VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Viet Ha

ENHANCING WEBSHELL DETECTION WITH DEEP LEARNING-POWERED METHODS

Major: Information Systems

Code: 9480104

SUMMARY OF THE PhD DISSERTATION

Supervisors:

Associate Professor Nguyen Ngoc Hoa

Doctor Phung Van On

Ha Noi - 2024

The thesis was completed at: **University of Engineering and Technology, Vietnam National University, Hanoi**.

Supervisor:

- Associate Professor, PhD Nguyen Ngoc Hoa

- PhD Phung Van On

Reviewer: .....

.....

.....

Reviewer: .....

.....

.....

Reviewer: .....

.....

.....

The Dissertation is going to be defended before a National University-level Committee in .....

at ..... hour ..... date ..... month ..... year .....

Thesis can be found at:

- National Library of Vietnam.

- Information Center - Library, Vietnam National University, Hanoi.

Introduction

Motivation

The innovation of web development technology has made web applications more and more popular and is gradually replacing traditional native applications because they do not depend on the operating system. Along with this, the issues of information security for the web system have become increasingly important. Malicious code injection (webshell) attacks are the most common and also the most hazardous sort of web application attack.

According to analysis from Cloudflare, over two-thirds of webshells exhibit some form of obfuscation. Advances in detection techniques have struggled to keep pace as attackers continually release new, heavily obfuscated webshell tools to evade defenses. The increasing sophistication and variety of webshells, particularly those designed to evade traditional detection methods, underscores the urgent need for advanced techniques to improve their detection. Many studies demonstrate the effectiveness of using ML and DL algorithms to improve the ability to detect new variants of webshells. However, these studies still have certain limitations, and there is much room for improvement.

Research Challenges

From the research context analyzed show that webshell detection algorithms that can be applied in practice still exists a number of challenges as follows:

1. **The diversity of webshell languages** underscores the dynamic landscape of cybersecurity threats and defensive measures. Each language offers unique features and capabilities, therefore how to represent the source files so that the functionalities of the webshell can be fully expressed is the issue.

2. **Advanced webshells** often exhibit complex functionalities and evasion techniques, leveraging obfuscation, encryption, and polymorphism to conceal their

presence and evade detection by traditional security measures. As a result, detection requires more advanced techniques, such as behavior-based monitoring, anomaly detection, and runtime analysis of memory activities.

3. **Quality of the Datasets** is one of the key factors in the development of webshell defense techniques. However webshell is a sensitive data, there will not be many official, reliable data sources willing to share.

4. **Accuracy and Detection Speed** are two criteria for any cybersecurity solution. In webshell detection problems, the big challenge is to build a solution while attaining both criteria of accuracy in detection of advanced webshells and detection speed fast enough to minimize damage to the system.

Objectives of Dissertation

The main objective of the dissertation is to propose webshell detection methods that employ the deep learning models in order to improve the performance in term of accuracy and effective. To achieve the main objective of the dissertation, four specific objectives are as follows:

- Objective 1: Overview of webshell, the most advanced techniques used by hackers to hide or evade their webshell attack. Research webshell detection techniques and analyse the advantages and disadvantages of each method. Evaluate the results of the latest research on the problem of detecting webshell attacks.

- Objective 2: Proposing an DL-Powered Source Code Analysis Framework, namely ASAF, that combines signature-based techniques with deep learning algorithms. This hybrid approach enables the rapid and accurate detection of both known and unknown webshell types. The proposed framework provides a guideline for developing specific models tailored to various programming languages.

- Objective 3: Based on the proposed framework above, develop two comprehensive systems tailored to detect webshell attacks using PHP (interpreted language) and ASP.NET (compiled language). The deep learning models integrated into the systems must be optimised for their specific webshell detection problems to ensure effective detection with minimal computational resources. The

detection results of the systems must be compared with those of other studies to prove their effectiveness.

- Objective 4: Proposing a deep learning model for webshell attacks that perform in-depth analysis of HTTP queries directed at web application systems, effectively identifying queries that indicate both known and unknown webshell attacks. The model is capable of seamlessly integrating into NetIDPS, demonstrating its practical applicability for automatic blocking of suspicious webshell attack source addresses in real-time.

Research Contributions

The dissertation has the following contributions:

1. Proposing an DL-powered source code scanning framework for webshell detection that combines signature-based techniques with deep learning algorithms. This framework provides guidance for developing specific models for accurate and efficient webshell detection in a variety of programming languages. For each type of interpreted and compiled programming language, we chose PHP and ASP.NET as the most popular languages of each type to build a webshell detection model based on ASAF. We conducted experiments and compared the above model with other studies to prove the effectiveness of ASAF. *This contribution have been presented in four publications, including one article in the SCI-E/Scopus journal [LVH-J1], one article in an international journal [LVH-J3] (indexed by E-SCI until 2023), one article in the national journal of science and technology on information security [LVH-J4], and one paper at the WoS/Scopus conference [LVH-C1]. Methods for detecting malicious code in web application source code using PHP and ASP.NET have also been registered for patents at the Department of Intellectual Property, Ministry of Science and Technology [LVH-P1, LVH-P2]. In particular, the method for detecting ASP.NET webshells was granted a patent on May 19, 2023.*

2. Propose a deep learning model to thoroughly analyze the HTTP traffic to the web application server in order to quickly detect webshell queries. To solve the problem of data imbalance for training sets, we also propose an algorithm to

improve the quality of training sets employed in the deep learning model. To demonstrate its effectiveness, we experimented with and compared the model to other studies on the same CSE-CIC-IDS2018 dataset. The deep learning model can work with the intrusion detection and prevention system to add attack source IPs to a blacklist and proactively block URI queries to webshell on the web server before they happen. *This contribution have been presented in two publications: one article in the SCI-E/Scopus journal [LVH-J2], one paper at the WoS/Scopus conference [LVH-C2].*

Dissertation Structure

In addition to the introduction and conclusion, this dissertation is organized into 3 main chapters with the following main contents:

- Chapter 1 provides an overview of webshells, methods for detecting webshell attacks, the use of ML/DL in webshell detection, a review of scientific literature, and criteria for evaluating the effectiveness of ML/DL models.

- Chapter 2 proposes an DL-powered source code scanning framework that combines signature-based techniques with deep learning algorithms. This hybrid approach enables the rapid and accurate detection of both known and unknown webshell types. The proposed framework provides a guideline for developing specific models tailored to various programming languages. Based on the proposed framework above, we will develop two comprehensive systems tailored to detect webshell attacks using PHP and ASP.NET.

- Chapter 3 proposes deep learning model for webshell attacks that perform in-depth analysis of HTTP queries directed at web application systems, effectively identifying queries that indicate both known and unknown webshell attacks. We experiment with the model on two datasets and compare the results with those of other studies to showcase its effectiveness. We have also integrated the model into a NetIDPS system to automatically block suspicious source addresses in real-time, ensuring its practical applicability.

1 Background and preliminaries

1.1 Fundamental Concepts

1.1.1 Web Application

The web server utilizes the HyperText Transfer Protocol (HTTP) along with other protocols to receive user requests through a web browser. The server side will need a programming language to handle requests sent from the client side (browser). In compilation, the entire program is translated into machine code by a compiler before execution. In interpretation, the program is translated line by line or statement by statement by an interpreter during execution.

1.1.2 Webshell Attack

A webshell is often a small piece of malicious code injected on web servers by attackers to grant remote access and code execution that is written in popular web development programming languages (e.g., ASP, PHP, JSP).

Basically, a webshell attack is divided into four stages: Finding and Exploiting Vulnerabilities, Persistent Remote Access, Privilege Escalation, Pivoting and Launching Attacks.

1.1.3 Webshell Classification

webshells can have many different classifications based on characteristics, scripting languages, capabilities, etc. Below is the most common classification based on programming language: PHP webshells, ASP/ASPX webshells, Shell Script webshells and Others Server-side Programming Language webshells.

Besides, another classification method is based on the way the webshell communicates with the hacker's control computer.

1.1.4 Webshell Evasion

Hacker employ various techniques to evade detection and enable webshells to bypass security defenses. These evasion tactics manipulate characteristics of the webshell code, communication channels, and execution environment to avoid detection by security systems. One common tactic is obfuscation of the webshell payload source code through methods like encryption, encoding, and polymorphism.

1.1.5 Webshell Feature

Webshells closely resemble benign, which complicates their differentiation. Previous studies have employed three types of metadatas and five set of features to distinguish malicious webshells. Three distinct types of metadata are commonly associated with webshells, each providing unique insights into their characteristics and behaviors: source code, instruction sequence, and HTTP requests.

1.2 Webshell Detection Approaches

1.2.1 Source Code Analysis

For webshell detection, analysis of source code and opcode involves examining the code without executing it, focusing on identifying patterns, structures, and anomalies indicative of webshell presence. Source code analysis entails scrutinizing the textual programming statements comprising the webshell and leveraging syntactic, semantic, and statistical features to distinguish between benign and malicious code. This analysis may involve identifying specific keywords, functions, or patterns commonly associated with webshells, such as system command executions, file manipulations, or network communications.

1.2.2 HTTP Traffics Analysis

HTTP traffics analysis for webshell detection involves scrutinizing the network communication between clients and web servers to identify patterns, anomalies, or signatures indicative of webshell activity. This analysis typically involves capturing and inspecting HTTP requests and responses exchanged between clients and servers,

focusing on various attributes such as request methods, URIs, headers, parameters, payloads, and response codes.

1.3 Related Work

Statistics of research related to ***Webshell Detection*** from reputable sources [1] show that From those 41 studies, 17 of them (42%) adopted machine learning, 12 studies (29%) used deep learning technology and 12 studies (29%) proposed other kinds of solutions.

1.4 Dissertation Research Direction

The dissertation proposes to build two models following two approaches: network-based analysis and source code analysis, to take advantage of each approach, specifically as follows:

- Propose a method of analyzing web application source code to detect webshell. It combines the advantages of two signature-based and CNN-based approaches that are able to detect innovative webshells with very high accuracy. Because each programming language has its own characteristics, within the scope of this study, we will focus on the two most popular server-side programming languages today: interpreted PHP and compiled ASP.NET. In addition, the method also proposes algorithms that represent source files as vector strings, which fully reflect the characteristics of the webshell and are used as input for deep learning models.

- Propose a DNN model to analyze in-depth HTTP queries into the web application system to detect queries that indicate webshell on the web application server. The model is capable of integrating into IDS/IPS systems to automatically add the suspicious source addresses to the blacklist and block the URI of the webshell on the web server.

[1] *IEEE Xplore, *ACM Digital Library, *SpringerLink, *Wiley Online Library, *ScienceDirect

2 DL-Powered webshell detection by source code analysis

2.1 Problem Statement

Typical research works in analyzing application source code to detect webshells have shown that traditional methods and methods using ML/DL both have different advantages and disadvantages, but currently there are not many studies using a combined approach to take advantage of the advantages of these two methods. The dissertation determines the research direction that will focus on proposing an architecture that combines signature-based detection techniques and detection techniques based on AI algorithms. The architecture will allow very fast detection of known webshell and the ability to accurately detect innovation unknown webshell. And in this chapter, the study selects the two most popular server-side programming languages today, PHP and ASP.NET, to prove the correctness and feasibility of the architecture. **Three specific goals are as follows:**

- Proposing an DL-Powered Source Code Analysis Framework, namely ASAF, that mainly combines two techniques, signature-based and ML/DL algorithms, to allow fast and accurate detection of webshell types, including known and unknown types. The framework will be the orientation for building each specific model applicable to each different type of programming language.

- Proposing a complete interpreted language PHP webshell detection model built from ASAF. This model includes an algorithm that converts a PHP source file into a flat vector containing all the webshell features. The model also includes an ML/DL model with parameters tuned to best suit the PHP webshell detection problem, to ensure effective detection without requiring too much computational resources. Evaluating the effectiveness of the proposed model based on measurement criteria and comparing it to relevant studies.

- Proposing a complete compiled-language ASP.NET webshell detection model

built from ASAF. This model includes an algorithm that converts an ASP.NET source file into a flat vector containing all the webshell features. The model also includes an ML/DL model with parameters tuned to best suit the ASP.NET webshell detection problem to ensure effective detection without requiring too much computational resources. Evaluating the effectiveness of the proposed model based on measurement criteria and comparing it to relevant studies.

2.2 Proposed DL-Powered Source Code Analysis Framework

As analyzed above, the increasing sophistication and prevalence of webshells lead to the need for a common source code analysis framework that can be applied to many different programming languages and is capable of fast detection with a low false positive rate for known webshell types. At the same time, it is the ability to detect with high accuracy new types of webshells. Based on previous research results, this study proposes an DL-powered Source Code Analysis Framework, namely ASAF, that combines Yara rules for known webshell detection with a Convolutional Neural Network (CNN) model for detecting new, sophisticated webshell variants. By leveraging the strengths of both signature-based and deep learning-based methods, this framework aims to provide comprehensive and effective webshell detection. The structure of the framework will include five modules/components:

- **YARA Module:** The architecture of the YARA module in ASAF revolves around the YARA system. The main function of this system is to detect known webshells based on predefined patterns. YARA is made up of two components: the pattern-matching mechanism and the Yara-rules database.

- **Opcode Vectorization Module:** The purpose of the module within the webshell detection framework is to enhance the accuracy and depth of source code analysis by converting web source code into its corresponding opcode sequences.

- **Dataset Collecting and Cleaning:** In the ASAF, the dataset plays a critical role in training, validating, and testing the Convolutional Neural Network (CNN) model. The quality, diversity, and size of the dataset directly influence the effectiveness and accuracy of the webshell detection system. The dataset

should include both benign and malicious web application source files to train the CNN effectively. For collecting dataset, multiple data source must be used such as: Open-Source Repositories (GitHub, GitLab, Bitbucket, ...) or Open-Source Frameworks and CMS, ... Public Malware Repositories (VirusTotal, MalShare, TheZoo, Hybrid Analysis, ...) We also can access webshell data source through security forums and open-source repositories, such as: Exploit Database, Hack Forums, GitHub Repositories, ... Finally, personal and professional networks provide access to new types of webshells that are not yet widely shared.

- **CNN Model Architecture:** In a ASAF, CNN model architectural design plays an important role. Architecture of a proposed CNN model is composed of layers, relationships between layers and also hyperparameters whose value is set before the learning process begins. Usually, for each specific problem, there will be certain architectures that show outstanding advantages. However, it needs to go through a process called hyperparameters turning to achieve the best efficiency, performance and speed. Hyperparameters turning consumes quite a lot of time and resources, so not all hyperparameters will be refined when we know those are optimal for the problem. Therefore, at this step, we draft the CNN model architecture using same structure and optimal hyperparameters. The other hyperparameters will be selected after we make the turning at the next step.

- **Hyperparameter Tuning:** The CNN model architecture above is just a basic architectural framework designed to best suit the webshell detection problem, however, the programming language has a great influence on the characteristics of each type of webshell. Therefore, it is very important to perform hyperparameter tuning to build a CNN model for each type of webshell written in different programming languages.

- **ASAF Workflow:** The process begins with web application source files undergoing a pattern-matching analysis using YARA components, which consist of a pattern-matching mechanism and a YARA-rules database. If a match is found, the file is immediately flagged as a webshell. If no match is detected, the file is deemed benign and proceeds to the next stage, which involves deeper analysis using opcode generation and vectorization modules. These modules convert the

source code into opcode sequences, providing a low-level representation of the code's behavior.

The opcode sequences are then vectorized and fed into a Convolutional Neural Network (CNN) for further analysis. The cleaned dataset plays a critical role in training, validating, and testing the CNN model. The model also has been finely tuned through hyperparameter tuning, predicts whether the code is a webshell or benign. If the CNN detects a webshell, this prediction is forwarded to cyber-security experts for verification and rule updating, ensuring that new patterns are incorporated into the YARA-rules database. The framework also allow to automatically update YARA rules from shared IoC databases. Conversely, if the CNN predicts the code as benign, it is confirmed as safe. This dual-layered approach, leveraging both YARA rules for known threats and CNN models for unknown threats, ensures robust and dynamic detection of webshells, enhancing the security of web applications.

2.3 PHP Webshell Detection

2.3.1 YARA

The YARA-rules dataset used in this study is collected from many sources and over a very long period of work in the field of information security. The data set contains a total of *699 rules*, allowing detection of many popular PHP, JSP, ASP, ASP.NET, Python webshells today. This set of rules will continue to be updated regularly to enhance the ability to detect new webshell patterns.

2.3.2 Opcode Vectorization

VLD, short for Vulcan Logic Disassembler, is a powerful PHP extension designed to disassemble compiled PHP code, providing a detailed representation of its internal opcode.

2.3.3 Dataset Collecting and Cleaning

Following ASAF, in the step of reviewing collected webshell dataset with YARA and reviewed by experts, we eliminated 27 false positives. After removing irrelevant

and duplicate files the total number of benign files is **7275** and the total number of webshell files is **4087**.

2.3.4 Hyperparameter Tuninng CNN Model

The CNN model for detecting PHP webshells is built on the basis of ASAF's CNN model, the detailed parameters of the model are selected through the hyperparameter tuninng process. I choose to use the *grid search* technique that exhaustive search over a manually specified subset of the hyperparameter space. There are six hyperparameters that can be turning and they take two types of values that are range and choice. Especially for *filter size*, since the model uses three layers merged together in convolution layer, each layer receives a different filter size, so its value will be a 3-dimensional vector of the form $[x, x+1, x+2]$.

2.3.5 Experiment

The experimental part is performed with 3 scenarios along with the test dataset built in 2.3.3.

- S1: Evaluate the PHP webshells detection efficiency of YARA component in PHP-ASAF.
- S2: Evaluate the PHP webshells detection efficiency of CNN model in PHP-ASAF.
- S3: Evaluate the PHP webshells detection efficiency of PHP-ASAF.

2.3.6 Evaluation

To justify our ASAF's performance, we compare our results to those of other approaches. For comparison, we chose two non-AI approaches [1, 4] and two ML/DL approaches [2, 3]. 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 Section 2.3.3. The actual results of the simulated RF-GBDT and Word2Vec+CNN are not as high as announced. The comparison results in Table 2.1 show that the

Table 2.1: Comparison of different webshell detection approaches on our dataset (%)

Method	Venue	Accuracy	F1-Score
Simulated Word2Vec+CNN[3]	ICNCC, 2017	98.42	97.80
Simulated RF-GBDT[2]	DSC, 2018	98.59	98.05
GuruWS[4]	TCCI, 2019	85.56	92.00
php-malware-finder[1]	NBS, 2022	94.23	96.46
ASAF (our)	VCRIS, 2024	**98.9**	**98.48**

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.

2.4 ASP.NET Webshell Detection Model

According to W3Techs, ASP.NET is the second most used server-side programming language in the world after PHP. Unlike PHP, which is an interpreted language, ASP.NET is a compiled language, so the mechanism for creating opcodes from the source code of the web application is totally different.

2.4.1 Yara

The yara module in ASP.NET webshell attack detection solution is used in conjunction with the ruleset in the section 2.3.1. The rule set contains a total of 699 webshell patterns. We will regularly update this set of rules to improve our ability to detect new webshell.

2.4.2 Dataset Collecting and Cleaning

The total number of ASP.NET source files we gathered was 2.064 webshells and 3.347 benign files. To train the model and test the effectiveness of the proposed method, the dataset will be divided into two parts with the ratio 8:2 corresponding to the training dataset and the test dataset. The following table shows our final datasets for training and testing.

2.4.3 Opcode Vectorizaion

Firstly, the source code of an ASP.NET application can be represented as MSIL which is able to solves the problems related to code obfuscation. Unlike PHP which interprets every time a web page is requested, ASP.NET compiles dynamic web pages into DLL files that the server can execute quickly and efficiently. Then, the DLL files will continue to be converted to Opcode. There are two commonly used tools: ILDasm (IL Disassembler) and Mono.Cecil. We choose to use Mono.Cecil, which is a powerful library for reading, manipulating, and writing .NET assemblies.

2.4.4 Hyperparameter tunning CNN Model

Similar to the hyper process to build parameter selection for the CNN model of PHP-ASAF, we also perform this process to select optimal parameters for the ASP.NET-ASAF model. There are six hyperparameters that can be tunning and they take two types of values that are range and choice.

2.4.5 Experiment

To evaluate the effectiveness of the ASP.NET-ASAF, we conducted experiments with the following three scenarios:

- S1: Evaluate the ASP.NET webshells detection efficiency of YARA component in ASP.NET-ASAF.
- S2: Evaluate the ASP.NET webshells detection efficiency of CNN model in ASP.NET-ASAF.
- S3: Evaluate the ASP.NET webshells detection efficiency of ASP.NET-ASAF.

S3: Evaluation of ASP.NET-ASAF

Combining the advantages of the two YARA methods and the CNN deep learning model, ASP.NET-ASAF will solve the problem of effectively detecting webshell attacks, including unknown patterns. Experiments give the results of confusion matrix and key measures in Table 2.2 and Table 2.3. The result shows that the F1-score increase from 97.95% to 98.07%, accuracy from 98.43% to 98.52% when compare to

CNN prediction result. With the advantage of being able to detect known webshells with very high accuracy, YARA will minimize the number of webshells that CNN misclassifies as benigns. When combining YARA with CNN model, 1 misclassified webshells will be corrected. Then, the FNR decreased from 1.69% to 1.49%.

Table 2.2: Confusion matrix of webshell detection by using ASP.NET-ASAF

	Real Webshell	**Real Benign**
Predicted Webshell	407	10
Predicted Benign	6	659

Table 2.3: Key metrics of of ASP.NET webshell detection by using CNN (%)

Measure	**Value (%)**
Accuracy	98.52
Precision	97.60
Recall	98.55
Specificity	98.51
F1-Score	98.07
False Positive Rate	1.49
False Negative Rate	1.45

Currently there are not many studies capable of detecting ASP.NET webshell patterns using static analysis techniques, and the number of studies willing to share source code is even more limited. Therefore, comparing results with many studies to ensure objectivity is relatively difficult.

2.5 Summary of Chapter 2

Chapter 2 of the dissertation proposes an DL-Powered Source Code Analysis Framework at 2.2, namely ASAF, to effectively detect malicious code injection attacks into web application source code using known and unknown webshells.

The dissertation is directed toward combining the advantages of two popular detection techniques today, the pattern matching technique using Yara to effectively

detect known webshells and the CNN deep learning model to detect new webshells. The framework includes a total of five components linked together through the ASAF workflow. This framework allows us to build each specific system to effectively detect webshell attacks developed in different languages.

To prove the feasibility of this framework, for each type of interpreted and compiled language, the study has experimented with building systems to detect the two most popular server-side programming languages today, PHP at 2.3 and ASP.NET at 2.4. Experimental results are compared with a number of other research results to demonstrate effectiveness.

The research results in this chapter have been presented in four publications, including one article in the SCI-E/Scopus journal [LVH-J1], one article in an international journal [LVH-J3] (indexed by E-SCI until 2023), one article in the national journal of science and technology on information security [LVH-J4], and one paper at the WoS/Scopus conference [LVH-C1]. Methods for detecting malicious code in web application source code using PHP and ASP.NET have also been registered for patents at the Department of Intellectual Property, Ministry of Science and Technology [LVH-P1, LVH-P2]. In particular, the method for detecting ASP.NET webshells was granted.

3 DL-Powered proactive webshell detection and prevention by HTTP traffic analysis

3.1 Problem Statement

Network Intrusion Detection/Prevention Systems (NetIDPS) are pivotal in identifying and mitigating webshell attacks, which pose significant threats to web server security by providing unauthorized remote access to attackers. This approach is effective in identifying previously documented webshells but requires regular updates to the signature database to remain relevant against emerging threats. Thanks to the development of ML/DL algorithms that allow deep analysis of network traffic

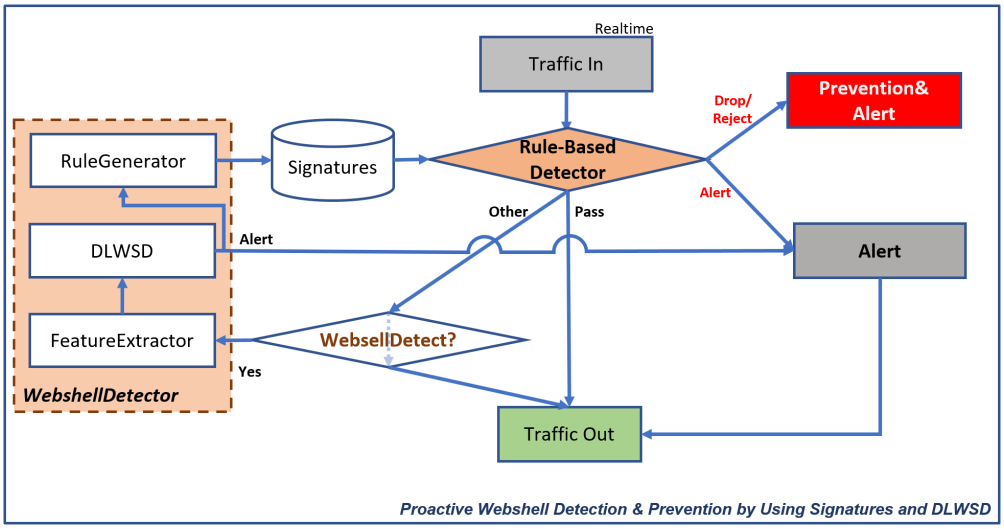
to quickly detect anomalies, it has motivated many studies. However, the previous researches results have shown that there are some major challenges need to be solved. The dissertation determines the research direction that will focus on proposing a comprehensive solution that combines signature-based detection techniques with the deep neural network model to effectively real-time detect and prevent various types of webshell attacks.

Three specific goals are as follows:

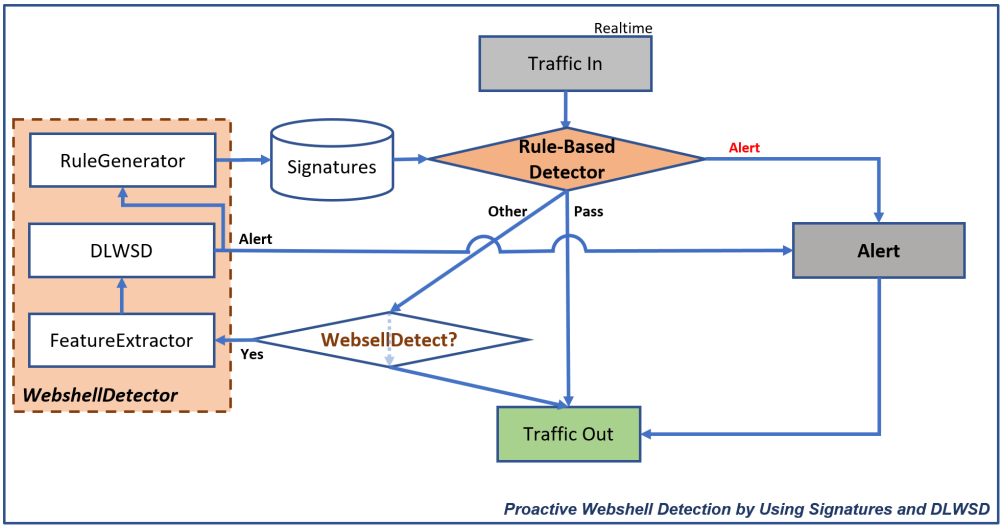
- Proposing a deep neural network (DNN) model allows in-depth analysis of network traffics exchanged with web servers to detect real-time signs of webshell attacks. The DNN model will undergo hyperparameter tuning to optimize performance and accuracy.
- Proposing an improved loss-function algorithm to solve the problem of imbalance in the data set.
- Integrating the DNN model with the NetIDPS solution to enable automatically adding attack source IPs to a blacklist and proactively blocking URI queries to webshell on the web server.

3.2 Proposed Architecture

The intrusion detection method combining rule matching and the deep learning model is illustrated in Fig. 3.1a and Fig. 3.1b. In this method, when the NetIDPS system device is deployed in the inline mode, the network traffic, in both receiving and transmitting directions, is captured by the NetIDPS system. After performing the decoding step, it will proceed with the intrusion detection process based on two detectors. The first detector uses a rule set for intrusion detection. This ruleset is stored in files that have the same rule format as the rules of the Suricata. Each rule will have a pattern or a unique signature that allows the identification of network traffic as under attack. Based on the ruleset, in the inline mode, each network packet will be inspected by the NetIDPS and granted by four actions: (i) drop the packet; (ii) reject the packet (discard the packet and notify the source that sent the packet); (iii) pass and alert; (iv) and pass without warning.



(a) IPS mode



(b) IDS mode

Figure 3.1: Proactive Webshell Detection Method based Signatures and DNN model

3.3 Deep Learning Intrusion Detection Model

In the combined method that we propose in this present invention, we choose to use a deep learning model using DNN. Based on our experimental results, the network intrusion detection with DNN deep learning model using the *tabular_learner* technique of fastAI development framework allows achieving the best accuracy, minimizing the false detection rate when compared to other deep learning models as well as other development frameworks like Keras, TensorFlow, Theano, etc.

3.3.1 Webshell Detection and Prevention

From the trained and stored DLWSD deep learning model, we built the *DeepInspector*, deploying a service running in kernel mode (daemon), to receive network traffic flows saving as PCAP files by NetIDPS. DeepInspector takes the role of monitoring the sampling flows generated frequently $inspection\_frequency$ during $inspection\_duration$ by NetIDPS to analyze and detect the webshell attacks. The process of performing webshell analysis and detection is illustrated in Algorithm **??**. Accordingly, when the NetIDPS system is configured to enable intrusion detection mode that combines both law and machine learning, the network data streams will be received and sent to DeepInspector system software according to multi-process communication mechanism using Unix socket.

3.3.2 Handling Imbalanced Datasets

In fact, when implementing IDPS systems, compromised traffic flows are often very small compared to "benign" network flows. It is also shown in statistics that most of the standard datasets related to network intrusion have a much lower number of attacked flows than benign flows. The imbalance samples between classes cause severely affects both the training process and the classification process in machine learning models, especially for deep learning models. For the binary classification problem, the class having higher samples is called a "majority class"; the other is called a "minority class".

From there, it is necessary to have methods to handle data imbalance between classes. One of the methods of handling data imbalance between classes often used in machine learning models is to assign weights to classes. The samples imbalances are used to create the weights for each class in the training process. These weights are used by the cross-entropy loss function in order to ensure that the majority class is down-weighted accordingly.

3.4 Experiments and evaluation

In order to validate and evaluate the effectiveness of the DLWSD method, we use two datasets:

Table 3.1: Comparison of DLWSD with other methods with DS2

Method	**Accuracy**(%)	**Precision**(%)	**F1-score**(%)	**Recall**(%)
DLWSD	**99.99**	**99.99**	**99.98**	**99.96**
DNN fast.ai	99.92	99.85	99.85	99.85
DSSTE+ miniVGGNet	96.97	97.94	97.04	96.97

- Dataset 1 (DS1): This is the data set that we directly build through the testbed system described above.
- Dataset 2 (DS2): We use a well-known and reliable dataset on CSE-CIC-IDS2018.

3.4.1 Evaluation

From the proposed method, we built the tool to evaluate our method with the cleaned datasets mentioned above. We implement DLWSD based on the FastAI framework. To validate the efficiency of our method DLWSD, three scenarios were built and described as follows:

- S1: Use consecutively DS1 and DS2 to perform the training and testing the DLWSD method without applying the adjustment of the imbalanced dataset.
- S2: Use consecutively DS1 and DS2 to perform the training and testing the DLWSD method within handing the imbalanced dataset.
- S3: Use DS3 to train and test the DLWSD within balancing the classes.

3.4.2 Comparison

To objectively evaluate the effectiveness of the DLWSD method, we experimented and compared the results with the DNN model using fast.ai and the DSSTE+miniVGGNet model on the same CSE-CIC-IDS2018 dataset on 03-02-2018 (DS2). The results in Table 3.1 show that all performance metrics of the DLWSD method are higher than the others methods.

The source code and the dataset used in our experiment can be freely accessed from the GitHub link: https://github.com/levietha0311/DLWSD/.

3.5 Summary of Chapter 3

In this Chapter, we propose the DLWSD method based on the DNN deep learning network model combined with the traditional rule-based detection model. Compared to the baseline results from previous studies, DLWSD also showed superior results on the same experimental dataset. From that result, we have built a specialized DeepInspector suite for detecting Webshell exploit-type intrusion attacks. This detector is integrated into the NetIDSP system according to the Unix socket IPS communication mechanism. NetIDPS is built by us using a SmartNIC network accelerator to be able to analyze and process packets in user space with the DPDK mechanism. The multi-pattern Regex matching method using Hyperscan is also implemented by us in NetIDPS. Actual test results allow us to control, detect and prevent intrusions, especially with tons of Webshell exploits with 4x10Gbps network flows.

Conclusion and future works

Contribution Highlights

The dissertation has made the following specific contributions:

- Proposing an DL-Powered Source Code Analysis Framework (ASAF) that integrates signature-based techniques with deep learning (DL) algorithms. This hybrid approach facilitates the rapid and precise detection of both known and unknown webshell types. The proposed architectural framework serves as a guideline for developing specific models tailored to different programming languages.

- Proposing two ASAF-based comprehensive webshell detection solutions with CNN models tailored for PHP as interpreted and ASP.NET as compiled languages. Each model includes an algorithm that transforms the respective source files into flat vectors, encompassing all webshell features. Furthermore, the models incorporate ML/DL algorithms optimized for their specific webshell detection problems to ensure effective detection with minimal computational resources. The effectiveness of these models will be evaluated based on defined measurement criteria and compared to relevant studies.

- Proposing the DLWSD method to detection and proactively prevent the webshell attacks, which combines a DNN deep learning network model with a traditional rule-based detection model. By correcting the loss function to address the training dataset imbalance, our proposed DLWSD method achieves excellent results on both our generated dataset and the reputable dataset from the Canadian Institute for Cybersecurity. From that result, we developed a specialized DeepInspector suite for detecting webshell exploit-type intrusion attacks. This detector is integrated into the NetIDPS system using the Unix socket IPS communication mechanism.

Dissertation Limitations

Although the dissertation has achieved good research results and made practical contributions as mentioned above, it still has limitations, specifically as follows:

- Most of the current research related to detecting webshell attacks uses autogenerated data sets. This shows that there is actually no webshell data set that is considered standard and widely used in the research community. Out of that general trend, the dissertation is also using self-collected data sets, thus causing many difficulties in objectively comparing the results of other studies.

- The diversity of server-side programming languages leads to a diversity of webshell types. Besides, each type of webshell, according to programming languages, has different characteristics, so different feature extraction methods need to be built. Because of time and resource limitations, the dissertation only chose the two most popular languages today, PHP and ASP.NET, as research and experimental subjects.

- The field of artificial intelligence is currently exploding and constantly making new advances. The introduction of advanced deep learning and machine learning models is continuously being made. Due to time limitations, the dissertation has not been able to research and test the latest current models to apply to the webshell detection problem.

Future Works

Although the dissertation achieve significant results, this field still has a lot of room for expanding research. In the future works, we would like to explore the following research directions as follows:

- Conducting a general survey of webshell datasets used in current research, thereby building a good data set that can be used as a standard for later research related to webshells.
- Continue research and experimentation with the latest single DL/ML models and ensemble models to improve the ability to accurately detect advanced webshells.
- Deeper research into the operating mechanism and characteristics of Webshell will allow the construction of toolkits to automate the Yara rule creation process.
- Expanding research on webshells written in other languages, such as JSP, Ruby, Python, etc., towards building a general model that can effectively detect all types of webshells without depending on the programming language.

List of publications

1. Nguyen, Ngoc Hoa and **Le, Viet Ha** Method for detecting malicious code in Web application source code using ASP.NET language. 1-0036538-000 (27/06/2023).

2. **Le, Viet Ha** and Nguyen, Ngoc Tu and Nguyen, Ngoc Hoa and Le, Linh (2021) An Efficient Hybrid Webshell Detection Method for Webserver of Marine Transportation Systems. IEEE Transactions on Intelligent Transportation Systems. ISSN 1524-9050 (SCI, Scopus-Q1 2021).

3. **Le, Viet Ha** and Du, Phuong Hanh and Nguyen, Ngoc Cuong and Nguyen, Ngoc Hoa and Hoang, Viet Long (2021) A Proactive Method of the Webshell Detection and Prevention based on Deep Traffic Analysis. International Journal of Web and Grid Services. ISSN 1741-1114 (SCI-E, Scopus-Q1 2021);

4. **Le, Viet Ha** and Phung, Van On and Nguyen, Ngoc Hoa (2020) Information Security Risk Management by a Holistic Approach: a Case Study for Vietnamese e-Government. IJCSNS International Journal of Computer Science and Network Security, 20 (6). pp. 72-82. ISSN 1738-7906 (E-SCI 2020);

5. Phung, Van On and **Le, Viet Ha** and Nguyen, Ngoc Hoa, A Solution for Assessing and Managing Information Security Risks in e-Government. Journal of Science and Technology on Information Security, 1(13), pp. 35-48, 2022, ISSN 1859-1256, DOI: 10.54654/isj.v1i13.144.

6. Nguyen, Hoa & **Le, Viet Ha** & Phung, Van-On & Du, Phuong-Hanh. (2019). Toward a Deep Learning Approach for Detecting PHP Webshell. SoICT 2019: Proceedings of the Tenth International Symposium on Information and Communication Technology. 514-521. 10.1145/3368926.3369733;

7. **Ha V. Le**, Hoang V. Vo, Tu N. Nguyen, Hoa N. Nguyen, and Hung T. Du (2022) Towards a Webshell Detection Approach Using Rule-Based and Deep HTTP Traffic Analysis. Computational Collective Intelligence: 14th International Conference, ICCCI 2022. vol 13501 pp. 571–584.