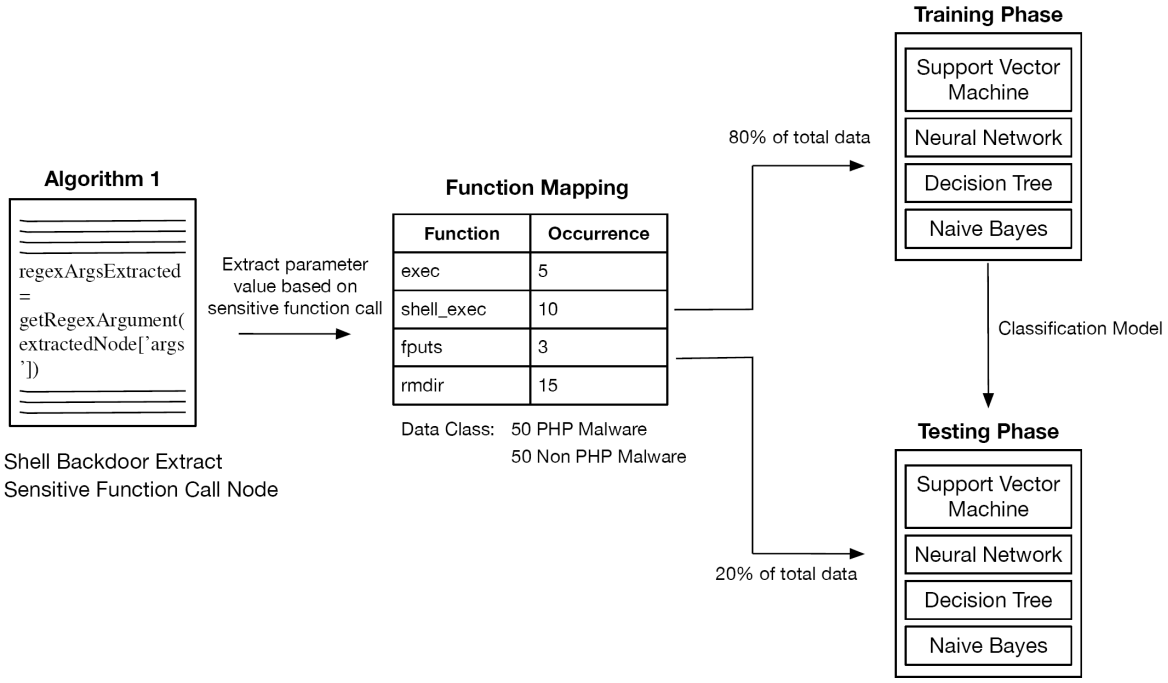

FIGURE 3. Flowchart of classification algorithm based on extracted parameter value with Algorithm 1

Figure 3 shows those classification process executed by using the numbers of sensitive function call extracted by Algorithm 1. The training phase used 80% random data to classify between malware and non-malware. The testing phase used 20% random data to test the classification correctly or not. The purposes of both phases are getting values of true positive (TP), true negative (TN), false positive (FP), and false negative (FN). These result values can be seen in Table 2.

4. **Result and Discussion.** This paper already trained 100 PHP source code files that consist of 50 PHP malware and 50 PHP non-malware. The classification experiment result

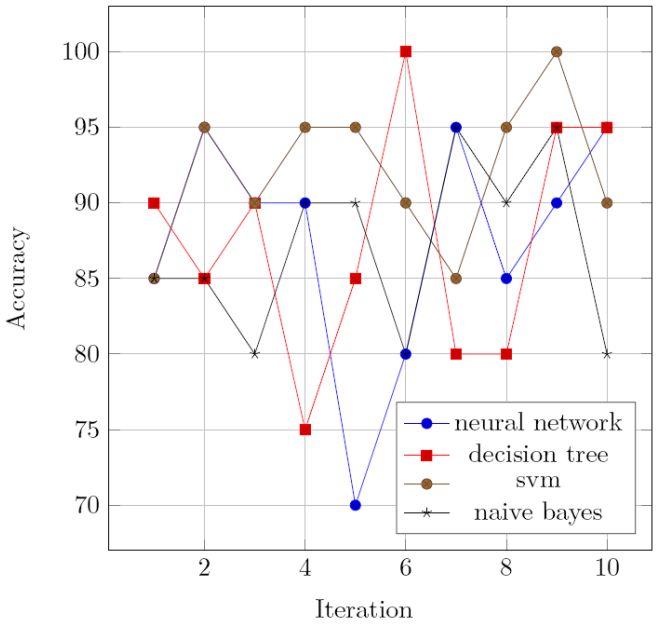

FIGURE 4. Comparison accuracy of classification algorithms

TABLE 2. Average training and classification result

Classification	TP	FN	FP	TN	Precision	Recall	Accuracy
Support Vector Machine	**9**	**1**	**1**	**9**	**0.9275**	**0.9295**	**0.92**
Neural Network	8	2	1	9	0.894	0.891	0.875
Decision Tree	9	1	2	8	0.885	0.8365	0.875
Naive Bayes	9	1	2	8	0.876	0.873	0.87

is tested on 20 random PHP file of 100. Those one hundred PHP files are through 10 iterations for each classification training algorithm. The precision result comparison can be seen in line chart at Figure 4.

Figure 4 shows that support vector machine (SVM) has the highest accuracy and is more stable compared to others. Support vector machine accuracy is 0.92 (92%) averagely. Table 2 shows the average value of those algorithms. For example, the accuracy value of SVM is 0.92 (92%) that shows the algorithm is able to classify 18.4 file malware or not malware correctly. Meanwhile, 1.6 files are not classified correctly. The precision value can reach 0.927 (92.7%). The recall value, which is 0.929 (92.9%), shows that the SVM is able to detect 9.29 shell backdoor files from 10 files of malware class correctly, and 9.29 shell backdoor files from 10 files of non-malware class correctly. The neural network and decision tree has the same accuracy, but the neural network is better on the precision. Support vector machine has the highest accuracy because it has regularization parameter that minimizes over-fitting compared to other algorithms. Support vector machine also has kernel trick that transforms data separated by hyperplane. The classification separates two classes more accurately based on altitude features or dimension.

5. **Conclusion.** Shell backdoor has many variety types. It is caused by the malware that can be crafted easily with utilized sensitive functions. Shell backdoor can be detected and classified by using how many sensitive function call is executed based on static analysis process. Those classification algorithms need static analysis to extract sensitive function call correctly. Our result shows SVM has the highest accuracy value, which is 0.92 (92%) and 0.927 (92.7%) for the precision value. Our next research will inspect more deeply

whether those sensitive function call executed with taint parameter to detect the shell backdoor more accurately with automata.

REFERENCES

[1] N. Christianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, Cambridge, 2000.

[2] B. Hagen and S. Behrens, *Web Shell Detection Using NeoPI*, 2013.

[3] D. Canali, D. Balzarotti and A. Francillon, The role of web hosting providers in detecting compromised websites, *Proc. of the 22nd International Conference on World Wide Web*, pp.177-188, 2013.

[4] B. Chess and J. West, *Secure Programming with Static Analysis*, Pearson Education, 2007.

[5] K. Cooper and L. Torczon, *Engineering a Compiler*, Elsevier, 2011.

[6] Emphosa, *Php Shell Detector: Web Shell Detection Tool*, 2011.

[7] R-fx Network, *Linux Malware Detect*, 2013.

[8] D. Grune, K. Van Reeuwijk, H. E. Bal, C. J. H. Jacobs and K. Langendoen, *Modern Compiler Design*, Springer Science & Business Media, 2012.

[9] M. T. Hagan, H. B. Demuth, M. H. Beale et al., *Neural Network Design*, Pws Pub., Boston, 1996.

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

[11] M. Xu, X. Chen and Y. Hu, Design of software to search ASP web shell, *Procedia Engineering*, vol.29, pp.123-127, 2012.

[12] Netcraft, *January 2018 Web Server Survey*, 2018.

[13] I. Rish, An empirical study of the naive Bayes classifier, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, vol.3, pp.41-46, 2001.

[14] S. R. Safavian and D. Landgrebe, A survey of decision tree classifier methodology, *IEEE Trans. Systems, Man, and Cybernetics*, vol.21, no.3, pp.660-674, 1991.

[15] Y. Tian, J. Wang, Z. Zhou and S. Zhou, CNN-webshell: Malicious web shell detection with convolutional neural network, *Proc. of the 2017 VI International Conference on Network, Communication and Computing*, New York, NY, USA, pp.75-79, 2017.

[16] J. T. Townsend, Theoretical analysis of an alphabetic confusion matrix, *Perception & Psychophysics*, vol.9, no.1, pp.40-50, 1971.

[17] A. Young and M. Yung, The dark side of "black-box" cryptography or: Should we trust capstone?, *Annual International Cryptology Conference*, pp.89-103, 1996.

[18] A. Young and M. Yung, *Malicious Cryptography: Exposing Cryptovirology*, John Wiley & Sons, 2004.

[19] A. Kurniawan, B. S. Abbas, A. Trisetyarso and S. M. Isa, Static taint analysis traversal with object oriented component for web file injection vulnerability pattern detection, *The 3rd International Conference on Computer Science and Computational Intelligence, Procedia Computer Science*, pp.596-605, 2018.