Poster: Long PHP webshell files detection based on sliding window attention

Zhiqiang Wang
Beijing Electronic Science
& Technology Institute, Beijing, China
wangzq@besti.edu.cn

Haoyu Wang
Beijing Electronic Science
& Technology Institute, Beijing, China
20232909@mail.besti.edu.cn

Lu Hao
Beijing Municipal Public
Security Bureau, Beijing, China
hlucky@2008.sina.com

***Abstract*—Webshell is a type of backdoor, and web applications are widely exposed to webshell injection attacks. Therefore, it is important to study webshell detection techniques. In this study, we propose a webshell detection method. We first convert PHP source code to opcodes and then extract Opcode Double-Tuples (ODTs). Next, we combine CodeBert and FastText models for feature representation and classification. To address the challenge that deep learning methods have difficulty detecting long webshell files, we introduce a sliding window attention mechanism. This approach effectively captures malicious behavior within long files. Experimental results show that our method reaches high accuracy in webshell detection, solving the problem of traditional methods that struggle to address new webshell variants and anti-detection techniques.**

I. Introduction

The webshell injection plays a vital role in the hacker attack chain, enabling the attacker to remotely control devices, acquire sensitive data, and further expand attack activities. Therefore, Detecting and removing webshells is an effective way to defend against attacks and ensure web security.

Traditional webshell detection methods [1], [2] based on pattern matching usually rely on recognizing known features, including source code features, traffic features, dynamic function calls and other relevant features. However, as attack techniques evolve, the variability and obfuscation of webshells have become more prevalent. Attackers often use obfuscation, dynamic loading, encryption and decryption techniques to evade detection, making traditional detection methods inadequate for recognizing new types of webshells.

In this context, webshell detection methods using deep learning [3], [4], [5], including those based on source code or opcode, have become a research hotspot and have shown promising results. However, current deep learning-based webshell detection methods still face challenges [6]. For datasets, publicly available datasets are outdated and do not contain the latest samples. Therefore, their performance in real-world environments for detecting may not be good. For data processing, a good data processing method is often more important than the detection model. The opcode-based detection methods typically extract only a single sequence of opcode instructions (called Opcode Single-Tuples) without effectively capturing low-level code features. The source code-based method is complicated for processing webshells that use anti-detection techniques. In addition, detecting long sequence files (such as complex dynamic encryption and decryption scripts or large files) is quite challenging. Methods such as sample slicing [3] or TextRank [5] are often used to reduce data size, which may result in some loss of code information or disruption of contextual relationships.

This study focuses on the PHP language because PHP is used by 75.1% of all the websites whose server-side programming language [7]. To address the challenges, this study contribution includes (1) collating a new high-quality Webshell dataset, (2) proposing a PHP code data processing method to extract Opcode Double-Tuples(ODTs) including opcode instructions and operands instead of Opcode Single-Tuples(OSTs), (3) introducing a window attention mechanism to solve the long text problem.

II. METHODOLOGY

The detection method consists of two steps. First, the PHP source code in the dataset is processed into ODTs. Second, using a sliding window attention mechanism, we combine the CodeBert model [8] and the Fasttext model [9] for feature representation and binary classification of the ODTs. Our dataset and processing code are publicly available: https://github.com/w-32768/PHP-Webshell-Detection-via-Opcode-Analysis

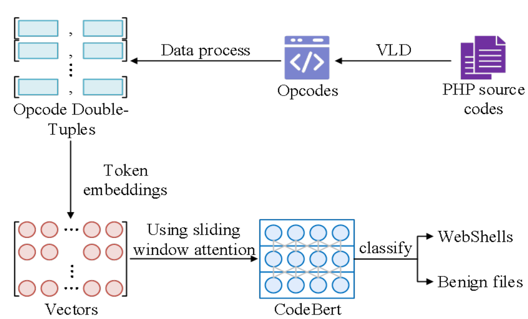


Fig. 1. Overview of the detection method.

A. Data processing

The dataset consists of PHP source code files containing 5001 webshell samples and 5936 benign PHP files. Firstly, we convert the PHP source code to the opcode. The opcode, generated by the Zend Engine in PHP, is a low-level abstraction of source code. As anti-detection techniques are mostly used at the source code level, we have a natural advantage in using opcode detection.

After obtaining the opcodes, a series of data processing steps are performed. We use expert knowledge to establish fine-grained processing rules, extracting high-value instructions for detection while excluding those of low relevance, thus reducing opcode length without compromising contextual semantics. Operands may be encoded by URL or Base64 encoding, making it difficult to determine their semantics. Therefore, we perform the decoding operation. The original string content is restored based on string feature recognition. After this extraction, we have the set of opcode instructions and operands, called Opcode Double-Tuples. Experimental comparisons show that, under the same detection model training on our dataset, ODTs achieve a 4.6% accuracy improvement compared to OSTs, confirming that our data processing method is advanced and professional.

B. Feature Representation and Binary Classification

After data processing, this study explores using the CodeBert model and various embedding models for feature representation and binary classification of ODTs. The steps are as follows:

1) Feature Representation.

- **CodeBert Model:** The CodeBert Model is a widely used pre-trained language model optimized for code understanding tasks and pre-trained on PHP code. We input the ODTs into the CodeBert model to generate high-dimensional feature vector representations that capture the semantic and syntactic information of the opcodes.
- **Embedding Models:** To enhance opcode feature representation, we compared four embedding models: Word2Vec, FastText, Glove, and Doc2Vec. Experimental comparisons show that FastText performs best in the opcode classification task; therefore, we chose FastText as the embedding model.
- **Feature Fusion:** We fuse the feature vectors generated by CodeBert with the embedding vectors from FastText to form the final feature representation. The specific fusion formula is as follows:

$$E = \lambda E_{\text{CodeBert}} + (1 - \lambda) E_{\text{FastText}} \quad (1)$$

E_{CodeBert} and E_{FastText} represent the feature vectors generated by CodeBert and FastText, respectively. λ is the weight coefficient, and its optimal value is determined through experimentation.

2) Sliding Window Attention Mechanism:

We introduce a sliding window attention mechanism to address the high computational complexity of global self-attention mechanisms for long opcode sequences. The opcode sequence is divided into multiple windows of size W with a stride of $Sr(Sr < W)$. Specifically, Self-attention is calculated independently within each window. The global feature representation is obtained by averaging the last hidden states from the CodeBert encoder across all windows. This mechanism reduces memory requirements and allows longer sequences to be processed. Furthermore, the overlap between adjacent windows allows information exchange, making it possible to detect malicious behaviors.

The sliding window attention mechanism reduces computational complexity and preserves the contextual information of the opcode sequence. Thus, the problem of incomplete information caused by other methods is avoided.

3) Binary Classification:

After getting the global feature representation of the ODTs, we input them into a binary classifier. The classifier consists of fully connected layers and activation functions, trained by minimizing the binary cross-entropy loss function. It distinguishes between benign PHP code and malicious webshells.

4) Model Training and Evaluation:

We fine-tuned the CodeBert model using the AdamW optimizer. Experimental results show that our proposed optimal model achieves an accuracy of 99.2% and an F1 score of 99.1% on the test set. Comparative experiments with accessible state-of-the-art webshell detection methods, including webshellPub [2] (Acc: 77.3%, F1: 68.5%), PHP Malware Finder [1] (Acc:83.4%, F1:78.9%), and MSDetector [3] (Acc:97.1%, F1: 97.3%), demonstrate the superiority of our method.

III. CONCLUSION

This study presents a PHP webshell data processing method that extracts ODTs, addressing the limitations of single-tuples detection. Additionally, we introduce a sliding window attention mechanism that effectively mitigates the challenges of long text detection. This study offers a new perspective on the field of malicious code detection. In the future, we aim to continually explore multi-language webshell detection tasks to improve detection performance and generalization capabilities.

ACKNOWLEDGMENT

This work was supported by “the Fundamental Research Funds for the Central Universities” (Grant Number:3282024050).

REFERENCES

[1] NBS System, “PHP malware finder,” 2022. [Online]. Available: https://github.com/nbs-system/php-malware-finder.

[2] ShellPub, “PHP webshell detection,” 2024. [Online]. Available: https://n.shellpub.com/en.

[3] B. Cheng, Y. Guo, Y. Ren, G. Yang, and G. Xu, “MSDetector: a static PHP webshell detection system based on deep learning,” in *Theoretical Aspects of Software Engineering*, vol. 13299, 2022, pp. 155–172.

[4] A. Hannousse, M. Nait-Hamoud, and S. Yahiouche, “A deep learner model for multi-language webshell detection,” *International Journal of Information Security*, vol. 22, no. 1, pp. 47–61, 2023.

[5] T. An, X. Shui, and H. Gao, “Deep learning based webshell detection coping with long text and lexical ambiguity,” in *Information And Communications Security*, 2022, pp. 438–457.

[6] M. Ma, L. Han, and C. Zhou, “Research and application of artificial intelligence based webshell detection model: a literature review,” *ArXiv*, vol. 2405.00066, 2024.

[7] W3Techs, “Usage statistics and market share of PHP for websites,” 2025. [Online]. Available: https://w3techs.com/technologies/details/pl-php.

[8] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong et al., “Codebert: a pre-trained model for programming and natural languages,” *ArXiv*, vol. 2002.08155, 2020.

[9] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” *ArXiv*, vol. 1607.01759, 2016.

Poster: Long PHP webshell files detection based on sliding window attention

Zhiqiang Wang[1✉], Haoyu Wang[1✉], Lu Hao[2]
[1]Beijing Electronic Science & Technology Institute, [2]Beijing Municipal Public Security Bureau, Beijing, China
Email: wangzq@besti.edu.cn, 20232909@mail.besti.edu.cn

Problem and Motivation

Background

Webshell is a malicious code that threatens web application security.

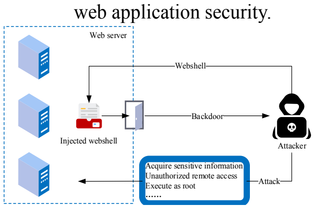
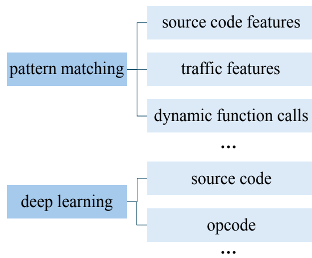


Fig 1.Webshell Threats to Web Application Services

Existing webshell detection methods



Key Points and Current Challenges

1. **Evasion of Pattern Matching**: Attackers can easily bypass traditional pattern-matching methods.
2. **Difficulty with Long Texts**: Deep learning approaches struggle to process long sequences effectively.
3. **Complex Source Code Processing**: Processing source code is complicated due to multiple anti-detection techniques.
4. **Inefficient Opcode Feature Extraction**: Opcode-based methods fail to capture low-level code features adequately.

Long Webshell Files Detection Method

Detection steps

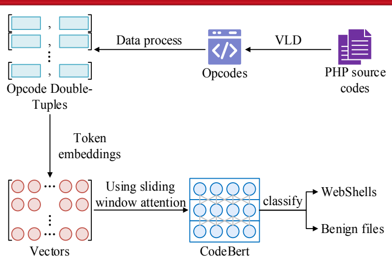


Fig 2. Overview of the detection method

Our method consists of two steps:

1)Source code to Opcode Double-Tuples
2)Feature representation and binary classification

Dataset and processing code are available: https://github.com/w-32768/PHP-Webshell-Detection-via-Opcode-Analysis

Token embedings formula:

$$E = \lambda E_{CodeBert} + (1 - \lambda) E_{FastText}$$

Sliding Window Attention

A. Self-attention:

Calculate independently within each window.

B. Global feature:

Averaging the last hidden states from the CodeBert encoder across all windows.

Windows size :W

Stride: Sr (W > Sr)

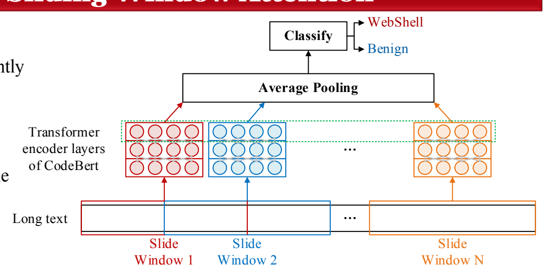


Fig 7. Sliding Window Attention Mechanism

Data process : PHP Source code to Opcode Double-Tuples

Source code ⟶ Raw opcode ⟶ Opcode Double-Tuples. We use ODTs for detection.
You can intuitively perceive the advancement of ODTs by comparing the following images. Four images represent the same file.
(Long strings have been simplified)

```php
<?php
function date_custom($data)
{return base64_decode($data);}
$krl="aHR0cHM6Ly9yYXcuZ210aHV...";
$dt=date_custom($krl);
$ia=file_get_contents($dt);
eval("?>" . $ia);
?>
```

Fig 3. Source code

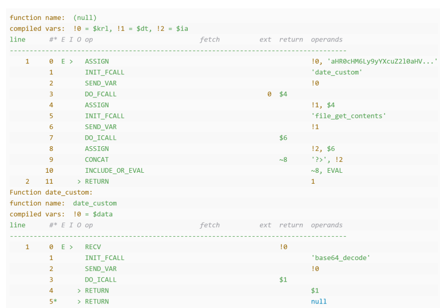


Fig 4. Raw opcode

```
FunctionStart
ASSIGN              'https://raw.githubusercontent.com/...'
INIT_FCALL          'date_custom'
DO_FCALL
INIT_FCALL          'file_get_contents'
DO_ICALL
CONCAT              '?>'
INCLUDE_OR_EVAL     'EVAL'
RETURN 1
FunctionEnd
FunctionStart
INIT_FCALL          'base64_decode'
DO_ICALL
RETURN
RETURN              'null'
FunctionEnd
```

Fig 5. Opcode Double-Tuples*

```
ASSIGN
INIT_FCALL
DO_FCALL
INIT_FCALL
DO_ICALL
CONCAT
INCLUDE_OR_EVAL
RETURN
INIT_FCALL
DO_ICALL
RETURN
RETURN
```

Fig 6. Opcode Single-Tuples

Embedding model

We used four different embedding models to perform feature fusion with CodeBERT embeddings. The comparative experimental results are as follows.
Training set: validation set: testing set = 8:1:1
Embedings formula: $E = \lambda E_{CodeBert} + (1 - \lambda) E_{EmbModel}$

Methods	Accuracy	Precision	Recall	F1 Score
Glove	98.0%	98.7%	96.7%	97.8%
Doc2Vec	98.8%	99.2%	98.3%	98.6%
Word2Vec	98.9%	99.4%	98.1%	98.8%
FastText	**99.2%**	**99.2%**	**99.0%**	**99.1%**

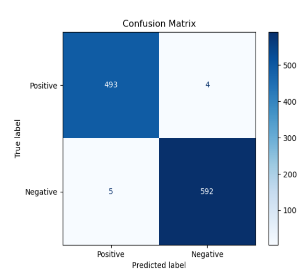
Webshells : positive samples.

Benign files : negative samples.

Experimental result

Comparative experiments with state-of-the-art webshell detection methods, including webshellPub [2] , PHP Malware Finder [1] and MSDetector [3].
Our detection method was also compared with the Opcode Single-Tuples(OSTs) detection method, demonstrating the superiority of our use of Opcode Double-Tuples(ODTs).

Methods	Accuracy	Precision	Recall	F1 Score
ShellPub	77.3%	96.4%	53.1%	68.5%
PMF	83.4%	96.4%	66.8%	78.9%
MSDetector	97.2%	97.6%	97.1%	97.3%
Our Method	**99.2%**	**99.2%**	**99.0%**	**99.1%**
OSTs	94.6%	97.2%	92.4%	94.7%



REFERENCES

[1] NBS System, "PHP malware finder," 2022. [Online]. Available: https://github.com/nbs-system/php-malware-finder.
[2] ShellPub, "PHP webshell detection," 2024. [Online]. Available: https://n.shellpub.com/en.
[3] B. Cheng, Y. Guo, Y. Ren, G. Yang, and G. Xu, "MSDetector: a static PHP webshell detection system based on deep learning," in *Theoretical Aspects of Software Engineering*, vol. 13299, 2022, pp. 155–172.