Journal of Theoretical and Applied Information Technology

30th September 2023. Vol.101. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

WEBSHELL DETECTION BASED ON BYTECODE FEATURE WITH CONVOLUTIONAL NEURAL NETWORK

DIAN ANGGRAINI[1], ABBA SUGANDA GIRSANG[2]

[1,2]Computer Science Department, BINUS Graduate Program – Master of Computer Science, Bina Nusantara University, Jakarta 11480, Indonesia

Email: [1]dian.anggraini@binus.ac.id, [2]agirsang@binus.edu

ABSTRACT

Web shell is a malicious program used to remotely access web servers during cyberattacks. Malicious web shells closely resemble benign web shells, making them difficult to distinguish. The challenge in detecting pre-existing web shells is that this type of malware is hard to detect using an intrusion detection system (IDS) or antivirus techniques. This is because web shells are usually hidden within web applications, making them challenging to differentiate from regular web application source code. Therefore, traditional detection models that analyze the dynamic features of web shell script execution are more effective in detecting existing malware attacks. In this study, A method of web shell detection based on dynamic bytecode features using a convolutional neural network (CNN) has been proposed in this research. Word2vec is employed to obtain vectorized features from the bytecode or opcode. Experimental results using a training dataset of 2577 samples and a validation dataset of 645 samples yield the best model with an accuracy of 99.86% at epoch 100. The experiments demonstrate that this model effectively detects web shells, with a significant increase in accuracy levels.

Keywords— *Web Shell, Machine Learning, CNN, Cyber Security, Opcode.*

1. INTRODUCTION

Technological advances in the modern world are developing rapidly, especially in the field of web-based applications. Almost all organizations and companies have adopted web-based applications to support their business processes. Web-based applications are very useful for organizations because they have the advantages of not needing installation, compatible across various platforms, less load on the system, and good adaptability, especially in today's cloud computing era. Currently, many web-based application developments are less aware of website security, so vulnerability assessment and penetration testing are not carried out before the application is launched. Along with the increase in cybercrime activities, there are security risks that can occur in web-based applications if they do not follow the guidelines issued by the government and related authorities.

The highest security risks found in web-based applications according to OWASP are broken access control, cryptographic failures, injection, insecure design, security misconfiguration, vulnerable and outdated components, identification and authentication failures, software and data integrity failures, security logging and monitoring failures, and server-side request forgery [1]. The types of attacks that include injection are SQLI, XSS, and backdoor shells.

A backdoor shell, commonly known as web shell, is a code that is compiled into a secret script used to control a website or server [2]. A web shell allows unauthorized access to a web server by bypassing the firewall through port 80 and other required authentication mechanisms. Web shell are easy to encode but difficult to detect [3]. Web shells, or malware, have the ability to behave differently depending on the program that is programmed to be executed. Therefore, it is important to understand its function well. There are two methods to understand the behavior of malware: static analysis and dynamic analysis. Static and dynamic analysis are two different approaches used in the field of computer security to examine and analyze the behavior of malware. Static analysis refers to the process of analyzing the source code or binary files without executing the software. Its purpose is to identify potential hazards and malware features before they spread. On the other hand, dynamic analysis involves testing and monitoring the system directly when it is running, with the aim of understanding the behavior of malware and preventing

its spread. Dynamic feature detection depends on observing the behavior of web shell files [4], [5], analyzing the communication traffic [6]–[8] associated with the web shell, and identifying other characteristics. This technique can be performed using methods such as an intrusion detection system (IDS) and antivirus software running in the background to detect suspicious activities. Detection methods based on dynamic traffic information have the potential to achieve high levels of accuracy in detecting security threats but are difficult to implement in practical applications. This is due to the difficulty in obtaining the necessary traffic information and also the difficulty in preventing web shell attacks that may occur first.

The first challenge in detecting web shells is that this type of malware is difficult to detect using an intrusion detection system (IDS) or antivirus techniques. This is because web shells are usually hidden inside web applications, making them difficult to distinguish from regular web application source code. Therefore, traditional detection models that analyze dynamic features of the web shell script execution process, such as eval execution context, file read and write operations, and other dynamic behaviors, are more effective in detecting malware attacks that have already occurred.

Previous research has been conducted on web shell detection based on the statistical features of source code analysis using the deep learning method by Tao et al. [9]. The statistical features capture specific file aspects, condensing web shell characteristics from a broader perspective. However, evolving web services have introduced code obfuscation, blurring the distinction between normal files and web shells, diminishing the effectiveness of statistical-based detection. This highlights the need for more comprehensive detection methods as web shells exhibit both file attributes and scripting language structure.

Meanwhile, prior studies on web shell detection based on dynamic features have utilized varying datasets and classification methods. Zhang et al [10] research employed TF-IDF and Word2vec with an ensemble algorithm approach. The best model obtained using Word2vec with an accuracy of 98.60%. Tianmin et al. [11] and Ai et al [12] utilized Ngram-TF-IDF with ensemble algorithm methods. In Tianmin et al [11] study, the classification method achieved an accuracy of 97.71% using XGBoost.

Considering the points mentioned earlier, it is evident that there are still shortcomings arising from the utilization of Ngram and TF-IDF and recognizing the mounting limitations of statistical features. Hence, we adopt dynamic feature detection by vectorizing PHP opcodes using word2vec to address the concern of improved word representation in detecting web shell scripts for effective web shell detection. A major contribution to our research is to improve the ability of detection models by analyzing the executable data characteristics of PHP code using the Convolutional Neural Network (CNN) algorithm. This can be more effective than relying on traditional statistical characteristics.

2. RELATED WORK

There are two main categories of methods used to detect traditional web shells: static feature detection, which relies on identifying characteristics of web shell files, and dynamic feature detection, which focuses on analyzing the behavior of web shell file processes. The features extracted from web shell scripts can be categorized into five distinct classes based on their properties: lexical features, syntax, semantic, statistical, and abstract[13]. Abstract features are used to refer to vectorized data such as source code, opcodes, and web traffic.

2.1 Static Feature Detection

Static feature detection is usually done using reverse engineering and static analysis techniques, where the code is broken down into individual instructions and inspected manually or automatically to find patterns or features associated with malware. To perform simple pattern matching, static detection can be done on a few feature strings using regular expressions. These feature strings generally contain high-risk code, such as the eval() and system() functions from the system functions, as well as specific markers of classic web shells. Some security products have built specific static feature rule libraries by collecting existing web shell samples. If the tested file meets certain detection rules, the security product will issue a warning.

Li et al. [14] proposed a detection method using an optimal malicious signature, a sample malicious function, and the longest character at the beginning and end of the file. The method used to detect it uses RNN-GRU. The specialty of this model lies in its focus on capturing word associations that are on a single line in the source script. Therefore, each line in the web shell source is assigned a vector of words. The level of accuracy in this RNN-GRU model is 98.94%.

However, the performance of this model is slightly lower in detecting PHP web shell compared to research that proposes the FRF-WD ensemble learner method.

Fang et al [15] suggested a detection method using ensemble learning, i.e., a random forest classifier, by combining it with FastText. FastText classification training is carried out on the opcode sequence derived from the PHP script. The prediction values obtained from FastText are combined with five statistical features to create input for random forest training and prediction. Experimental results based on 10-fold cross-validation showed that the model performed excellently, and accuracy reached 99.23%.

2.2 Dinamic Feature Detection

Dynamic feature detection based on the execution of a web shell file process aims to identify the level of abnormal opcode called by the runtime to determine malicious web shells and benign web shells. This is done using machine learning algorithms that are able to learn certain patterns associated with malware behavior, such as unusual pattern execution instructions or access to suspicious system files.

Ai et al. [12] proposed an ensemble model based on binary weighted voting determined by the accuracy of each classification to classify web shells. The ensemble detection model is called WS-LSMR, consisting of logistic regression (LR), support vector machine (SVM), multi-layer perceptron (MLP), and random forest (RF). To determine the weight of each basic classification, a well-defined formula is used based on accuracy at the time of training. Furthermore, the basic classification is trained and tested on 4-gram-based TF-IDF opcode vectorization in a well-defined feature selection algorithm. The accuracy resulting from the combination of four algorithms is 94.28%. However, the performance of the WS-LSMR model, which is a combination of single, ensemble, and deep learning, is slightly lower than the research suggested with the deep learning method.

Wang et al [16] proposed a method of web shell detection based on a multilayer neural network, or MLP, using two hidden layers. This method uses the extraction feature with bigram optimization of PHP opcode, then the opcode sample is performed with word frequency analysis using TF-IDF to obtain the level of interest of each sample in the sample set. The accuracy rate of the MLP model for web shell detection is 94.4%. The accuracy level of this model is still lower than research using ensemble learning.

Zhang et al [10] suggested a web shell detection model based on the PHP opcode feature using TF-IDF text processing and then compared it to Word2vec. The study used the ensemble learning method XGBoost by comparing accuracy levels with four different algorithms, namely, neural networks, random forests, and SVM. The results of the evaluation of the four models found that XGBoost performed best, with the highest accuracy of 98.20%.

2.3 Malicious Webshell Detection on Deep Learning

Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in ways that inspire the human brain. Deep learning is considered to be the most important advancement in the field of computer science recently, and it has had a broad impact in almost all fields of science. These advances have disrupted and transformed industries. Currently, there is competition between leading companies in the fields of economics and technology to drive advances in deep learning[17]. Deep learning, also known as neural networks, can be classified into several types, one of which is the convolutional neural network (CNN). CNN is a type of neural network architecture that is developed specifically for processing data that has a grid or matrix structure, such as images, videos, and other data that are arranged in a similar way. Yong et al. [18] proposed a web shell detection model based on PHP opcode features using CNN. The accuracy level of this model is 98.22%.

3. RESEARCH METODOLOGY

3.1 Design Architecture

A method of web shell detection based on dynamic bytecode features using a convolutional neural network (CNN) has been proposed in this research. The research is divided into four stages: preprocessing, feature selection, classification, and evaluation, as illustrated in the flowchart in Figure 1.

Preprocessing: First, all datasets in the web shell data set must be filtered to only have PHP files and also deduplicated to prevent interference with the detection results. The process of extracting PHP code into bytecode or opcode. Opcode is used to process feature selection by distinguishing between malicious

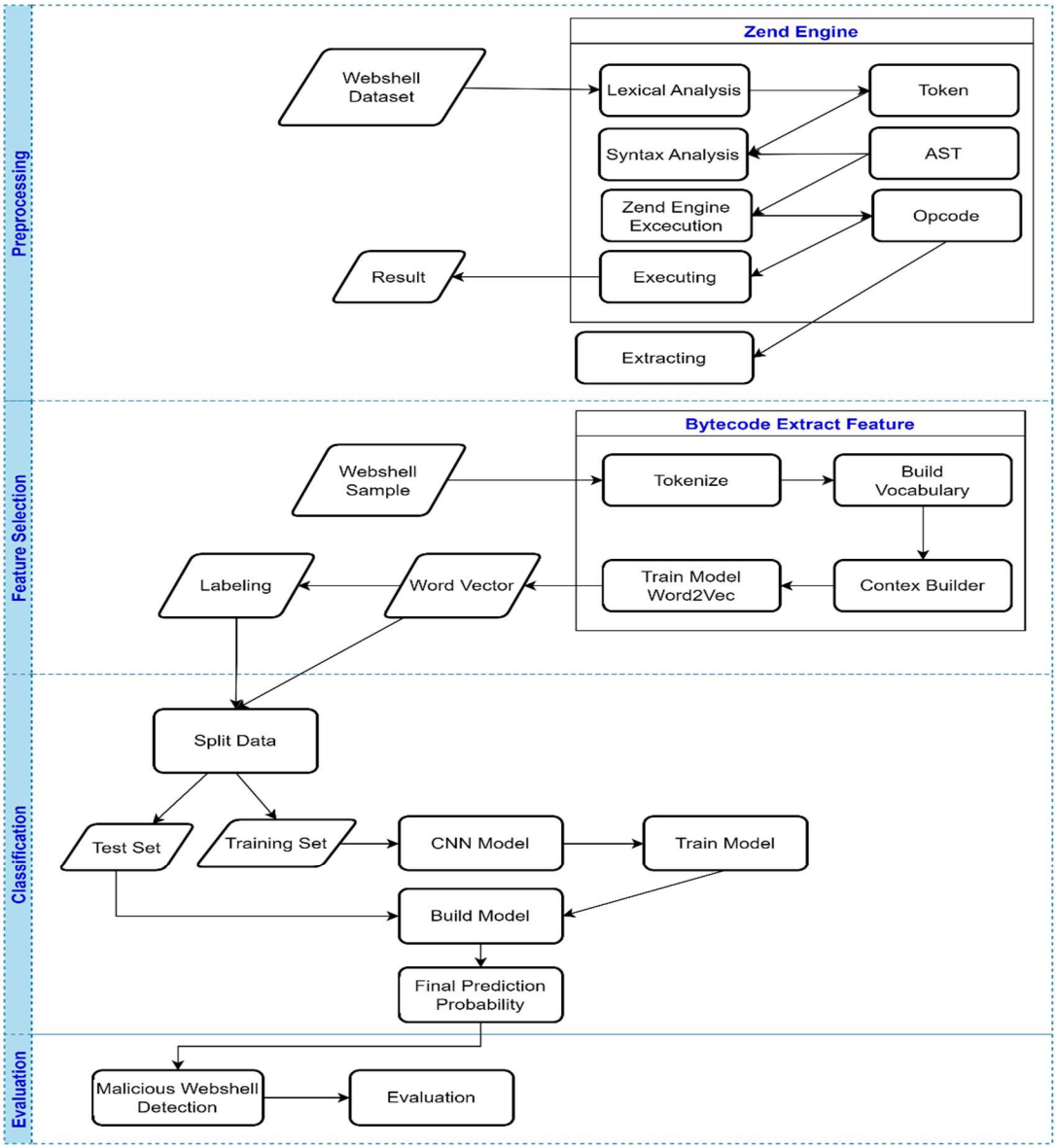

Figure 1: Flowchart Desain Architecture

and benign files. Feature selection: This process involves selecting features from a set of PHP opcodes. First, the extracted PHP code is tokenized using regular expressions [19]. Then, the tokenized opcodes undergo word embeddings using word2vec to convert words into numbers (vectors). The next step is to label the malicious and benign web shells and split the data into two parts, namely the training data and the testing data. Classification: In this process, the CNN model is developed using three convolutional layers. After the CNN model is constructed, it is trained using the training data. Once the CNN model has been trained, the best model is then evaluated. Evaluation: This process is the final stage in detecting PHP web shells. This process uses testing data to generate predictions of malicious web shells so that the model's accuracy can be evaluated.

3.2 PHP Opcode

PHP is a programming language that is executed directly by the PHP virtual machine at runtime. The PHP virtual machine used is Zend Engine, which processes PHP code and translates it into bytecode instructions that are then executed by the virtual machine. In addition, Zend Engine also supports Just-In-Time (JIT) technology, which allows PHP code to be compiled into machine code directly at runtime to improve code execution performance. This allows the

PHP virtual machine to execute PHP code more quickly and efficiently. The process of executing PHP code is divided into three stages: lexical analysis, syntax analysis, and Zend Engine execution. The flowchart for the PHP code execution process is illustrated in Figure 1.

Lexical analysis refers to the transformation of the character sequence in a source code file into a sequence of tokens. This process aids in the easier processing of the code by a compiler or interpreter. It is typically the initial phase of the compilation process, followed by syntactic and semantic analysis. In lexical analysis, the source code is scanned character by character, and tokens are created by grouping characters according to the programming language's rules. These tokens represent fundamental components of the program's syntax, such as keywords, identifiers, punctuation, and constants. The lexer, also known as a tokenizer, performs the task of lexical analysis. The outcome of this phase is a stream of tokens that can be more efficiently processed by the syntax analyzer, which ensures the program's syntax and structure are correct.

Syntax analysis, or parsing, refers to the examination of a sequence of symbols in either a natural language or a computer language based on predefined grammar rules. Its primary objective is to assess whether the provided input conforms to the syntax rules of the language. In the context of natural language processing, syntax analysis plays a vital role in comprehending and interpreting the structure of sentences. This entails identifying the various parts of speech (nouns, verbs, adjectives, etc.), establishing relationships between words (such as subject-verb agreement), and constructing a hierarchical representation of the sentence known as a parse tree or Abstract Syntax Tree (AST). After the syntactic analysis, the next step is the execution stage of the Zend Engine, which compiles bytecode by reading the Abstract Syntax Tree (AST) and translating operator nodes into corresponding bytecode, also known as opcode. The next process is code execution. The Zend Engine executes the sequence of opcodes one by one to run the PHP program. In extracting PHP code into bytecode or opcode, two tools can be used, namely Vulcan Logic Disassembler (VLD) [19] by Derick Rethans and PHP Debugger (phpdbg) [20], which is part of the PHP project. Vulcan logic disassembler works by reading the PHP file and then extracting the bytecode contained in it. After successfully retrieving the bytecode, it will read the instructions in the bytecode and display the output in a more human-readable format. Meanwhile, PHP debugger works by executing PHP code in debug mode. When debug mode is activated, it will display information about the opcode instructions executed by the PHP code, as well as variable values and other debugging information. Generally, vulcan logic disassembler is more suitable for analyzing PHP code, while PHP debugger is more suitable for finding errors or bugs in PHP code. In this paper, the usage of vulcan logic disassembler is employed to extract PHP code into bytecode for the analysis of malicious PHP files as follows:

```
<?php if ($_POST['cmd']){ $cmd = $_POST['cmd']; passthru($cmd);}?>
```

After the Zend Engine execution process on the above PHP file, the following opcode sequence is generated: [FETCH_R], [FETCH_DIM_R], [JMPZ], [FETCH_R], [FETCH_DIM_R], [ASSIGN], [INIT_FCALL], [SEND_VAR], [DO_ICALL], [ECHO], [RETURN] as seen in Table 1. After collecting the sequence of opcode instructions, our subsequent step involves analyzing the malicious PHP files.

Table 1: The Sequence of Opcode Instructions

No.	Opcode
0	FETCH_R
1	FETCH_DIM_R
2	JMPZ
3	FETCH_R
4	FETCH_DIM_R
5	ASSIGN
6	INIT_FCALL
7	SEND_VAR
8	DO_ICALL
9	ECHO
10	RETURN

3.3 Feature Selection

Feature selection is a method of selecting the most effective features from the available ones. In this study, these features function to distinguish between malicious and non-malicious files. Additionally, it plays a role in improving algorithm efficiency by ensuring appropriate index evaluation. Since most opcode features only use limited words in the vocabulary, this will result in sparse word vectors. To maintain computational efficiency, it is imperative to eliminate unnecessary features that could potentially result in a decline in performance.

Word2vec is a method in natural language processing used to represent words as numeric vectors that was introduced by Google in 2013.[21]. It has become one of the most popular Natural Language Processing (NLP) techniques for word vectorization [22]–[24]. The purpose of word2vec is to estimate the distribution representation of the words, so that the words that often appear in the same context will have similar representations in the vector space. Word2vec utilizes machine learning techniques based on neural networks, consisting of two model architectures: Continuous Bag-of-Words (CBOW) and Skip-Gram. In the CBOW architecture, the model predicts the target word based on the given context words, while in the Skip-Gram architecture, the model predicts the context words based on the given target word. In this study, the Continuous Bag-of-Words (CBOW) architecture was employed for constructing the word2vec model. The word2vec learning process is conducted by calculating the likelihood or probability of words appearing in a given context using the softmax function. Subsequently, the model parameters are updated based on the loss value generated in each iteration of the learning process. Once the learning process is complete, each word is represented as a numeric vector with a specific dimension, where the vector represents the word in a semantic vector space. For example, words with similar meanings will have vector representations that are close to each other in the vector space. The model parameters used in this study can be seen in Algorithm 1. The next step is to vectorize PHP opcodes using the pre-trained Word2Vec model, as outlined in Algorithm 2.

Algorithm 1 *Training Model Word2Vec*

Input : opcode_php_list
Output : model Word2Vec

1: Creating a word2vec model using the CBOW architecture with parameters: vector size=200, window=5, min_count=5 and epoch=10
2: Build vocabulary from opcode_php_list
3: Training models with corpus_count and epoch models
4: Saving the word2vec model.

Algorithm 2 *PHP Opcode Vectorization*

Input : model, opcode_php_list
Output : opcode vector

1: Determine the dimensions of embedding = model vector_size
2: **For** text → *opcode_php_list* **do**
3: Cut the words according to the maximum length of the document = 500
4: Use of word embedding
5: **End for**
6: Return *opcode_vector*

3.4 Classification

After feature selection from the set of opcode and bytecode samples is completed, the feature matrix resulting from opcode vectorization is used as the input, and the labeling of web shell or normal files is used as the output. Then, classification is employed for training. Before that, the data is divided into three sets: training dataset, validation dataset, and testing dataset. In this study, the data will be divided into training and testing sets with an 80:20 ratio.

After separating the dataset, the next step is testing the algorithm model, Convolutional Neural Network (CNN). Convolutional neural networks (CNN) are a type of neural network used specifically for computer vision tasks such as image recognition or classification. CNN consists of five layers: the input layer, the convolutional layer, and the fully connected layer. In the formation of a CNN classification model using the TensorFlow and TFLearn libraries, three convolutional layers will be used, as seen in Figure 2.

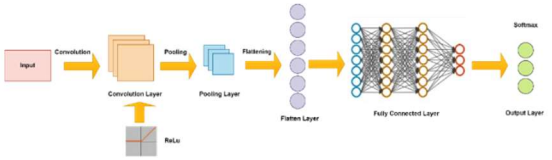

Figure 2: CNN Model Process

The step-by-step process performed in the Convolutional Neural Network (CNN) is as follows:

1. Input layer: The input data consists of documents with dimensions ['none', 500, 200]. 'None' indicates that the batch size can vary; 500 is the maximum document length allowed in the dataset, and 200 represents the dimension of each word in the document.
2. Convolutional layer: Next, create three branches of convolutional layers. Each branch has 200 filters, kernel sizes of 3, 4, and 5, respectively, a ReLU activation function, and L2 regularization. Each branch processes the input data in parallel.
3. Flatten layer: The output results from the three branches of the convolutional layer are merged into one using the merge function with **'concat'** mode and axis = 1.

4. Pooling layer: Global max pooling is used to generate the best feature value from the previous feature extraction.
5. Dropout layer: A dropout layer with a probability value of 0.8 is used to prevent overfitting.
6. Fully connected layer: A fully connected layer with two output units and a softmax activation function is used to generate the classification output probabilities.
7. Regression function: The regression function is used to determine the loss function. In this implementation, the 'adam' optimizer is used with a learning rate of 0.001 and the categorical_crossentropy loss function
8. DNN model: The CNN model is initialized using tflearn. DNN function with the network as the input and tensorboard_verbose set to 0 to avoid verbose display during model training.

3.5 Evaluation

At this stage, evaluation and analysis are performed by comparing the accuracy of the model. Several metrics such as accuracy (ACC), precision (P), recall (R), and F1-score (F1) are used to evaluate the performance of the model in classifying malicious or benign web shells along with the confusion matrix to measure the model's performance in terms of correctly and incorrectly classified data from the tested dataset. The evaluation confusion matrix of the model is shown in Table 2.

Table 2: Confusion Matrix

Prediction	Actual	
	True	**False**
True	True Positive (TP)	False Positive (FP)
False	False Negative (FN)	True Negative (TN)

True positive is the result where the model correctly predicts the positive class (True Positive = TP). For example, if an instance is a benign (normal) sample and it is predicted as a benign (normal) sample, it is considered a true positive (TP). False negative is the result where the model incorrectly predicts the negative class (False Negative = FN). For example, if an instance is a malicious (web shell) sample but it is predicted as a benign (normal) sample, it is considered a false negative (FN). True negative is the result where the model correctly predicts the negative class (True Negative = TN). For example, if an instance is a malicious (web shell) sample and it is predicted as a malicious (web shell) sample, it is considered a true negative (TN). False positive is the result where the model incorrectly predicts the positive class (False Positive = FP). For example, if an instance is a benign sample but it is predicted as a malicious (web shell) sample, it is considered a false positive (FP). From the performance metrics of this study, the following are the formulas for each matrix related to the actual versus predicted values in formulas 1 to 4.

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN} \quad (1)$$

Eq. (1) shows the number of proportions of samples correctly classified by the model prediction.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

Eq. (2) shows the sum of the proportion of the true-positive prediction to the total positive prediction.

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

Eq. (3) shows the exact number of predictions of the correct actual number.

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

Eq. (4) shows the merger of information about precision and recall. F1 scores provide a more holistic measurement of model performance, as it considers both the number of false negative and false positive in model predictions. The evaluation results of the model built using the CNN algorithm will be compared with the Random Forest (RF) and XGBoost models.

4. EXPERIMENT ANALYSIS

4.1 Data Source

The data used for this research consists of a collection of PHP backdoor web shell datasets available for download from the internet. The dataset consists of several open-source web shell projects as malicious samples, while some CMS projects, the Yii framework, and OA are used as normal samples, as shown in Table 3. We found that there is no readily available collection of PHP web shell data that has been cleaned and processed on the internet. Therefore, samples of PHP web shell samples from seven open-source projects that have been collected were then analyzed manually.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 3: Data Source of PHP Samples

Sample	Source
Malicious Sample	https://github.com/JohnTroony/php-webshells
	https://github.com/BlackArch/webshells
	https://github.com/LuciferoO/webshell-collector
	https://github.com/xl7dev/WebShell
	https://github.com/tennc/webshell
	https://github.com/webshellpub/awsome-webshell
	https://github.com/mattiasgeniar/php-exploit-scripts
Normal Sample	https://github.com/yiisoft/yii2
	https://github.com/rainrocka/xinhu
	https://github.com/johnshen/phpcms

After collecting the data, the next step is to perform data cleaning on the PHP web shell samples used from seven open-source projects. The data cleaning steps included removing duplicate data so that there was no unnecessary repetition of data. In addition, web shell data that could not be run due to syntax errors in the program code was also removed. This data cleaning is done so that the data used in the research becomes more valid and reliable. Clean and structured data will make it easier to conduct analysis and minimize the possibility of errors in drawing conclusions. The results of data cleaning on the web shell have a total of 1461 samples, and data on normal samples have a total of 2528 samples. The data is divided into 3 categories: training data with a total of 2577, validation data with a total of 645, and testing data with a total of 767.

4.2 Experiment Environmental

The experiments conducted in this research were based on the Python 3.10 programming language and executed in an experimental environment using the 64-bit Windows 11 operating system. For more detailed information, please refer to Table 4.

Table 4: Experiment Environmental

Hardware and Software	Specification
Memory	12 Gb
Processor (CPU)	Intel Core i7-8750H CPU @ 2.20 Ghz (12 CPUs)
Graphics (GPU)	GeForce GTX 1050 4 Gb
Operating System (OS)	Win11 64bit
CUDA	NVIDIA CUDA 12.0
Language Programming	Python 3.10.6
PHP	PHP 8.1.16
Machine Learning Library	Tensorflow 2.11.0 and Tflearn 0.5.0

4.3 Preprocessing Data

In this stage, cleaning is performed on the collected PHP web shell files. The purpose of this process is to extract the PHP web shell files into opcodes to obtain a set of features that can detect malicious PHP web shell files in token form. The following processes are carried out:

4.3.1 Opcode PHP Extraction

The process conducted in this stage involves extracting PHP web shell files into PHP opcodes using the VLD (Vulcan Logic Dumper) extension, as explained in Chapter III. The extracted PHP opcode results are shown in Figure 3.

Figure 3: PHP Web Shell Opcode Results

4.3.2 Tokenizing

In this process, numbers, punctuation marks, and other characters that are considered to have no influence on PHP opcode processing are also removed. In this study, the "re" module's findall() function is used to find "all" occurrences that match the given pattern. The results of tokenization using the findall module can be seen in Figure 4.

Figure 4: PHP Opcode Tokenization Result

4.4 Feature Selection

After the tokenizing stage, the next step is to create a word2vec model using the CBOW architecture with a window size of 5. Then, the opcode PHP is

Journal of Theoretical and Applied Information Technology

30th September 2023. Vol.101. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

vectorized using the trained model. This process will generate numerical vector representations for each opcode word in the PHP corpus. The vectorization results can be seen in Figure 5.

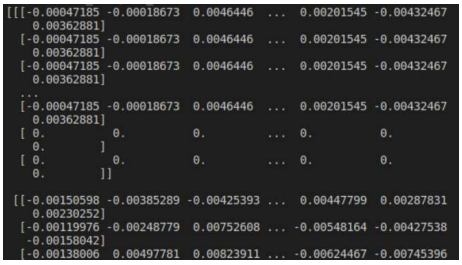

Figure 5: PHP Opcode Vectorization Results

4.5 Classification

The classification process is performed by applying the CNN algorithm, which consists of three convolutional layers. The classification process is divided into two parts: training and validation results, as well as testing results.

4.5.1 Training Result

The CNN model training results were performed with epoch 10 to 100. The results of the experiment using training data 2577 with validation data 645 obtained the comparison of accuracy and validation precision that can be seen in Figure 6. Comparison of data loss and validation in Figure 7.

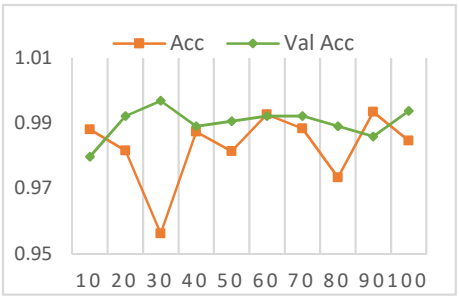

Figure 6: The Effects of Different Epochs on Accuracy and Validation Accuracy

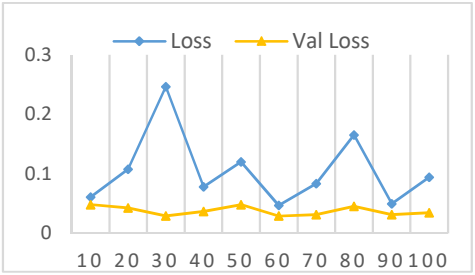

Figure 7: The Effects of Different Epochs on Loss and Validation Loss

4.5.2 Testing Result

In the testing phase, PHP opcodes are tested by inputting them into the pre-existing CNN model. If the prediction result for the test data shows a prediction value of less than 50%, the opcode will be classified as black, indicating that it is a web shell. However, if the prediction value is greater than 50%, the opcode will be classified as white or a normal file. Pada Table 5 shows examples of the prediction results from the model testing.

Table 5: CNN Model Testing Prediction Results

Opcode PHP Testing	Prediction Result	Classification
ECHO BEGIN_SILENCE INIT_FCALL FETCH_R FETCH_DIM_R SEND_VAL DO_ICALL ASSIGN END_SILENCE ECHO ISSET_ISEMPTY_CV BOOL_NOT JMPZ L …	0.25%	Black (Web Shell)
RETURN FETCH_R ASSIGN_OBJ OP_DATA FETCH_R ASSIGN_OBJ OP_DATA FETCH_OBJ_R FETCH_OBJ_R ASSIGN_OBJ OP_DATA FETCH_OBJ_R FETCH_OBJ_R ASSIGN_OBJ OP_DATA …	99.9%	White (Normal)

4.6 Evaluation Result

After testing the PHP opcodes on the CNN model using Word2Vec, the resulting confusion matrix can be seen in Table 6. he confusion matrix obtained from the testing shows that there are 503 samples predicted as non-malicious or normal (True Positive), 262 samples predicted as malicious or web shell (True Negative), 2 samples predicted as malicious but actually non-malicious (False Positive), and 0 samples predicted as non-malicious but actually malicious (False Negative). Based on the confusion matrix data, the accuracy, precision, and F1-score results from epochs 10 to 100 are shown in Table 7. In Figure 8 , it can be observed that the influence of each epoch can determine different metric results, leading to fluctuations in accuracy for each epoch. However, at epoch 100, the best accuracy is achieved.

Table 6: Confusion Matrix Result

Prediction	Actual	
	True	False
True	504 (TP)	1 (FP)

False	0 (FN)	262 (TN)

Table 7: Model Performance Results

Epoch	Accuracy	Precision	Recall	F1-Score
10	0.993481	0.981273	1.0	0.990548
20	0.994784	0.988636	0.996183	0.992395
30	0.989569	0.970370	1.0	0.984962
40	0.992177	0.981203	0.996183	0.988636
50	0.993481	0.984905	0.996183	0.990512
60	**0.997392**	0.992424	1.0	0.996197
70	0.992177	0.977611	1.0	0.988679
80	0.989569	0.970370	1.0	0.984962
90	0.992177	0.977611	1.0	0.988679
100	**0.998696**	0.996197	1.0	0.998095

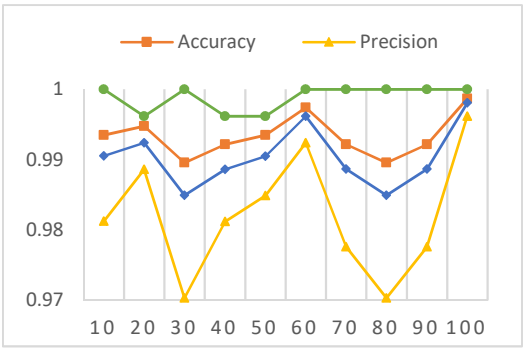

Figure 8: Effect of Different Epoch on Model Performance

To compare the performance of the CNN model, testing is conducted using ensemble learning algorithms, namely Random Forest and XGBoost. The Random Forest classification utilizes a base classifier with 100 estimators as its parameter. The Random Forest classification model is then tested, and the testing results yield an accuracy of 98.56% with a corresponding confusion matrix, which can be seen in the following Table 8.

Table 8: Random Forest Confusion Matrix Result

Prediction	Actual	
	True	**False**
True	495 (TP)	10 (FP)
False	1 (FN)	261 (TN)

The XGBoost classification utilizes a base classifier with the parameter objective = "reg:squarederror", which performs regression with the objective of minimizing the squared error between predicted values and actual values. The XGBoost classification model is then tested, and the testing results yield an accuracy of 98.56% with a corresponding confusion matrix, which can be seen in the following Table 9.

Table 9: XGBoost Confusion Matrix Result

Prediction	Actual	
	True	**False**
True	495 (TP)	10 (FP)
False	1 (FN)	261 (TN)

From the testing results using the two ensemble methods, a comparison of accuracy with the CNN model is obtained, as can be seen in Table 10. The Random Forest and XGBoost classification models have the same accuracy, which is 98.56%, while the CNN model has a higher accuracy of 99.86%. This indicates that the CNN model performs better compared to Random Forest and XGBoost.

Table 10: Comparison Results with Other Machine Learning Methods

Model	Accuracy	Precision	Recall	F1-Score
CNN	**0.998696**	0.996197	1.0	0.998095
RF	0.985658	0.963099	0.996183	0.992395
Xgboost	0.985658	0.963099	0.996183	0.992395

4.7 CNN Model's Comparison with Related Works

To establish the superiority of our model compared to previous research, a comprehensive comparison with other existing methods, namely WS-LSMR's method[12], Zang's method[10] dan Wang's method[16]. Through this comparison, the aim is to demonstrate that these methods are outperformed by our model in terms of various performance metrics, showcasing its enhanced effectiveness and efficiency. The experimental results can be seen in Table 11.

Table 11: Comparison of the model's performance with related work

Model	Precision	Accuracy	Recall
Word2Vec-CNN	0.996197	**0.998696**	1.0
WS-LSMR	N/A	0.9428	0.9914
Zhang's Method	0.9403	0.9860	0.9562
Wang's Method	0.932	0.944	0.968

5. CONCLUSION

This research proposes the utilization of deep learning methods employing the Convolutional Neural Network (CNN) algorithm to classify malicious PHP web shells. The testing results demonstrate that the implementation of the CNN algorithm with Word2Vec achieves an accuracy of 99.8%. We present a comparison between the CNN method and ensemble

learning methods, namely Random Forest and XGBoost. The tests conducted reveal that CNN outperforms Random Forest and XGBoost in terms of accuracy. Furthermore, our model exhibits superior performance compared to previous research findings.

REFERENCES:

[1] OWASP, "OWASP Top 10 Web Application Security Risks," 2021. https://owasp.org/www-project-top-ten/ (accessed Aug. 26, 2022).

[2] M. Siddik Hasibuan and L. Mashur Gultom, "Analisis Serangan Deface Menggunakan Backdoor Shell Pada Website Analysis of Deface Attacks Using Backdoor Shell On Websites," 2018.

[3] L. Qi, R. Kong, Y. Lu, and H. Zhuang, "An End-to-End Detection Method for WebShell with Deep Learning," in *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, 2018, pp. 660–665. doi: 10.1109/IMCCC.2018.00143.

[4] Z. Pan, Y. Chen, Y. Chen, Y. Shen, and X. Guo, "Webshell detection based on executable data characteristics of PHP code," *Wirel Commun Mob Comput*, vol. 2021, 2021, doi: 10.1155/2021/5533963.

[5] W. Kang, S. Zhong, K. Chen, J. Lai, and G. Xu, "RF-AdaCost: WebShell Detection Method that Combines Statistical Features and Opcode," 2020, pp. 667–682. doi: 10.1007/978-981-15-9739-8_49.

[6] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, "CNN-Webshell: Malicious Web Shell Detection with Convolutional Neural Network," Mar. 2017, pp. 75–79. doi: 10.1145/3171592.3171593.

[7] H. Zhang *et al.*, "Webshell Traffic Detection With Character-Level Features Based on Deep Learning," *IEEE Access*, vol. 6, pp. 75268–75277, 2018, doi: 10.1109/ACCESS.2018.2882517.

[8] W. Yang, B. Sun, and B. Cui, "A Webshell Detection Technology Based on HTTP Traffic Analysis," in *Innovative Mobile and Internet Services in Ubiquitous Computing*, F. and J. N. and E. T. Barolli Leonard and Xhafa, Ed., Cham: Springer International Publishing, 2019, pp. 336–342.

[9] Tao Fangjian, C. Cao, and Liu Zhihui, "Webshell Detection Model Based on Deep Learning," in *Artificial Intelligence and Security*, Z. and B. E. Sun Xingming and Pan, Ed., Cham: Springer International Publishing, 2019, pp. 408–420.

[10] H. Zhang, M. Liu, Z. Yue, Z. Xue, Y. Shi, and X. He, "A PHP and JSP Web Shell Detection System with Text Processing Based on Machine Learning," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 1584–1591. doi: 10.1109/TrustCom50675.2020.00219.

[11] G. Tianmin, Z. Jiemin, and M. Jian, "Research on Webshell Detection Method Based on Machine Learning," in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, 2019, pp. 1391–1394. doi: 10.1109/EITCE47263.2019.9094767.

[12] Z. Ai, N. Luktarhan, Y. Zhao, and C. Tang, "WS-LSMR: Malicious WebShell Detection Algorithm Based on Ensemble Learning," *IEEE Access*, vol. 8, pp. 75785–75797, 2020, doi: 10.1109/ACCESS.2020.2989304.

[13] A. Hannousse and S. Yahiouche, "Handling webshell attacks: A systematic mapping and survey," *Comput Secur*, vol. 108, p. 102366, Mar. 2021, doi: 10.1016/j.cose.2021.102366.

[14] T. Li, C. Ren, Y. Fu, J. Xu, J. Guo, and X. Chen, "Webshell Detection Based on the Word Attention Mechanism," *IEEE Access*, vol. 7, pp. 185140–185147, 2019, doi: 10.1109/ACCESS.2019.2959950.

[15] Y. Fang, Y. Qiu, L. Liu, and C. Huang, "Detecting Webshell Based on Random Forest with FastText," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, in ICCAI 2018. New York, NY, USA: Association for Computing Machinery, 2018, pp. 52–56. doi: 10.1145/3194452.3194470.

[16] Z. Wang, J. Yang, M. Dai, R. Xu, X. Liang, and others, "A Method of Detecting Webshell Based on Multi-layer Perception," *Academic Journal of Computing & Information Science*, vol. 2, no. 1, 2019.

[17] A. Shrestha and A. Mahmood, "Review of Deep Learning Algorithms and Architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019, doi: 10.1109/ACCESS.2019.2912200.

[18] B. Yong, X. Liu, Y. Liu, H. Yin, L. Huang, and Q. Zhou, "Web Behavior Detection Based on Deep Neural Network," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBD Com/IOP/SCI)*, 2018, pp. 1911–1916. doi: 10.1109/SmartWorld.2018.00320.

[19] Derick Rethans, "VLD," *pecl.php.net*, 2003. https://pecl.php.net/package/vld (accessed Apr. 04, 2023).

[20] "Interactive PHP Debugger," *php.net*, 2001. https://www.php.net/PHPdbg (accessed Apr. 04, 2023).

[21] T. Mikolov, K. Chen, G. s Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *Proceedings of Workshop at ICLR*, vol. 2013, Aug. 2013.

[22] S. Al-Saqqa and A. Awajan, "The Use of Word2vec Model in Sentiment Analysis: A Survey," in *Proceedings of the 2019 International Conference on Artificial Intelligence, Robotics and Control*, in AIRC '19. New York, NY, USA: Association for Computing Machinery, 2020, pp. 39–43. doi: 10.1145/3388218.3388229.

[23] W. Tian, J. Li, and H. Li, "A Method of Feature Selection Based on Word2Vec in Text Categorization," in *2018 37th Chinese Control Conference (CCC)*, 2018, pp. 9452–9455. doi: 10.23919/ChiCC.2018.8483345.

[24] V. Vargas-Calderón and J. E. Camargo, "Characterization of citizens using word2vec and latent topic analysis in a large set of tweets," *Cities*, vol. 92, pp. 187–196, 2019, doi: https://doi.org/10.1016/j.cities.2019.03.019.