EasyChair Preprint

№ 15426

ASAF: AI-Powered Static Analysis Framework for Webshell Detection

Ha V. Le, Hieu T. Hoang, On V. Phung and Hoa N. Nguyen

November 14, 2024

ASAF: AI-powered Static Analysis Framework for Webshell Detection

Ha V. Le
Office of the Government
Hanoi, Vietnam
levietha@chinhphu.vn

Hieu T. Hoang
FPT Software
Hanoi, Vietnam
hieuht24@fpt.com

On V. Phung
Office of the Government
Hanoi, Vietnam
onphungvan@gmail.com

Hoa N. Nguyen
VNU University of Engineering and Technology, Hanoi, Vietnam
hoa.nguyen@vnu.edu.vn

***Abstract*—The increasing sophistication and prevalence of webshells present a significant threat to web application security, necessitating the development of more advanced detection methods. This study introduces an AI-powered Static Analysis Framework (ASAF) designed to detect both known and novel webshell variants with high accuracy and efficiency. ASAF combines the pattern-matching capabilities of Yara rules for identifying known webshells with the advanced detection power of Convolutional Neural Networks (CNNs) for uncovering new and obfuscated threats. The framework consists of five core components: (1) Yara, which employs textual and binary pattern matching to detect known webshells; (2) Opcode Vectorization, which translates web source code into opcode sequences for deeper analysis; (3) Dataset Collecting and Cleaning, which ensures the framework is trained on high-quality data; (4) CNN Model, designed to capture intricate patterns in opcode sequences. Through the integration of static signature-based and CNN-based methods, ASAF provides a comprehensive and robust solution for webshell detection.**

***Index Terms*—Webshell Detection, Convolutional Neural Networks, Static Analysis**

I. Introduction

Webshells present a significant threat to web applications by granting unauthorized access to attackers, leading to severe security breaches [1]. Traditional detection methods, which rely on signature-based techniques, are effective for known threats but fail to detect new or modified webshells. These methods also struggle with high false-positive rates, leading to alert fatigue and decreased detection efficiency [2]. Machine learning (ML) and deep learning (DL) methods can learn and adapt to complex patterns in code, improving detection of novel webshell variants [3]. Techniques such as multi-classifier ensembles, feature vectorization, and deep learning models have shown high accuracy in distinguishing between benign and malicious code [4] [5]. This research introduces a hybrid detection architecture that integrates signature-based techniques with AI algorithms to enhance detection speed and resource efficiency, effectively identifying both known threats and novel webshells in PHP to bolster web application security.

In this paper, we present an innovative AI-powered Static Analysis Framework (ASAF) designed to enhance webshell detection by integrating traditional static analysis with advanced machine learning techniques. Our work makes two main contributions:

- ASAF Framework: Combines signature-based algorithms and machine learning/deep learning (ML/DL) techniques for detecting both known and unknown webshells, adaptable to multiple programming languages.
- PHP Webshell Detection Model: Converts PHP source files into feature vectors using an algorithm, then applies an optimized ML/DL model to detect webshells efficiently, with low computational resource usage.

The remaining parts include: Section §II describes related works. §III describes in detail AI-powered Static Analysis Framework for Webshell Detection method. In §IV, we describe the implementation process and results. Finally, §V summaries and future directions.

II. Related Works

Signature-based webshell detection uses code patterns or sequences called signatures. Yara [6], an open-source pattern matching program is widely used to identify threats through byte sequences, string patterns, and regular expressions, but its main drawback is the potential to miss new webshells, leading to false negatives.

ML is being used by more webshell detection researchers to bypass signatures. ML models learn from data and find new patterns to detect webshells. A proposed ML model categorizes files as safe or risky using static and dynamic web application source code data. Method found signature-avoiding complex webshells. Webshell detection is possible using a CNN-based framework [7], which analyzes web application code commands. CNN outperforms machine learning at finding hidden webshells. The training of DL models requires massive datasets and processing resources, which may limit their practical use.

Hybrid signature-ML enhances webshell detection by using DL models to identify new or obfuscated webshells after Yara rules filter them, while combining Yara rules with gradient boosting machines effectively detects both known and unknown webshells [8].

CNNs can identify unfamiliar webshells using feature extraction and complex pattern learning from raw data. CNNs, unlike SVMs or Random Forests, can automatically extract hierarchical features from web source code, detecting subtle patterns that may indicate malicious behavior [9] CNNs identify dangerous sequences across coding styles and obfuscations

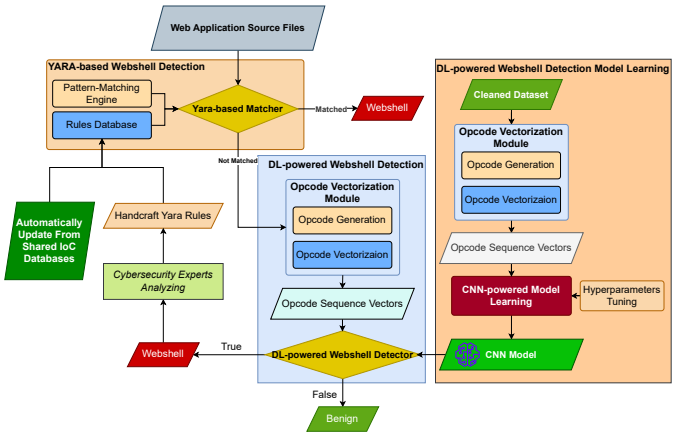

Fig. 1. Overall Architecture of AI-powered Static Analysis Framework

with spatial invariance and local connectedness, capturing complex structures and providing scalable, real-time detection for large-scale code analysis.

III. Proposed Method

A. Approach Direction

The increasing complexity and prevalence of webshells require a unified static analysis framework that works across multiple programming languages for quick detection with low false positives, while accurately identifying new variants. This study introduces the **AI-powered Static Analysis Framework**, shortly called **ASAF**, which integrates Yara rules for detecting known webshells and a CNN model for identifying more sophisticated variants, combining the strengths of signature-based and machine learning methods for a comprehensive detection solution.

B. ASAF Workflow

Fig. 1 presents a comprehensive webshell detection framework using static analysis and machine learning. Yara scans source files for known patterns, flagging matches as webshells, while unmatched files are converted into opcode sequences and analyzed by a CNN model, with detections verified by experts to ensure robust identification of both known and unknown webshells.

C. Yara Analysis

The Yara component in ASAF detects known webshells using a pattern matching engine and Yara-rules database, utilizing text and binary matching techniques, including wildcards and regular expressions, along with Boolean logic for complex detections. The process is illustrated in Algorithm 1.

The Yara-Rules database, regularly updated from the IOC database or by experts to stay current with emerging threats, identifies known webshells by matching specific textual and binary patterns, and in the ASAF architecture, the Yara module scans source code files, flagging those

Algorithm 1 Pattern-Matching Algorithm

Input: S - Collection of web application source files; RS - collection of Yara-Rules
Output: WS - list of webshells

$WS \leftarrow \emptyset$
for each $f \in S$ **do**
  for each $r \in RS$ **do**
    if $PatternMatch(r, f) = true$ **then** ▷ if file f match with rule r
      $WS.append(f)$ ▷ f should be appended to WS as a webshell
    end if
  end for
end for
return WS

D. Opcode Vectorization

The module enhances webshell detection by converting web source code into opcode vectors, which enables ML/DL models to identify concealed malicious patterns and detect advanced evasion and obfuscation techniques [10]. The module has two main components: Opcode Generation, which converts source code into low-level opcodes using tools like VLD or ILDasm, and Opcode Vectorization, which transforms these opcodes into numerical formats for machine learning models, using methods like one-hot encoding or Word2Vec. The Opcode Index Vectorization Algorithm (OIVA) enhances this process by creating indexed vectors based on the order of instructions, enabling CNN models to effectively detect webshells by capturing essential features of the source code, shown in Algorithm 2.

Algorithm 2 OIVA: Opcode Index Vectorization Algorithm

Input: of - Opcode file generated by VLD/ILDasm; IS - Indexed opcodes set of a programming language
Output: OCI - vector containing the opcode indexes of of.

$OCI \leftarrow \emptyset$
for each $line \in of$ **do**
  for each $opcode \in IS$ **do**
    if $opcode \in line$ **then**
      $OCI.append(getIndex(opcode, IS))$ ▷ Add the index of $opcode$ in IS to vector OCI
    end if
  end for
end for
return OCI

E. Dataset Collecting and Cleaning

In the ASAF framework, the dataset for training and testing the CNN model for webshell detection includes benign files from repositories like GitHub and malicious webshells from malware repositories, honeypots, and security forums, with confirmed webshells added and the dataset preprocessed for high-quality model training.

TABLE I
CLEAN BENIGN AND WEBSHELL DATASETS

	Training Set	Testing Set
Benign Dataset	5,820	1,455
Webshell Dataset	3,270	817

F. CNN Model Architecture

In the proposed ASAF, a CNN detects webshells by processing vectorized opcode sequences through an Input Layer, extracting features with Convolutional Layers, reducing complexity via Global Max-Pooling, and outputting probabilities through a Classification Layer with Softmax activation. The Adam optimizer is employed to fine-tune the model for efficient convergence, with parameters such as a 0.05 learning rate, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$ [11] in place. This architecture is designed to effectively analyze and classify opcode sequences, providing a reliable approach to the detection of sophisticated webshells [12].

IV. EXPERIMENTS & EVALUATION

To demonstrate ASAF's effectiveness, we conduct in-depth experiments to address key research questions:

- RQ1: How is ASAF used to detect PHP webshells?
- RQ2: How does ASAF improve webshell detection performance compared to other static analysis methods?
- RQ3: Does deep learning allow for improved webshell detection performance compared to some other state of the art (SOTA) machine learning methods?

A. Dataset Preparation

The benign dataset has been collected from the official websites of various PHP frameworks, forums, and content management systems. They consist of Laravel, Wordpress, Joomla, phpMyAdmin, phpPgAdmin, and phpbb[1]. Thus, the benign set comprises a total of **7,400** files after the removal of non-PHP files. In order to construct the webshell dataset, we gathered a diverse selection of webshells from the most reputable and highly rated sources on Github[2]. Furthermore, we implemented several webshell samples that we acquired during our employment. Consequently, there are a total of 4,171 webshell files.

Upon Yara's and expert input's review of the collected webshell dataset, we eliminated 27 false positives, resulting in **7,275** benign files and **4,087**webshell files. In order to train and validate our PHP webshell detection method, we adhered to industry standards by dividing the dataset into training and testing sets at an 8:2 ratio [13]. Table I illustrates the ultimate distribution of files between the non-duplicate training and testing sets.

[1]Benign repos on Github: Laravel; WordPress; Joomla; phpmyadmin; phppgadmin; phpbb.

[2]Webshell repo on Github: Tennc, PHP-backdoors, B374k, php-webshells, xl7dev/WebShell, BlackArch/webshells, fuzzdb, webshell-collector, webshell-sample, awsome-webshell, WebShell-Bypass-WAF, indoxploit-shell.

B. Implementation Details

We constructed and implemented our solution, **ASAF**, in the Python language, based on the proposed method. The experiments were conducted on a computer that was equipped with 2 x Intel Xeon E5-2697 v4, 128 GB of RAM, CentOS Linux 7-9.2009, and Python release 3.8. We employ tensorflow v.1.14.0, scikit-learn v.0.20.4, scipy v.1.2.2, numpy v.1.16.5, and yara-python v.3.10.0 for the deep learning platform.

The experimental phase is conducted using the test dataset constructed in §IV-A, in conjunction with three scenarios.

- S1: Evaluate the detection efficiency of the Yara component for PHP webshells in ASAF.
- S2: Assess the detection efficiency of the CNN model for PHP webshells in ASAF.
- S3: Evaluate the aggregate detection efficiency of all ASAF components for PHP webshells.

C. Results and Evaluation

1) S1: Yara-based Webshell Detection: In our work, the Yara-Rules dataset contains*699 rules* collected from sources like GitHub, GitLab, and professional contributions, enabling the detection of webshells in languages such as PHP, ASP.NET,JSP, and Python. The system achieved high performance in testing, with 94.89% accuracy, 98.88% precision, 86.76% recall, and 99.45% specificity. However, the recall rate shows that 13.24% of webshells go undetected, highlighting the system's limitation in detecting new or obfuscated variants. To improve adaptability and detection of emerging threats, integrating machine learning techniques is recommended.

2) S2: CNN-based Webshell Detection: The dataset described in §IV-A is also utilized to evaluate the CNN model using the tensorflow engine, as in the previous experiment.

The most recent assessment of the CNN-based detection system demonstrates its remarkable accuracy of 98.81% in the identification of recognized webshell patterns. The model achieves a high precision rate of 97.94%, effectively minimizing false positives and allowing security analysts to focus on genuine threats. With a recall rate of 98.78%, the system robustly detects nearly all actual webshells, while its specificity of 98.83% ensures accurate identification of benign files. The F1-score of 98.35% demonstrates a strong balance between precision and recall, further validating the model's effectiveness. Furthermore, the system's reliability and accuracy in detecting webshells are substantiated by its low false positive rate of 1.17% and false negative rate of 1.22%.

3) S3: ASAF-based Web Detection: From the above two experiments, we have shown the advantages and disadvantages of Yara and CNN techniques in detecting PHP webshells. In this section, we will continue to experiment with the ASAF framework on the same dataset to prove its effectiveness over the above two techniques in detecting PHP webshell attacks. After conducting the three methods, we obtained the experimental results of the methods shown in the confusion matrix in Table II and III.

TABLE II
CONFUSION MATRIX OF PHP WEBSHELL DETECTION

		Real Webshell	Real Benign
S1-Yara	Predicted Webshell	709	8
	Predicted Benign	108	1447
S2-CNN	Predicted Webshell	807	17
	Predicted Benign	10	1438
S3-ASAF	Predicted Webshell	809	17
	Predicted Benign	8	1438

TABLE III
KEY METRICS OF PHP WEBSHELL DETECTION (%)

Measure	S1-Yara(%)	S2-CNN(%)	S3-ASAF(%)
F1-Score	92.43	98.35	98.48
Specificity	99.45	98.83	98.83
False Positive Rate	0.55	1.17	1.17
False Negative Rate	13.24	1.22	0.98
Accuracy	94.89	98.81	98.9
Precision	98.88	97.94	97.94
Recall	86.76	98.78	99.02

ASAF effectively detects both known and unknown webshell patterns by integrating Yara's pattern matching with CNN's deep learning, outperforming both in all metrics; for instance, it recorded 8 false negatives (FNs) among 817 PHP webshell files, compared to CNN's 10, as 2 of those FNs were correctly identified by Yara.

4) Evaluation: To justify our ASAF's performance, we compare our results to those of other approaches. For comparison, we chose two non-AI approaches [14], [15] and two ML/DL approaches [16], [17]. Due to the limitations of sharing source code, we have simulated the RF-GBDT and Word2Vec+CNN model as described by the authors, which are currently one of the best archivements for evaluation on our dataset aforementioned in §IV-A. The actual results of the simulated RF-GBDT and Word2Vec+CNN are not as high as announced. The comparison results in Table IV show that the ASAF model achieves the best results in both accuracy of 98.9% and F1-Score of 98.48% when compared with other methods on our dataset.

V. CONCLUSIONS

This research introduces a framework for the detection of webshells that integrates static analysis techniques with deep learning. To provide a more detailed explanation, our ASAF framework employs the pattern recognition capabilities of CNNs, opcode vectorization techniques, and Yara-based matching to precisely identify malicious scripts that are embedded within web applications. Our ASAF investigations with the PHP language have enabled us to verify that the integration of static analysis with AI results in a rapid and efficient solution that surpasses conventional detection methods. Future research may investigate the application of this method for detecting webshells in additional programming languages and the utilization of more sophisticated deep learning models to enhance detection rates across various web environments.

TABLE IV
COMPARISON OF DIFFERENT WEBSHELL DETECTION APPROACHES ON OUR DATASET (%)

Method	Venue	Accuracy	F1-Score
Simulated Word2Vec+CNN [17]	ICNCC, 2017	98.42	97.80
Simulated RF-GBDT [16]	DSC, 2018	98.59	98.05
GuruWS [15]	TCCI, 2019	85.56	92.00
php-malware-finder [14]	NBS, 2022	94.23	96.46
ASAF (our)	VCRIS, 2024	**98.9**	**98.48**

REFERENCES

[1] V.-G. Le, H.-T. Nguyen, D.-P. Pham, V.-O. Phung, and N.-H. Nguyen, *GuruWS: A Hybrid Platform for Detecting Malicious Web Shells and Web Application Vulnerabilities*, 01 2019, pp. 184–208.

[2] H. V. Le, O. V. Phung, and H. N. Nguyen, "Information security risk management by a holistic approach: a case study for vietnamese e-government," *IJCSNS International Journal of Computer Science and Network Security*, vol. 20, no. 6, pp. 72–82, 2020.

[3] Wenjuan-Lian, Qi-FAN, Dandan-Shi, Qili-An, and B. Jia, "Webshell detection based on multi-classifier ensemble model," *Journal of Computers*, vol. 31, no. 1, pp. 242–252, Feb 2020.

[4] X. Zhongzheng and N. Luktarhan, "Webshell detection with byte-level features based on deep learning," *Journal of Intelligent & Fuzzy Systems*, vol. 40, pp. 1585–1596, 01 2021.

[5] A. Pu, X. Feng, Y. Zhang, X. Wan, J. Han, and C. Huang, "Bert-embedding-based jsp webshell detection on bytecode level using xgboost," *Security and Communication Networks*, vol. 2022, no. 1, p. 4315829, 08 2022.

[6] R. H. Mahdi and H. Trabelsi, "Detection of malware by using yara rules," in *2024 21st International Multi-Conference on Systems, Signals & Devices (SSD)*, 2024, pp. 1–8.

[7] H. Luo, X. Ye, X. Xie, and X. Liu, "Design and implementation of an intelligent webshell detection tool," in *2023 3rd International Conference on Digital Society and Intelligent Systems (DSInS)*, 2023, pp. 202–206.

[8] T. Zhu, Z. Weng, L. Fu, and L. Ruan, "A web shell detection method based on multiview feature fusion," *Applied Sciences*, vol. 10, p. 6274, 09 2020.

[9] Y. Wu, M. Song, Y. Li, Y. Tian, E. Tong, W. Niu, B. Jia, H. Huang, Q. Li, and J. Liu, "Improving convolutional neural network-based webshell detection through reinforcement learning," 2021, p. 368–383.

[10] M. Maskur, Z. Sari, and A. S. Miftakh, "Implementation of obfuscation technique on php source code," in *International Conference on Electrical Engineering, Computer Science and Informatics*, 2018, pp. 738–742.

[11] N.-H. Nguyen, V.-H. Le, V.-O. Phung, and P.-H. Du, "Toward a deep learning approach for detecting php webshell," in *International Symposium on Information and Communication Technology*, 2019, p. 514–521.

[12] H. V. Le, T. N. Nguyen, H. N. Nguyen, and L. Le, "An efficient hybrid webshell detection method for webserver of marine transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2630–2642, 2023.

[13] L. V-G, N. H-T, L. L-D, and N.-H. Nguyen, "A solution for automatically malicious web shell and web application vulnerability detection," in *Computational Collective Intelligence*, 2016, pp. 367–378.

[14] "Php malware finder," https://github.com/nbs-system/php-malware-finder, 09 2022.

[15] V.-G. Le, H.-T. Nguyen, D.-P. Pham, V.-O. Phung, and N.-H. Nguyen, "Guruws: A hybrid platform for detecting malicious web shells and web application vulnerabilities," *Transactions on Computational Collective Intelligence XXXII*, pp. 184–208, 2019.

[16] H. Cui, D. Huang, Y. Fang, L. Liu, and C. Huang, "Webshell detection based on random forest–gradient boosting decision tree algorithm," in *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, 2018, pp. 153–160.

[17] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, "Cnn-webshell: Malicious web shell detection with convolutional neural network," in *International Conference on Network, Communication and Computing*, 2017, pp. 75–79.