Ordernumber:
Serialnumber:

PEOPLE'SDEMOCRATICREPUBLICOFALGERIA
Ministry of Higher Education and Scientific Research

UNIVERSITY OF ECHAHID HAMMA LAKHDAR - EL OUEDFACULTYOFEXACT SCIENCES
ComputerScienceDepartment

Thesis
Submitted in partial fulfillment of the requirements for the degree of

ACADEMICMASTER

Field: **MathematicsandComputerScience**
Option: **ComputerScience**
Specialty:**Distributed Systems and Artificial Intelligence**

Presentedby:

- **MeriemBensaci**
- **FeryalBrahmi**

Title

Machineanddeeplearningtechniquesfo rdetectionofPHPWebshells

Defended onJune 06\06\2023

ExaminationCommittee:

M.	AbdelkamelBenAli	MCA	Supervisor
M.	Messaoud Abbes	MCA	Examiner
M.	Kholladi nedjoua	MCB	Examiner

Academicyear:2022-2023

Thanks

First of all, we thank Allah, who gave us the strength and the will to carry out this work. We extend our sincere thanks to our supervisor

Dr. Abdelkamel BenAli who was responsible for providing advice and guidance for each small and each large in this memory.

We extend our sincere thanks to all the teachers of the computer science department of Echahid Hamma Lakhdar University, El-Oued.

Also to our colleagues from the 2022-2023 promotion. We also thank all those who participated closely or far to develop this work.

ABSTRACT :

Web shell attacks pose a significant threat to web application and server security. This dissertation focuses on web shell detection and compares the performance of machine learning models: SVM, DNN, KNN, and CNN-LSTM. A dataset of benign and malicious scripts is collected for training and evaluation. Feature extraction techniques are applied to capture relevant characteristics. SVM, a popular binary classification algorithm, is implemented and evaluated. DNN is explored for its potential in web shell detection. KNN, a non-parametric algorithm, is trained and evaluated. CNN-LSTM is investigated to leverage spatial and temporal dependencies. Performance evaluation includes accuracy, precision, recall, F1-score, and ROC analysis. Comparative analysis identifies strengths and weaknesses. SVM achieves high accuracy and precision. DNN shows promise in deep learning. KNN performs reasonably well. CNN-LSTM leverages spatial and temporal features. This research assists in selecting an appropriate model for web shell detection, contributing to cybersecurity efforts.

Key words: Cyberattack, Web shell, Machine learning, Deep learning.

RÉSUMÉ :

Les attaques par Webshell constituent une menace importante pour la sécurité des applications Web et des serveurs. Cette mémoire de master se concentre sur la détection des Webshells et compare les performances des modèles d'apprentissage automatique suivants : SVM, DNN, KNN et CNN-LSTM. Un ensemble de données de scripts bénins et malveillants est collecté pour les tests et l'évaluation. Des techniques d'extraction de caractéristiques sont appliquées afin de capturer les caractéristiques pertinentes. SVM, un algorithme de classification binaire populaire, est mis en œuvre et évalué. DNN est exploré pour son potentiel dans la détection des Webshells. KNN, un algorithme non paramétrique, est entraîné et évalué. CNN-LSTM est étudié afin de tirer parti des dépendances spatiales et temporelles. L'évaluation des performances inclut l'exactitude, la précision, le rappel, le score F1 et l'analyse ROC. L'analyse comparative permet d'identifier les forces et les faiblesses. SVM atteint une précision élevée. DNN présente des perspectives prometteuses dans l'apprentissage en profondeur. KNN fonctionne de manière raisonnable. CNN-LSTM exploite les caractéristiques spatiales et temporelles. Cette recherche contribue à la sélection d'un modèle approprié pour la détection des Webshells, renforçant ainsi les efforts en matière de cybersécurité.

Mots clés : cyberattaque, Webshell, apprentissage automatique, apprentissage en profondeur.

الملخص:

تشكل هجمات قذائف الويب تهديدا كبيرا لتطبيقات الويب وأمن الخادم. تركز هذه الأطروحة على اكتشاف غلاف الويب وتقارن أداء نماذج التعلم الآلي: SVM و DNN و KNN و CNN-LSTM. يتم جمع مجموعة بيانات من البرامج النصية الحميدة والخبيثة للتدريب والتقييم. يتم تطبيق تقنيات استخراج الميزات لالتقاط الخصائص ذات الصلة. يتم تنفيذ SVM ، وهي خوارزمية تصنيف ثنائي شائعة ، وتقييمها. يتم استكشاف DNN لإمكاناته في اكتشاف غلاف الويب. يتم تدريب وتقييم KNN ، وهي خوارزمية غير بارامترية. يتم التحقيق في CNN-LSTM للاستفادة من التبعيات المكانية والزمانية. يشمل تقييم الأداء الدقة والدقة والاستدعاء ودرجة F1 وتحليل ROC. يحدد التحليل المقارن نقاط القوة والضعف. يحقق SVM دقة ودقة عالية. DNN يظهر الوعد في التعلم العميق. KNN يؤدي بشكل جيد إلى حد معقول. تستفيد CNN-LSTM من الميزات المكانية والزمانية. يساعد هذا البحث في اختيار نموذج مناسب لإكتشاف قشرة الويب, مما يساهم في جهود الأمن السيبراني.

الكلمات المفتاحية: ويبشال, الأمن السيبراني, التعلم الآلي, التعلم العميق.

CONTENTS

LIST OF FIGURES

GENERAL INTRODUCTION

The risk of cyber-attacks has grown significantly for both individuals and enterprises due to the Internet is continued expansion and growing reliance on web applications. The security and integrity of web servers and the data they contain are particularly at risk from web shells. These malicious scripts, which are created using scripting languages like PHP, can provide users access to web servers without authorization, enabling attackers to run arbitrary instructions, steal confidential data, or even gain complete control of the system.

Traditional methods for detecting web shells have depended on rule-based or signature-based heuristics, both of which frequently fall behind the constantly changing landscape of web-based attacks. Therefore, there is an increasing demand for more sophisticated and adaptable methods to recognize and lessen the threat posed by web shells quickly.

Deep learning and machine learning have become effective methods for tackling difficult security concerns. Researchers and professionals have created creative methods for more accurately and effectively detecting web shells by utilizing the capabilities of these technologies. These algorithms may learn to differentiate between benign and malicious code, enabling proactive detection and prevention of web-shell assaults. They do this by training models on large datasets that contain both genuine PHP code and diverse instances of web shell scripts.

There are various benefits of using machine learning and deep learning to detect web shells:
- First, by automatically extracting pertinent elements from PHP code; these techniques enable a more thorough and nuanced knowledge of the underlying patterns and traits that set web shells apart from genuine scripts.
- Furthermore, machine learning models are well-suited to deal with the dynamic nature of web shell attacks because they can adapt and advance over time, continuously learning from fresh data and emerging attack methodologies.

- Finally, these models is automation lightens the load on security analysts, giving them more time to concentrate on more strategic and preventative security measures.

In this study, we investigate the use of deep learning and machine learning approaches to identify PHP webshells. We investigate the difficulties in webshell detection and highlight the possibility of cutting-edge models to reduce the dangers brought on by webshell attacks. In order to improve the security posture of web servers and safeguard sensitive data from unwanted access, we intend to contribute to the development of more robust and effective webshell detection systems by studying the advantages and disadvantages of these approaches.

The rest of this dissertation contains:

- Chapter 1:Computer Cyberattacks we introduces cyberattacks and different types of computer attacks. It also delves into the definition of web shells and their work, including aspects such as persistent remote access, privilege escalation, pivoting and launching attacks, and bot herding. The chapter further explores various methods of installing web shells and examines the impacts of web shells on computer systems, including data theft, unauthorized access, damage to system files, increase in network traffic, and the introduction of malicious code. Additionally, it discusses the difficulties associated with identifying web shells and presents webshell detection and prevention methods.

- Chapter 2: Machine and Deep Learning Techniques This chapter introduces machine and deep learning techniques that are relevant to the detection of web shells. It begins with an overview of the topic and then explores specific models such as Naive Bayes (NB), Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Deep Neural Network (DNN), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). For each model, the chapter discusses how it can be used, its main characteristics, advantages, and disavantages. Additionally, the chapter covers the concept of feature extraction in the context of machine and deep learning.

- Chapter 3: Machine and Deep Learning Techniques for Detection of PHP Web Shells. This chapter focuses specifically on the application of machine and deep learning techniques for the detection of PHP web shells. It introduces the topic and outlines the models used in the study. The chapter also describes the development environment, including the hardware and software components. Furthermore, it explains the evaluation scheme, which involves dataset collection, preprocessing, and evaluation metrics. Finally, the chapter concludes with a bibliography listing the relevant sources used in the dissertation.

CHAPTER 1

COMPUTER CYBERATTACKS

1.1 Introduction

The term "cyberattacks" refers to unwanted attempts to steal, expose, alter, disable, or destroy information through gaining unauthorized access to computer systems.

In addition to cybercrime, cyberattacks may also be linked to cyberwarfare or cyberterrorism, such as that carried out by hacktivists. In other words, the incentives can vary. And among these motivations, there are primarily three categories to be found: criminal, political, and personal.

The criminally motivated pirates seek financial gain through the theft of money, the theft of data, or the disruption of operations. The same is true for those driven by personal grievances, such as disgruntled former or current employees, who may seize money, data, or even just the chance to disrupt an organization is system. The majority of their search, though, is for revenge. The goal of pirates with sociopolitical intentions is to draw attention to their causes. They make sure the public is aware of their attacks; as a result, this is sometimes referred to as hacktivism.

The ability test and industrial espionage (in an effort to gain a competitive advantage over rivals) are two other reasons for cyberattacks. Businesses are the target of cyberattacks by criminal organizations, states, and people. Differentiating between external and internal threats is one method of categorizing the dangers of cyberattacks [5]:

• **External dangers :** Among the external cyber risks are: - criminal organizations or groups - Professional hackers, such as actors who are supported by the government. amateur hackers, like hacktivists.

• **Insider dangers :** Those who are legally permitted access to a company is resources and misuse them on purpose or mistakenly are known as insiders. Especially: - Employees that disregard security protocols and rules. - Access to the system by unhappy current or former workers, business. - Partners, clients, subcontractors, or suppliers When businesses, persons acting on behalf of states, or private parties desire to take one or more things, such as: - Details about the company is finances.

- Customer databases.

that contain personally identifiable information (PII).

- Email addresses.

- Login credentials as well as customer lists and financial information, as well as any intellectual property.

- Such as trade secrets or product designs.

- Access to IT resources.
- Including the ability to collect payments.
- Access to personal information.
- The capacity to snoop around in American government departments and agencies.

Cyberattacks can hurt businesses if they are successful. These can result in costly downtime, data alteration or loss, and financial loss due to ransomware. Downtime can also result in significant downtime and financial loss. Similar to: - DoS, DDoS, and malware assaults can bring down servers or cause a system failure.

- Attacks using DNS tunneling or SQL injection have the power to change, remove, insert, or steal data from a system.
- Hackers can access a system to do harm or steal important data via phishing attacks and zero-day vulnerabilities.
- A system may be disabled by ransomware attacks until the company pays the hacker a ransom.

An organization is cybersecurity system can lessen cyberattacks. Protecting sensitive information and important systems from online attacks is the practice of cybersecurity, which involves people, technology, and processes. Using essential cybersecurity technology and industry best practices, an efficient cyber security system prevents, detects, and reports cyberattacks, including:

- Identification and Access Control (IAM).
- A complete platform for data security.
- Event Management and Security data (SIEM).
- Threat intelligence and offensive, defensive, and security services [5].

1.2 Different types of computer attacks

Cyber attacks are malicious attempts to disrupt, damage, or gain unauthorized access to a computer system, network, or device Here are some of the most common types of cyber attacks:

• **Phishing:** A phishing attack attempts to steal sensitive information, such as login credentials or credit card numbers, by posing as a trustworthy entity.

• **Malware:** Malware refers to any malicious software that is designed to harm or disrupt a computer system. Examples of malware include viruses, Trojans, and spyware.

• **Ransomware:** Ransomware is a type of malware that encrypts a victim's files and demands

payment in exchange for the decryption key.

• **DDoS (Distributed Denial of Service):** A DDoS attack overwhelms a target with traffic from multiple sources, rendering it unavailable to users.

• **SQL Injection:** An SQL injection attack involves injecting malicious code into a vulnerable database, allowing the attacker to access sensitive information or make unauthorized changes.

• **Cross-Site Scripting (XSS):** A XSS attack involves injecting malicious scripts into a vulnerable website, allowing the attacker to steal sensitive information or hijack a user is session.

• **Man-in-the-Middle (MitM) Attack:** A MitM attack involves intercepting communication between two parties, allowing the attacker to eavesdrop or manipulate the transmitted data.

• **Botnets:** A botnet is a network of compromised computers that are controlled remotely by an attacker to carry out malicious activities.

• **Password Attack:** A password attack attempts to crack a password using techniques such as brute force, dictionary, or social engineering attacks.

• **Drive-by Download:** A drive-by download attack involves infecting a victim is computer with malware when they visit a compromised website.

• **Spear Phishing:** A spear-phishing attack is a targeted phishing attempt that is specifically targeted at a specific individual or organization.

• **Advanced Persistent Threat (APT):** An APT is a type of cyber attack that is carried out by a highly skilled and well-funded adversary.

• **Watering Hole Attack:** A watering hole attack is a type of attack that involves compromising a website that is frequently visited by the target audience to infect their computers with malware.

• **Wireless Eavesdropping:** Wireless eavesdropping involves intercepting and monitoring wireless communication, such as Wi-Fi or Bluetooth, to steal sensitive information.

• **Social Engineering:** Social engineering is an attack that relies on tricking individuals into revealing sensitive information or performing actions that are harmful to their security.

• **Email Spoofing:** Email spoofing involves forging the sender is address in an email to trick the recipient into believing it is from a trusted source.

• **Cryptojacking:** Cryptojacking involves using someone else is computer resources to mine cryptocurrency without their consent.

• **Denial of Service (DoS):** A DoS attack involves overwhelming a single target with traffic, rendering it unavailable to users.

• **Zero-day exploit:** A zero-day exploit is an attack that takes advantage of a previously

unknown vulnerability in software or hardware.

• **Remote Access Trojan (RAT):** A RAT is a type of malware that allows an attacker to control a victim is computer remotely. These are just a few examples of the different types of cyber-attacks that exist. It is important to stay informed and aware of the latest threats and to take proactive steps to protect your systems and data [6].

1.3 Definition of webshells

A webshell is a malicious script an attacker uses to maintain persistent access to an online application that has already been compromised. A web shell must always be the second phase of an attack because it cannot attack or exploit a remote vulnerability (this stage is also called post-exploitation). To get file upload capabilities and transfer the malicious data, an attacker can exploit well-known web page vulnerabilities like SQL injection, remote file inclusion (RFI), or even use cross-site scripting (XSS). The common features include but are not limited to file management, database enumeration, code execution, and access to the cmd or command line.

Web shells can be created in various web languages; PHP web shells, for instance, are widely used. Whether your system is built using proprietary software or a widely used content management system like WordPress with plugins, they might still impact you. Because web shells do not employ common executable file types, antivirus, and anti-malware tools may also fail to detect them. The public can easily access them at the same time, for instance, through a number of GitHub projects. In this brief series, we will go in-depth on how web shells operate (using a PHP shell as an example) and how to spot web shells and safeguard your assets .

• **how they work:** Web shell attacks have several stages: first, the attacker creates a persistent mechanism on the server enabling remote access. Then, they attempt to escalate privileges and leverage the backdoor to attack the organization or use its resources for criminal activity [1].

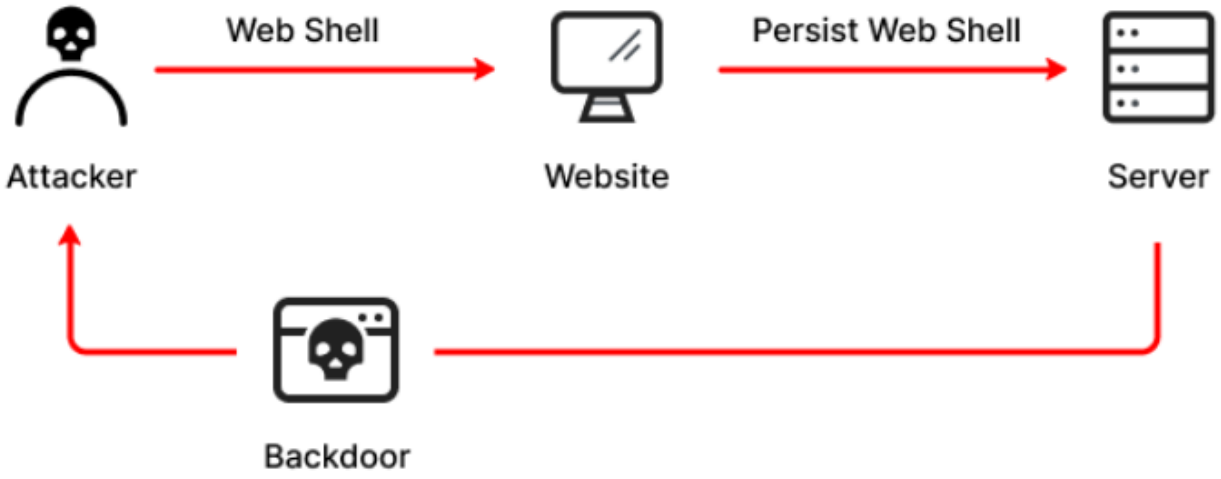

Figure 1.1: Web shall attack stages [1].

1.3.1 Persistent Remote Access

Web shell scripts provide a backdoor allowing attackers to access an exposed server remotely. Persistent attackers dont have to exploit a new vulnerability for each malicious activity. Some attackers even fix the vulnerability they exploit to prevent others from doing the same and avoid detection. Some web shells use techniques such as password authentication to ensure that only specific attackers can access them. Web shells usually obfuscate themselves, including code that prevents search engines from blacklisting the website where the shell is installed [1].

1.3.2 Privilege Escalation

Web shells normally run with user permissions, which can be limited. Attackers can escalate privileges through web shells by exploiting system vulnerabilities to acquire root privileges. Root account access allows attackers to perform almost any actionthey can install software, change permissions, add or remove users, read emails, steal passwords, etc [1].

1.3.3 Pivoting and Launching Attacks

Attackers can use web shells to pivot to additional targets both in and out of the network. The process of sniffing network traffic to identify live hosts, firewalls, or routers (enumeration) can take weeks, during which attackers will keep a low profile to avoid detection.
An attacker that successfully persists on a network will move patiently, possibly even using a compromised system to attack other targets. This allows the attacker to remain anonymous,

and pivoting through several systems can make it virtually impossible to trace attacks to the source [**?**].

1.3.4 Bot Herding

Web shells can be used to connect servers to a botnet (a network of systems controlled by the attacker). The affected servers execute commands sent by attackers through a command and control server connected to the web shell. This is a common technique for DDoS attacks that require extensive bandwidth. Attackers are not directly targeting the system where they have installed the web shell, but are simply exploiting it for its resources to attack more valuable targets [1].

1.4 Different methods of installing Webshells

1. **B**ind Shell

   Bind Shell is a type of shell that is installed on the target device. It gets bound to a certain port on the host and listens for incoming connections to the device. The hacker can then access this web shell remotely and use it to execute scripts on the target host [2].

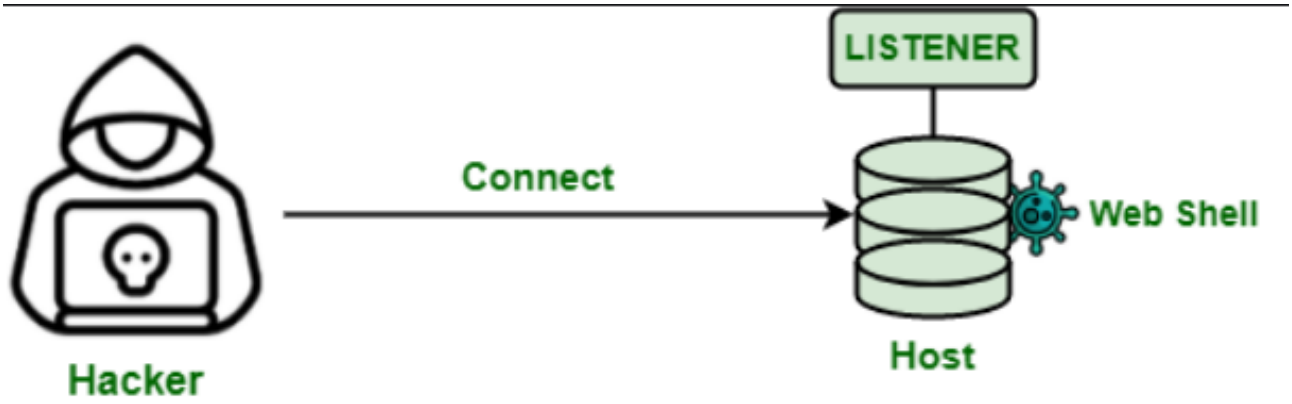

Figure 1.2: Bind shell [2].

2. **Reverse Shell**

   A reverse shell is also known as a Connect-Back Shell. The hackers must look for a remote command execution vulnerability and exploit it for the delivery of the web shell. Unlike

a bind shell, the target host connects back to the hacker is device, which it listens for an incoming connection [2].

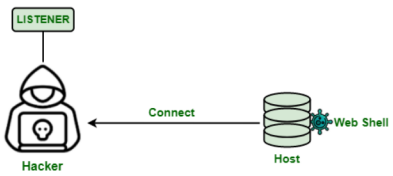

Figure 1.3: Reverse Shell [2].

1.5 Impacts of webshells on computer systems

Web shells are one of the most useful tools for corporate persistence once installed on a server. Web shells are frequently used purely as persistence mechanisms, which is a common occurrence. Web shells are a surefire way for an infected network to have a backdoor since they allow an attacker to leave a malicious implant behind after taking over a server.

Web shells give attackers a chance to keep collecting data from and making money off of the networks they have access to if they go undiscovered. Without identifying and eliminating the attacker persistence techniques, compromise recovery cannot be effective and long-lasting. Furthermore, while reconstructing a single compromised system is an excellent solution, for many, restoring already-existent assets is the only practical choice. Hence, locating and eliminating all A crucial component of compromise recovery is backdoors.

This brings up the issue of web shell detection once more. Web shells can be used more generally to execute arbitrary attacker input via an implant, as we previously described. The first challenge is dealing with how many different methods an attacker can execute code. Online applications offer a high degree of flexibility and compatibility that attackers can exploit because they support various languages and frameworks.

Additionally, because of the amount of network traffic and the constant noise from online attacks, targeted traffic directed at a web server can easily blend in, making the identification of web shells much more difficult and necessitating improved behavior-based detections that can Web shells are malicious scripts that are uploaded to a web server, allowing a remote attacker to control the server and interact with it. These scripts can have a significant negative impact on computer systems, including: [3].

1.5.1 Data theft

Web shells can be used to steal sensitive data, such as passwords or confidential information [3].

1.5.2 Unauthorized access

Web shells can be used to gain unauthorized access to a system, allowing an attacker to take control of the system and perform malicious activities [3].

1.5.3 Damage to system files

Web shells can be used to modify or delete system files, which can lead to system instability or data loss [3]

1.5.4 Increase in network traffic

Web shells can generate a large amount of network traffic, which can overwhelm the system and reduce its performance[3].

1.5.5 Introduce malicious code

Web shells can be used to introduce malicious code into a system, which can be used to infect other systems on the network [3].

1.6 Difficulties with identifying web shells

Any of the common web application languages can be used to create web shells. There are numerous ways to carry out arbitrary commands in each language, as well as numerous ways for arbitrary attacker input. The user agent string or any other parameter that is supplied during a communication between a web server and client can also be used by attackers to conceal instructions [4].
Attackers create a web shell by condensing all these options into a small number of bytes, such as:

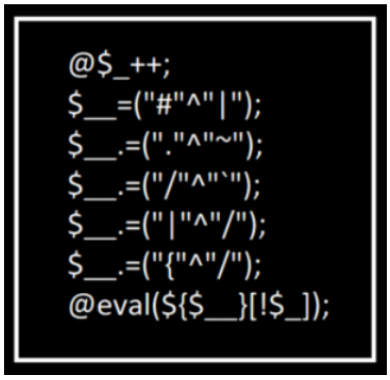

Figure 1.4: screenshot of webshell code [3].

The sole accessible word in the web shell in the aforementioned example is "eval," which is simple to overlook or misunderstand. Using contextual cues is crucial while assessing script. For instance, it would be wise to pay more attention to a scheduled job called "Update Google" that downloads and executes code from a dubious website [4].

With web shells, analyzing context can be a challenge because the context is not clear until the shell is used. In the following code, the most useful clues are system and cat /etc /passwd, but they do not appear until the attacker interacts with the web shell:

```
<?php
  // Adversary sends POST with variable '1' = 'system' and '2' = 'cat /etc/passwd'
  $_=$_POST['1'];
  $__=$_POST['2'];

  //The following will now be equivalent to running -> system('cat /etc/passwd');
  $_($__);
?>
```

Figure 1.5: screenshot of webshell code [4].

Finding intent is a hurdle in web shell detection. Depending on the intention, a script that seems innocent could actually be malevolent. But, if attackers have access to the web directory and can upload any file, they can upload a fully functional web shell that permits arbitrary code execution, as some very basic web shells can.

Because they cannot independently execute commands from an attacker, these file-upload web shells are straightforward, light, and simple to overlook. Instead, they can only post files to web servers, like fully functional web shells.

They are frequently employed by attackers for persistence or the initial stages of exploitation because of their simplicity, which makes them hard to detect and easily mistaken for benign activity [4].

Finally, it is common knowledge that hackers conceal web shells in non-executable file formats, such as media files. Because media files on a web server are checked for server-side execution instructions, web servers configured to run server-side code present extra difficulties in identifying web shells. Attackers can upload photos to web servers that have web shell programs hidden inside them. The image is safe when this file is loaded and examined on a workstation. Yet, harmful code runs server-side when a web browser requests this file from a server[4]. Web shells are becoming more and more common as an attack technique due to these difficulties in detecting them. We keep an eye on how these evasive threats are used in cyberattacks and work to make our defenses stronger. We go through behavior-based detection technologies assistance in defending against web shell assaults in the following section.

1.7 Webshell detection and prevention methods

Web shells are nothing new, and numerous efforts have been made to identify and stop them. It is pretty simple (albeit time-consuming) to just browse the server looking at the upload and modification dates of files relative to the discovery date and manually check suspiciousà-looking uploads to see if they cause the problem once a system breach has been identified. But what if web shells could be identified before being utilized maliciously? [7] There are various methods for achieving it.

As antivirus software does with other types of malware, one method is to have an automated system scan the contents of recently uploaded or modified files to check if they match a recognized web shell.

This is effective when an attacker utilizes a well-known web shell, but it soon fails when dealing with custom code. Another method is to use pattern matching to search for code segments that are frequently harmful (down to the level of individual function calls), such as requests made to the system to alter files or open connections. The issue is that web shell authors are completely aware of this method and purposefully write their code in a way that is opaque and complicated, making pattern matching incredibly challenging to do with any real accuracy. Another method is to search for code segments that are frequently harmful (down to the level of individual function calls) using pattern matching, such as calls to the system[1].

To open connections or handle files. The issue is that web shell authors are completely aware of this method and purposefully write their code in a way that is opaque and complicated, making pattern matching incredibly challenging to do with any real accuracy. The following are some things to look for if an administrator has reason to believe that a web shell

is on their system or is simply performing a normal check. Initially, popular terms used by web shells must be removed from the server access and error logs. Filenames and/or parameter names fall under this category. The example below demonstrates how to search for the string file in URLs in the Apache HTTP Server is access log[1].

```
root@secureserver:/var/www/html# cat /var/log/apache2/access.log | awk -F\" ' { print $1,$2 } ' | grep "file"

--> 192.168.5.26 - - [30/Feb/2020:08:30:53 +0100] GET /demo/shell.php?file=/etc/passwd
```

Figure 1.6: screenshot of webshell code [1].

The filesystem (usually the web server root) must be searched for common strings in files or filenames.

```
root@secureserver:/var/www/html/demo# grep -RPn "(passthru|exec|eval|shell_exec|assert|str_rot13|system|phpinfo|base64_decode|chmod|mkdir|fopen|fclose|readfile) *\("

--> Shell.php:8: eval($string);
eval.php:1:?php system($_SERVER['HTTP_USER_AGENT']); ?>
Ad.php:9: eval($string);
```

Figure 1.7: screenshot of webshell code [1].

Search for very long strings, which may indicate encoding. Some backdoors have thousands of lines of code[1].

```
root@secureserver:/var/www/html/demo# awk 'length($0)>100' *.php

--> eval(gzinflate(base64_decode('HJ3HkqNQEkU/ZzqCBd4t8V4YAQI2E3jvPV8/1Gw6orsVFLyXefMcFUL5EXf/yqceii7e8n9JvOYE9t8
sT8cs//cfWUXldLpKsQ2LCH7EcnuYdrqeqDHEDz+4uJYWH3YLflGUnDJ40DjU/AL1miwEJPpBWlsAxTrgB46jRW/00XpggW00yDI/H1kD7UqxI/3q
jQZ4vz7HLsfNVW1BeQKiVH2VTrXtoiaKYdkT4o/p1E8W/n5eVhagV7GanBn0U7OCfD7zPbCQyO0N/QGtstthqJBia5QJsR6xCgkHpBo1kQMlLt6u+
+SBvtw5KSMwtG4R2yctd0mBNrlB3QQo4aQKGRgRjTa0xYFw1vVM9ySOMd44sSrPe…
```

Figure 1.8: screenshot of webshell code [1].

Search for files that were modified in the recent X days. In the following example, we searched for *.php files changed within the last day but it is recommended to search for any file change as a web shell can also be embedded into an image or any other file.

```
root@secureserver:/var/www/html/# find -name '*.php' -mtime -1 -ls

--> root@secureserver:/var/www/html/# find -name '*.php' -mtime -1 -ls
2885788 4 drwxrwxr-x 2 secuser secuser 4096 Apr 30 06:590 /demo/shell.php
2886629 4 -rw-rw-r-- 1 secuser secuser 260 Apr 29 11:25 /demo/b.php
2897510 4 -rw-r--r-- 1 root root 35 Apr 29 13:46 /demo/source.php
2883635 4 -rw-r--r-- 1 www-data www-data 1332 Apr 29 12:09 ./ma.php
```

Figure 1.9: screenshot of webshell code [1].

Monitor the network for unusual network traffic and connections.

```
root@secureserver:/var/www/html/demo# netstat -nputw

--> Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 192.168.5.25:37040 192.168.5.26:8181 ESTABLISHED 2150/nc
tcp 0 0 192.168.5.25:22 192.168.5.1:52455 ESTABLISHED 2001/sshd: secuser
tcp6 1 0 ::1:46672 ::1:631 CLOSE_WAIT 918/cups-browsed
tcp6 0 0 192.168.5.25:80 192.168.5.26:39470 ESTABLISHED 1766/apache2
tcp6 1 0 ::1:46674 ::1:631 CLOSE_WAIT 918/cups-browsed
```

Figure 1.10: screenshot of webshell code [1].

Analyze .htaccess files for modifications. The following are examples of changes that an attacker might make to .htaccess files.

```
# The AddType directive maps the given filename extensions onto the specified content type
AddType application/x-httpd-php .htaccess
AddType application/x-httpd-php .jpg
```

Figure 1.11: screenshot of webshell code [1].

- **Prevention:** The list of precautions to take with respect to webshells is not all-inclusive. 1. Disable potentially hazardous PHP functions if not utilized. exec(), shell_exec(), passthru(), system(), show_source(), proc_open(), pcntl_exec(), eval(), and assert(). 2. Make sure no unauthorized people have access to these scripts if it is really necessary to enable those commands. Use also escapeshellarg() and escapeshellcmd()to prevent command execution vulnerabilities caused by user input is injected into shell commands. 3. If your online application uses upload forms, check sure they are safe and that they only permit the upload of file types that are on a whitelist. 4. Never rely on user feedback. 5. Use caution while using code that you may encounter on websites or online forums. 6. Installing third-party plugins for WordPress is best avoided if you do not require them. If a plugin is necessary, be sure it is reliable and regularly updated. 7. In sensitive directories like photos or uploads, disable PHP execution. 8. Lockdown user permissions on the web server[1].

- **Final Remarks:** As we have seen, utilizing a web shell and writing code are both simple. Sadly, a lot of web servers are configured in a way that even a basic script can do a lot of harm. The fundamental cause of the thousands of publicly accessible web shells is this.

It is challenging for intrusion detection and intrusion prevention systems (IDS/IPS) to identify them because there are so many variations, especially if they rely on signatures to identify such web shells. Even with behavioral analysis, some web shells are incredibly sophisticated and nearly hard to find.

Having said that, we already established that web shells are post-exploitation tools earlier in this article series. So, the most effective approach to The only way to avoid exploitation is to stop them from ever being uploaded. The Acunetix vulnerability scanner can discover entry points that could allow attackers to install web shells since it scans websites and online applications for hundreds of vulnerabilities, including code execution and arbitrary file upload vulnerabilities. Additionally, the scanner may access a list of files on the server is back end when utilizing the AcuSensor technology since a sensor is placed inside the web application. As a result, the scanner can find web shells and other malicious scripts even if they are hidden deep within directories [1].

1.8 Conclusion

The chapter is main ideas are outlined in this section, which also outlines future directions for the study of web-based cyberattacks. It might also cover the requirement for more research into fresh preventive and detection techniques to counter Internet cyberattacks. We shall discuss the machine and deep learning approaches in the following chapter.

CHAPTER 2

MACHINE AND DEEP LEARNING TECHNIQUES

2.1 Introduction

A branch of artificial intelligence known as "machine learning" focuses on creating algorithms and statistical models that let computers learn and make predictions or judgments without having to be explicitly programmed. It involves using massive datasets for training algorithms, which then use these relationships and patterns to forecast or decide on fresh data.

On the other hand, deep learning is a branch of machine learning that uses neural networks with numerous layers to examine intricate patterns and connections in data. It is motivated by the structure and operation of the human brain and is effective at many different tasks, including speech recognition, image recognition, and natural language processing.

Large volumes of data are utilized to train deep learning models and algorithms that can develop over time, getting better as they process more data, increasing their accuracy. This enables them to learn and adjust to new circumstances and makes them well-suited to complex, real-world problems. By offering insights and automating decision-making processes, machine learning and deep learning have the potential to revolutionize a variety of industries, including healthcare, banking, retail, and transportation.
Machine learning is a branch of artificial intelligence (AI) that enables systems to gain knowledge and get better over time without being explicitly programmed to do so. Data are used by machine learning to train and produce precise outcomes. The goal of machine learning is to create computer software that can access data and utilize it to educate itself.

Artificial neural networks and recurrent neural networks are related to deep learning, which is a subset of machine learning. Similar to how machine learning is formed, the algorithms are also created, however, There are numerous other tiers of algorithms in it. Artificial neural network is the collective term for all of these algorithmic network structures. The idea behind deep learning is to duplicate the human brain as closely as possible because all of the neural networks in the brain are interconnected. With the use of algorithms and their methodology, it resolves all complex issues [7]. Now let is look at the difference between Machine Learning and Deep Learning:

Table 2.1: The difference between Machine Learning and Deep Learning [8]

S.NO.	**Machine Learning**	**Deep Learning**
1	Machine Learning is a superset of Deep Learning	Deep learning is a subset of machine learning of Machine learning
2	The data represented in Machine Learning is quite different compared to Deep Learning as it uses structured data	The data representation in Deep Learning is quite different as it uses neural networks (ANN).
3	Machine Learning is an evolution of AI	Deep Learning is an evolution of Machine Learning. Basically, it represents how deep the machine learning is.
4	Machine learning consists of thousands of data points	Big Data: Millions of data points.
5	Outputs: Numerical Value, like classification of the score	Anything from numerical values to free-form elements, such as free text and sound.
6	Uses various types of automated algorithms that turn into model functions and predict future actions	Uses neural networks that process data through layers to interpret data features and relations.
7	Algorithms are selected by data analysts to examine specific variables in data sets	Algorithms are largely self-depicted on data analysis once they are put into production.
8	Machine Learning is highly used to stay in competition	Deep Learning solves complex machine learning issues.
9	Training can be performed using the CPU (Central Processing Unit)	A dedicated GPU (Graphics Processing Unit) is required for training.
10	More human intervention is involved in getting results	Although more difficult to set up, deep learning requires less intervention once it is running.

Table 2.1: The difference between Machine Learning and Deep Learning (continued) [8]

S.NO.	**Machine Learning**	**Deep Learning**
11	Machine learning systems can be swiftly set up and run but their effectiveness may be constrained	Although they require additional setup time, deep learning algorithms can produce results immediately (although the quality is likely to improve over time as more data becomes available).
12	Its model takes less time in training due to its small size	A huge amount of time is taken because of very big data points.
13	Humans explicitly do feature engineering	Feature engineering is not needed because important features are automatically detected by neural networks.
14	Machine learning applications are simpler compared to deep learning	Deep learning systems utilize much more powerful hardware and resources.
15	The results of an ML model are easy to explain	The results of deep learning are difficult to explain.
16	Machine learning models can be used to solve straightforward or a little bit challenging issues	Deep learning models are appropriate for resolving challenging issues.
17	Banks, doctor is offices, and mailboxes all employ machine learning already	Deep learning technology enables increasingly sophisticated and autonomous algorithms, such as self-driving automobiles or surgical robots.
18	Machine learning involves training algorithms to identify patterns and relationships in data	Deep learning, on the other hand, uses complex neural networks with multiple layers to analyze more intricate patterns and relationships.

Table 2.1: The difference between Machine Learning and Deep Learning (continued) [8]

S.NO.	Machine Learning	Deep Learning
19	Machine learning algorithms can range from simple linear models to more complex models such as decision trees and random forests	Deep learning algorithms, on the other hand, are based on artificial neural networks that consist of multiple layers and nodes.
20	Machine learning algorithms typically require less data than deep learning algorithms, but the quality of the data is more important	Deep learning algorithms, on the other hand, require large amounts of data to train the neural networks, but can learn and improve on their own as they process more data.
21	Machine learning is used for a wide range of applications, such as regression analysis, classification, and clustering	Deep learning, on the other hand, is mostly used for complex tasks such as image and speech recognition, natural language processing, and autonomous systems.
22	Machine learning algorithms for complex tasks, but they can also be more difficult to train and may require more computational resources	Deep learning algorithms are more accurate than machine learning algorithms.

2.2 Naive Bayes (NB)

A popular probabilistic machine learning approach for classification tasks is called Naive Bayes. Assuming that each feature contributes to the probability of the class independently of the other features, it is based on the Bayes theorem and the feature independence assumption. Although Naive Bayes can be used to find PHP web shells, it might miss intricate inter-dependencies between characteristics[9].

2.2.1 The used of algorithm

1. Gather a tagged dataset of PHP files, including both web shells and authentic PHP code, for the preparation of the data. Extract pertinent elements from each file, including syntactic patterns, keywords, or numerical measurements.
2. In order for Naive Bayes to function properly, the extracted features must be transformed into an appropriate numerical representation. Techniques like bag-of-words, n-grams, or TF-IDF may be used for this.
3. Labeling: Indicate whether the examples are web shells (positive) or real PHP code (negative) by labeling them accordingly.
4. Divide the labeled dataset into training and testing sets using the data split method. The Naive Bayes model is trained using the training set, and its performance is assessed using the testing set.
5. Model training: Use the training data to run the Naive Bayes method. Based on the feature values, Naive Bayes determines the likelihood of each class (web shell or legitimate). It is assumed that, given the class, the features are conditionally independent.
6. Model evaluation: Use relevant evaluation metrics, such as accuracy, precision, recall, or F1-score, to assess the Naive Bayes models performance. Another method for getting a more accurate estimate of the model is performance is cross-validation.
7. Prediction: Based on the feature values of new, unknown PHP files, the Naive Bayes model can be used to forecast the class label (web shell or legitimate) using training data. The model determines each class is posterior probability and assigns the class label with the highest likelihood.

Due to its independence assumption, Naive Bayes may not be the best technique for capturing intricate connections between features. However, it can still offer a quick and easy method for detecting web shells, particularly when the characteristics are largely independent or when a lot of training data is accessible.

Remember that the dataset quality, feature selection and representation, and availability of relevant training examples for both web shells and genuine PHP code will all affect how well the Naive Bayes model detects PHP web shells [9].

2.2.2 The main equation

In Naive Bayes, the basic equation connects the properties of a PHP file to the likelihood that it belongs to a specific class (web shell or genuine). Whether Gaussian Naive Bayes or Multinomial Naive Bayes is chosen the Naive Bayes algorithm will determine the precise equation. I will give the equations for the two versions that are frequently employed in text categorization jobs here.

Caussian Naive Bayes:

In Gaussian Naive Bayes, it is assumed that the features follow a Gaussian (normal) distribution. The main equation for Gaussian Naive Bayes can be expressed as follows:

$\mathrm{P}(webshell/x) = \frac{P(webshell)\cdot P(x_1/webshell)\cdot P(x_2/webshell)\cdot....\cdot P(x_n/webshell)}{P(x)}$.

$P(\text{legitimate}|x) = \frac{P(\text{legitimate})\cdot P(x_1|\text{legitimate})\cdot P(x_2|\text{legitimate})\cdot....\cdot P(x_n|\text{legitimate})}{P(x)}$.

In these equations P(webshell|x) and P(legitimate|x) indicate the probability that a PHP file, given its x-factors, is either a web shell or real code. The file is given to the class with the highest posterior probability.

P(webshell) and P(legitimate) are the previous probabilities of finding valid code or a web shell in the dataset. Based on the class proportions in the training data, these can be approximated. $P(x_i|webshell)$ and $P(x_i|legitimate)$ reflect, assuming a Gaussian distribution, the conditional probabilities of the i^{th} feature value given the class. Based on the mean and standard deviation of the feature values observed in the corresponding class in the training data, these conditional probabilities are computed.

P(x) is the probability of the evidence, which serves as a normalizing factor. Regardless of the class, it represents the likelihood of observing the feature values x. When comparing the posterior probability, it can be left out because it is the same for both classes.

Multinomial Naive Bayes:

For discrete features like word frequencies, Multinomial Naive Bayes assumes that the features follow a multinomial distribution. Following is the primary equation for Multinomial Naive Bayes:

$P(\text{webshell}|\text{x}) = P(\text{webshell}) \cdot \prod_i P(x_i|\text{webshell})$

$P(\text{legitimate}|\text{x}) = P(\text{legitimate}) \cdot \prod_i P(x_i|\text{legitimate})$

In these equations:

P(webshell|x) and P(legitimate|x) represent the posterior probabilities of a PHP file being a web shell or legitimate code, given its features x.

P(webshell) and P(legitimate) are the prior probabilities of observing a web shell or legitimate code in the dataset, estimated based on the class proportions in the training data.
$P(x_i|webshell)$ and $P(x_i|legitimate)$ represent the conditional probabilities of the i^{th} feature value given the class. In Multinomial Naive Bayes, these probabilities are typically calculated using the frequency or count of each feature value in the respective class in the training data.
It is crucial to remember that, especially for complex interactions between features, the Naive Bayes assumption of feature independence may not always hold true in practice. Naive Bayes can nonetheless offer a straightforward and computationally effective method for classification applications, such as the detection of PHP webshells.

2.2.3 Advantages of NB model

1. Simpleness: Ignorant The Bayes algorithm is a straightforward one that is simple to comprehend and use. It is computationally effective since it is built on Bayes' theorem principle and implies independence between characteristics.
2. Scalability: Naive Bayes can quite well manage many features and variables. This makes it appropriate for identifying PHP web shells, which can include many complicated features.
3. Naive Bayes requires little computation during training because it simply needs to determine the probability of each feature. Similarly to that, Naive Bayes prediction is likewise quick because it only requires basic computations based on probabilities.
4. Naive Bayes may still perform well when dealing with high-dimensional data, which is frequently the case with web shell detection. It can effectively manage enormous feature spaces and still deliver trustworthy results [9].

2.2.4 Disavantages of NB algorithm

1. Naive Bayes erroneously assumes that all features are independent of one another, which is rarely the case in real-world situations. This assumption might not be accurate in the case of PHP web shell detection because various characteristics of a web shell might be connected or associated.
2. Naive Bayes is restricted expressiveness due to its feature independence requirement. Compared to more complicated models that can capture intricate interactions between features, this may lead to lesser accuracy.
3. Sensitivity to Feature Quality: The accuracy and usefulness of the characteristics employed for detection can affect how Naive Bayes responds. The model is performance can be greatly

impacted if crucial elements are absent or unnecessary features are added.

4. Imbalanced Data: Predictions may be skewed if the training data that was used to create the Naive Bayes model is significantly out of balance (i.e., one class predominates the other). This is crucial for web shell detection because web shells are typically uncommon compared to regular PHP code [9].

2.3 Logistic Regression

Is statistical model used for binary classification tasks is logistic regression. It simulates the relationship between a collection of characteristics and the likelihood of falling into a particular class. Although Logistic Regression is not designed expressly to find PHP webshells, it can be used well in this situation [10].

2.3.1 The used of algorithm

1. Gather a tagged dataset of PHP files, including both webshells and authentic PHP code, for the preparation of the data. Extract pertinent elements from each file, including syntactic patterns, keywords, or numerical measurements.

2. In order for Logistic Regression to function, the extracted features must be transformed into an appropriate numerical representation. Techniques like bag-of-words, n-grams, or TF-IDF may be used for this.

3. Labeling: Designate a label for each sample depending on whether it belongs to the webshell (positive class) or the valid PHP code (negative class) categories.

4. The labeled dataset should be split into training and testing sets. The Logistic Regression model is trained using the training set, and its performance is assessed using the testing set.

5. Apply the Logistic Regression algorithm to the training data while building a model. A logistic function also called the sigmoid function, is used in logistic regression to represent the relationship between characteristics and the likelihood of being in the positive class.

6. Model evaluation: Use relevant evaluation metrics, such as accuracy, precision, recall, or F1-score, to assess the effectiveness of the Logistic Regression model. A more reliable performance estimation method is cross-validation.

7. Prediction: Based on the attributes of a PHP file, you can use the trained Logistic Regression model to forecast the likelihood that it is a webshell. The model uses the learned coefficients to determine the probability, and then it applies the sigmoid function to transform the result

into a probability between 0 and 1.

Based on the estimated probability, the decision threshold can be changed to assign the expected class label (webshell or legitimate). For instance, if the threshold is set at 0.5, probabilities greater than 0.5 are labeled as webshells, whereas probabilities lower than 0.5 are labeled as genuine code.

It is vital to keep in mind that a linear relationship between the features and the log-odds of being in a positive class is presupposed by the logistic regression method. Despite the possibility that this supposition does not fully account for the complexity of PHP webshells, Logistic Regression can nevertheless offer a reliable foundational model for detecting tasks.

Keep in mind that the quality and representativeness of the dataset, the choice and representation of features, and the availability of sufficient training instances for both webshells and legal PHP code will all affect how well the Logistic Regression model for PHP webshell identification performs [10].

2.3.2 The main equation

The main equation in logistic regression connects the input features to the likelihood of falling into a particular class (in this case, whether a PHP file is a web shell or not). The logistic function also referred to as the sigmoid function, is used in the equation to convert the linear combination of information into a probability value. The key equation for logistic regression is as follows.

$P(\text{webshell / x}) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n)}}$.

In this equation:

P(webshell|x): represents the probability that a PHP file is a webshell given its features x.

e is the base of the natural logarithm (approximately 2.718).

$\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients (weights) associated with each feature x1, x2, ..., xn. These coefficients are estimated during the model training process.

$x_1, x_2, \ldots, x_n$ are the feature values of the PHP file.

The linear combination of characteristics and coefficients is mapped by the logistic function in the equation to a value between 0 and 1. This number shows the likelihood that the PHP file is a webshell, as predicted. The logistic function makes sure the result is consistently contained within the acceptable probability range.

The anticipated probability is subjected to a decision threshold in order to produce a binary classification choice. According to a standard threshold of 0.5, a PHP file is categorized as a

webshell if the estimated chance is higher than 0.5; otherwise, it is labeled as legitimate code. By maximizing the likelihood of the observed labels in the training data, the coefficients (0, 1, 2,..., n) are computed during the training phase. The ideal coefficient values are often determined using a variety of optimization procedures, such as gradient descent

It is crucial to keep in mind that the particular equation given considers just one independent variable (n = 1). Multiple features may exist in practice (x1, x2,..., xn), and the equation is extended to account for all of the characteristics in the linear combination.

To avoid overfitting and enhance the model is generalizability, regularization techniques like L1 or L2 regularization can also be used on the equation.

Logistic regression can accurately predict the likelihood that a PHP file is a webshell based on its attributes by training the model with the right features and a labeled dataset [10].

2.3.3 Advantages of LR algorithm

1. Logical Interpretability The interpretation of regression models is rather easy. Each feature is coefficient offers information about the significance and directionality of the feature in calculating the likelihood of a webshell. Understanding the underlying patterns and characteristics of observed webshells can be facilitated by this interpretability [10].

2. Efficiency and simplicity: A straightforward and effective computational approach is logistic regression. It is comparatively simple to comprehend, use, and train. The model is suited for real-time or high-traffic web server scenarios since it can be trained quickly, even on big datasets.

3. Binary classification-friendly: The goal of webshell detection is to categorize occurrences as either malicious or benign, and logistic regression is specifically made for binary classification tasks. The output of the model is a probability estimate that can be thresholded further to decide on the classification.

4. Robustness to noise: Logistic Regression models are capable of handling illogical or noisy features. The model can concentrate on the pertinent aspects of PHP webshells by giving lesser weights to less useful data, resulting in robust and accurate detection [10].

2.3.4 Disavantages of LR algorithm

1. Limited expressiveness: Compared to more complicated models like Random Forest or Deep Neural Networks, Logistic Regression models are less expressive. They could have trouble capturing intricate connections or non-linear dependencies found in webshell programs. They may therefore perform less accurately than more advanced models on complex webshell detection tasks.
2. Logistic regression makes the assumption that there is a linear relationship between the target variable is log-odds and its characteristics. The relationship may not be correctly represented by the model if it is strongly non-linear. For the purpose of properly capturing non-linear relationships, feature engineering methods or the use of more sophisticated models may be required.
3. Engineering specifications for the feature: logistic To make predictions, regression models use manually created characteristics. The caliber and applicability of the chosen features determine the model is efficacy. To extract informative characteristics from PHP webshell code, careful feature engineering and domain knowledge are required.
4. Sensitivity to outliers: Because outliers can significantly affect the coefficient estimates, logistic regression models may be sensitive to them. Outliers may skew the decision boundary of the model and impair overall performance. To address this problem, robust preparation approaches or outlier handling procedures should be taken into consideration [10].

2.4 Random Forest

An efficient machine-learning model for identifying PHP web shells is Random Forest. Multiple decision trees are combined in this ensemble learning technique to produce predictions. [11]

2.4.1 Decision trees

It might be beneficial to begin by briefly outlining the decision tree algorithm because the random forest model is made up of numerous decision trees. Should I surf? is a common starter question for decision trees. A sequence of queries, such as "Is it a long period swell?" and "Is the wind blowing offshore?," can then be used to arrive at an answer. These inquiries serve as the decision nodes in the tree, which divide the data. Each query aids a person in coming to a conclusion, which is indicated by the leaf node. The "Yes" branch will be followed by

observations that meet the requirements, while the opposite path will be taken by observations that do not. Decision trees try to identify Often trained using the Classification and Regression Tree (CART) technique, decision trees look for the optimal split to subset the data. The quality of the split can be assessed using metrics like mean square error (MSE), Gini impurity, and information gain.

A classification problem using the class labels "surf" and "do not surf" is illustrated by this decision tree. Although being a prevalent supervised learning approach, decision trees are susceptible to issues like bias and overfitting. But, when several decision trees are combined into an ensemble using the random forest algorithm, they are able to forecast outcomes more accurately, especially when the individual trees are uncorrelated [12].

2.4.2 Ensemble methods

number of classifiers, such as decision trees, are used in ensemble learning techniques, and their predictions are combined to determine the most common outcome. The most popular ensemble techniques are boosting and bagging, commonly referred to as bootstrap aggregation. Leo Breiman created the bagging method in 1996; in this method, a random sample of data in a training set is selected with replacement meaning that the individual data points can be chosen more than once. (Link lies outside of IBM.com.) (PDF, 810 KB). Following the generation of several data samples, these models are individually trained, and depending on the task, for example, classification or regression the average or majority of those predictions result in a more accurate estimate. This method is frequently applied to lessen variation in noisy dataset [11].

2.4.3 Random forest algorithm

The bagging method is extended by the random forest algorithm, which uses feature randomness in addition to bagging to produce an uncorrelated forest of decision trees. A random subset of features is generated via feature randomness, also known as feature bagging or "the random subspace approach" (PDF, 121 KB), which guarantees low correlation among decision trees. The main distinction between decision trees and random forests is this. Random forests merely choose a portion of those feature splits, whereas decision trees take into account all possible feature splits. If we return to the "should I surf?" example, the questions I would pose to arrive to the forecast might not be as thorough as those posed by someone else. By taking into account every possible variation We can lower the likelihood of bias, overfitting, and general

volatility in the data, leading to predictions that are more accurate. estimate. This method is frequently applied to lessen variation in noisy datasets [11].

2.4.4 Execution of RF algorithm

There are three key hyperparameters for random forest algorithms that must be set prior to training. Node size, tree count, and sampled feature count are a few of them. From there, classification or regression issues can be resolved using the random forest classifier. Each decision tree in the ensemble that makes up the random forest method is built of a data sample taken from a training set with a replacement known as the bootstrap sample. One-third of the training samplealso referred to as the out-of-bag (oob) sampleis set aside as test data; we'll return to this sample later. The dataset is subsequently given a second randomness injection by feature bagging, increasing its variety and lowering its correlation. between decision trees. The prediction will be determined differently depending on the type of issue. The individual decision trees will be averaged for the regression job, and for the classification task, the predicted class will be determined by a majority vote, or the most common categorical variable. In order to complete that prediction, the oob sample is then used for cross-validation [11].

2.4.5 Advantages of RF algorithm

1. High accuracy: The ensemble learning technique Random Forest mixes different decision trees to produce predictions. In classification tasks, such as the detection of PHP web shells, it can achieve excellent accuracy. The model is capacity to use several trees' aggregate judgments aids in lowering overfitting and enhancing generalization.

2. Random Forest models can withstand noisy and anomalous data without losing accuracy. Without noticeably degrading their performance, they can deal with inconsistent or irrelevant data. This resilience helps identify web shells that may use variants or obfuscation tactics.

3. Estimating feature importance: Random Forest models offer a scale for feature relevance. The most pertinent features for webshell detection can be found with the aid of this information. Security analysts can learn more about the underlying patterns and behaviors of web shells by knowing which aspects of PHP code are more important in the classification process.

4. Efficiency and scalability: Random Forest models are capable of effectively handling big datasets. Model construction can be sped up by parallelizing the training process. Additionally, Random Forest is suitable for real-time or high-traffic web servers because the prediction process is typically quick.

2.4.6 Disavantages of RF algorithm

1. Model interpretability: Compared to more straightforward models like Naive Bayes, Random Forest models are typically thought to be less interpretable. It is difficult to grasp and comprehend the model in the decision-making process because of Random Forest is an ensemble structure and the interactions between several decision trees.
2. Complexity of computation: Random Forest models can be expensive to compute, especially when working with many trees and features. As a model becomes more complicated, training and inference periods can lengthen, which could be problematic in real-time or high-throughput environments.
3. Random Forest models feature a number of hyperparameters that must be properly set to operate at their best. Experimentation and validation are needed to determine the ideal number of trees, the maximum depth, and other parameters, which can be time-consuming and need computer resources.
4. Memory usage: During training and inference, Random Forest models need to store a number of decision trees in memory. The amount of RAM needed grows along with the number of trees. Memory limitations may provide problems for large-scale PHP webshell detection, particularly in contexts with limited resources.

2.5 Support Vector Machine (SVM)

A support vector machine (SVM) is a supervised learning technique used for a variety of classification and regression issues, including medical signal processing, natural language processing, speech and picture recognition, and signal processing in the context of signal processing. Finding a hyperplane that effectively divides data points from one class from those from another is the goal of the SVM algorithm. The hyperplane with the widest margin between the two classes is referred to as "best"; plus and minus are used to illustrate this. Margin is the maximum thickness of the slab that is perpendicular to the hyperplane but has no interior data points. The approach can only locate such a hyperplane for linearly separable issues; for most real problems, it maximizes the soft margin, allowing a limited number of solutions [13]. We need to be aware of the applications for SVM at the moment in order to begin to comprehend this:

- Face detection
- Classification of images

- Text and hypertext categorization
- Bioinformatics
- Handwriting recognition
- Yes, SVM has a role to play in all of that.

This approach uses supervised machine learning. Here, machine learning algorithms make predictions based on historical input data. Support vector machines, which are utilized for classification and regression analysis, are essentially supervised learning models[13].

2.5.1 The importance of SVM

A model called SVM can forecast unknown data. We can simply train our model to recognize apples and strawberries, for instance, provided we have pre-labeled data on apples and berries. As a result, anytime we give it fresh information that is unknown, it can categorize it as either strawberries or apples.

That is SVM in action. Based on the labeled data it already has, it analyzes the data and places it in one of the two categories. The apples will be sorted under the apple category and the strawberries under the strawberry category, just like in the previous example [13].

2.5.2 the work of prediction

Here, using the labeled sample of data from the first graph, we apply our Support Vector Machine. We also demarcate the boundaries between the two groups. The decision boundary is the name of this line. Here, the decision boundary features strawberries on one side and apples on the other.

As shown in the third graph, new data is now automatically added to the right or left side of the decision border depending on which category it belongs to. Also, we may anticipate the unknown and categorize it as either an apple or a strawberry based on which side of the line the unknown sample data falls [13].

2.5.3 Advantages of SVM

- **Regularization capabilities:** L2 Regularization (Ridge Regression) is a feature of SVM. The squared magnitude of the coefficient is added to the loss function as a penalty term

by L2 regularization. It has good generalizability, which keeps it from fitting too tightly (modeling error which occurs when a function is closely fit in a limited set of data points).

- **Handles non-linear data efficiently:** Via the Kernel function, SVM effectively handles non-linear data (where data items are not arranged consecutively).

- **Solves both Classification and Regression problems:** Whereas SVR (Support Vector Regression) is used for regression issues, SVM is used for classification issues.

- **Stability:** The hyperplane is unaffected by even a tiny change in the data, demonstrating the SVM model's stability [13].

2.5.4 Disadvantages of SVM

- **Choosing an appropriate Kernel function is difficult:** Complexity is involved in selecting a suitable Kernel function (to handle the non-linear data). What occurs is that using a high dimensional kernel function may result in the generation of an excessive number of support vectors, which slows down training.

- **Extensive memory requirement:** To store all of the support vectors in memory, you obviously need a lot of memory. The amount of the training dataset causes this figure to increase with time.

- **Long training time:** Large datasets and a lengthy training period are needed for SVM [13].

2.6 K-Nearest Neighbors (KNN)

A machine learning model called K-Nearest Neighbors (KNN) is utilized for classification and regression problems. Although KNN was not created with the intention of identifying PHP web shells, it can be modified and used for this purpose[14].

2.6.1 The used of algorithm

1. Build a collection of PHP files that contains both benign and malicious samples for the dataset. A feature vector should be used to represent each PHP file. The features could consist of traits like file size, function calls, variable names, and other pertinent qualities [14].

2. To construct a feature vector representation, extract significant features from the PHP scripts. The features you choose will depend on the exact traits of the web shells you want to find. For instance, you may take into account characteristics connected to questionable function usage, peculiar variable names, or particular patterns frequently observed in web shells [**?**].
3. Label each PHP file in the collection as either benign or malicious. It should be clear from the labeling whether a file is a webshell or not. An initially labeled dataset is needed for training purposes in this step.
4. Divide the labeled dataset into a training set and a test set for training purposes. The KNN model is trained using the training set. The KNN method discovers patterns in the feature vectors of the labeled data during training [14].
5. KNN Model Construction: Using the training set, train the KNN model. The feature vectors are arranged in a multidimensional space by the KNN model, which also associates each vector with a corresponding label [14].
6. Identifying Webshells: Create a feature vector by extracting the features of a fresh, unlabeled PHP code. Use the trained KNN model to classify the file after that. Based on the similarities between the PHP file is features and those of the training samples, the KNN algorithm labels the PHP file as either benign or malignant [14].

2.6.2 The main equation

There is no specific equation for the K-Nearest Neighbors (KNN) approach designed to find PHP web shells. But the fundamental idea behind KNN is to locate the K closest neighbors of a sample in a feature space and use their labels to make predictions. An overview of the KNN algorithm is given below:
1. Calculate the distance between the feature vectors of all the labeled samples in the training set and the provided sample (which represents a PHP file). The particular qualities and manner in which the features are represented determine the distance metric to be used. Euclidean distance and cosine similarity are two widely used distance measurements.
2. Choose the K samples from the training set that are closest to the given sample, or the K nearest neighbors. The "nearest neighbors" of the provided sample are these K samples.
3. Determine the majority class (good or bad) by looking at the labels of the K closest neighbors. The typical method for doing this is to count the samples in each class, then choose the class with the highest count.
4. Prediction: Assign the sample the majority class label. This label represents the PHP file is

projected class, identifying whether the file is a good or bad webshell [14].

2.6.3 Advantages of KNN algorithm

1. Simpleness: KNN is a straightforward algorithm that is simple to comprehend and use. Finding the k closest neighbors to a given data point and classifying it according to the dominant class among those neighbors are the fundamental principles on which it operates [15].
2. No explicit training phase: Because KNN is a lazy learner, it does not need one. Instead, it employs the labeled data itself directly for classification. When dealing with dynamic or developing threats like web shells, this can be useful because the model can easily adjust to new patterns without retraining [15].
3. Non-parametric: Because KNN is non-parametric, it makes no assumptions about the distribution of the data at its core. Due to its adaptability, it can handle complex and non-linear correlations between features, which may make it useful for identifying different kinds of webshell patterns [15].
4. Local information capture: When classifying data points, KNN takes into account their immediate surroundings. This can be helpful for webshell detection because it can discover commonalities amongst malicious scripts nearby, making it easier to spot common patterns and features [15].

2.6.4 Disadvantages of the KNN algorithm

1. Complexity of computation: KNN can be expensive to run, especially when working with huge datasets. The time needed to categorize a new sample can be significant for webshell detection, where the number of instances and attributes can be high. This may prevent real-time detection, particularly in web servers with significant traffic [15].
2. High memory usage: Because KNN calculates distances between fresh samples and all training instances, it has to store the full training dataset in memory. For resource-constrained, large-scale online applications, this memory demand can be troublesome [15].
3. KNN can be sensitive to the scale of features because it relies on computing the distances between data points. Features with varied ranges or units may produce categorization results that are not correct. Consequently, to achieve optimal performance, preprocessing and normalization of features may be required [15].

4. Imbalanced classes: The model is performance may be biased toward the majority class if the dataset used to train the KNN model has imbalanced classes and the majority of samples belong to the non-malicious class. As a result, spotting unusual webshell instances may be less accurate [15].

2.7 Deep Neural Network

Is a kind of artificial neural network where the input and output layers are separated by a number of hidden layers. It can be used to find PHP web shells [16] efficiently.

2.7.1 The used of model

1. Create a dataset of PHP scripts that include both reputable and malicious samples. A feature vector should be used to represent each PHP file. Various criteria, including function calls, variable names, file size, and other pertinent attributes, can be used to extract the features.
2. Labeling the Dataset: To determine whether a PHP file is a webshell or not, label each file in the dataset as benign or malicious. For training purposes, this needs an initially labeled dataset.
3. DNN Architecture Design: Create the DNN model is architecture. The number of hidden layers, the number of neurons in each layer, the activation functions to be utilized, and any additional pertinent parameters should all be specified. Based on the difficulty of the webshell detection task and the available computational resources, the architecture should be selected.
4. Training: Make use of the labeled dataset to train the DNN model. To distinguish between good and bad web shells, the model learns the patterns and relationships in the feature vectors of the PHP scripts during training. The weights and biases of the neural network are frequently updated using optimization techniques like backpropagation and gradient descent.
5. Hyperparameter Tuning: Improve the performance of the DNN model by modifying its hyperparameters. Hyperparameters might be anything from the learning rate to the batch size to the regularization methods. These hyperparameters can be tuned to increase the model is precision and generalizability.
6. Identifying Webshells: Create a feature vector by extracting the features of a fresh, unlabeled PHP code. Classify the file using the learned DNN model. The DNN model runs the feature vector through each of its layers while running a number of mathematical operations to

determine whether the file is dangerous or benign. The DNN model is result shows how likely or likely it is that the PHP file is a webshell [16].

2.7.2 The main equation

A DNN (Deep Neural Network) for PHP webshell detection uses a core equation made up of mathematical operations carried out at each layer of the network. DNN models do not, it should be noted, have a single equation that describes the entire network. Instead, each layer is equation and the activation function are addressed separately. The equations used in a typical DNN for webshell identification are summarized here.

1. Feedforward Equation:

Each layer of the DNN processes the input data and passes it through a number of modifications during the feedforward phase. A single neuron in a layer's equation is represented by the following equation:

Z = W * X + b

A = f(Z)

where:

Z‘ is the weighted sum of inputs ‘X‘ multiplied by the corresponding weights ‘W‘, with the bias ‘b‘ added.

f()‘ represents the activation function applied element-wise to ‘Z.

A‘ is the output of the neuron, which becomes the input for the next layer.

2. Activation Functions:

The activation function 'f()' adds non-linearities to the network and enables it to learn intricate linkages. Sigmoid, tanh, ReLU, and its variations such Leaky ReLU and PReLU are common activation functions.

3. Backpropagation Equation:

The backpropagation algorithm is used to modify the DNN is weights during the training phase based on the discrepancy between the expected output and the actual label. Gradient calculations as well as network weight and bias updates are part of the backpropagation equations.

The actual structure and equations of a DNN for webshell detection can vary depending on the individual architecture, activation functions, and other design considerations, even though these equations give a high-level perspective. To further enhance the model's performance and generalizability, more sophisticated approaches like regularization, dropout, and batch

normalization may be used [16].

2.7.3 Advantages of the DNN model

1. Learning complicated non-linear correlations between input features and the target variable is possible with DNN models thanks to their non-linear feature learning capabilities. Because the model can recognize complex patterns and dependencies in the code even when they are not explicitly declared or obvious, it is helpful for identifying PHP web shells.
2. Automatic feature learning: From the unprocessed input data, DNN models may automatically identify pertinent features. They can find hidden representations and extract highly discriminatory high-level abstract features for webshell identification. As a result, human feature engineering is no longer necessary, and the model can now adapt to the evolving properties of webshell scripts.
3. Scalability: DNN models are capable of handling a large number of features and scaling to huge datasets. This is advantageous for webshell identification since it enables the model to efficiently analyse and learn from enormous quantities of PHP code, improving the likelihood of accurate detection and handling of massive web applications.
4. More data leads to better performance: DNN models often perform better with more labeled training data. The model can generalize better and capture a larger range of webshell features as the dataset expands. DNN models may perform better and more effectively generalize to fresh and undiscovered webshell instances as more data becomes available [16].

2.7.4 Disavantages of the DNN model

1. High computational needs: When working with deep architectures or huge datasets, DNN models are computationally expensive to train and demand significant computer resources. DNN models may need to be trained and run on powerful hardware, with lots of memory, and over an extended period of time.
2. Large training data requirements: To function well, DNN models frequently need a lot of labeled training data. It can take a lot of time and effort to gather and categorize a diverse collection of webshell scripts, especially in light of the continually changing nature of webshell threats.
3. Regularization and overfitting: DNN models are prone to overfitting, where the model memorizes the training data rather than discovering patterns that may be applied to new

data. When the model is overly sophisticated in comparison to the training data, overfitting might happen. To reduce overfitting and enhance generalization, regularization approaches like dropout or weight regularization must be used.

4. Interpretability: Because DNN models are sometimes seen as "blackbox" models, it might be difficult to read and comprehend how decisions are made. When explanations or regulatory compliance are crucial factors, the lack of interpretability may be a constraint [16].

2.8 Convolutional Neural Network

CNN (Convolutional Neural Network) is model for the efficient identification of PHP web shells is a deep learning architecture made especially to examine PHP code and recognize malicious web shell scripts [17].

2.8.1 The used of algorithm

1. PHP code snippets or files representing the web shell script that will be examined are provided as input to the model in the input layer. To prepare the input data for the CNN model, it may be preprocessed, tokenized, or encoded [17].

2. Convolutional Layers: The fundamental building blocks of a CNN are convolutional layers. These layers are made up of filters that convolution the incoming data to extract regional patterns and features. The filters use mathematical operations as they move over the input, looking for pertinent PHP code features [17].

3. Pooling Layers: Pooling layers assist in downsampling the output feature maps after the convolutional layers. Max pooling or average pooling are frequent forms of pooling processes that lower the spatial dimensions of the features while preserving crucial data [17].

4. Fully Connected Layers: Fully connected layers are often employed to do high-level reasoning and decision-making after the convolutional and pooling layers. These layers link the previously learned features to the output classes (web shell or non-web shell) by combining the learned features from prior layers [17].

5. Activation Functions: By introducing non-linearity to the model, activation functions enable it to learn intricate correlations between features. Common activation functions include sigmoid, softmax, and ReLU (Rectified Linear Unit) [17].

6. Output Layer: The CNN model is final layer, the output layer delivers the categorization or prediction outcomes. It may produce probabilities or predictions that would indicate whether

a web shell was there or not in the case of web shell detection [17].

7. Training: PHP code samples from a tagged dataset that have been classified as either web shells or innocuous code are used to train the CNN model. Using methods like backpropagation and gradient descent, the model modifies its internal parameters during training to reduce the discrepancy between expected and real labels [17].

2.8.2 The main equation

No specific "main equation" serves as the model is definition in a CNN model for the accurate detection of PHP web shells. Instead, a CNN model is made up of a number of interconnected layers and algorithms that combine to recognize and categorize patterns in the incoming data [17].

The different layers of a CNN model do, however, entail significant mathematical operations and equations. Here are some basic equations frequently used in CNN models :

Convolution Operation:

Output feature map: Convolution applies a filter/kernel to the input data, producing a feature map. The output feature map size is determined by the input size, kernel size, stride, and padding.

Equation:

Output = Input * Filter.

Activation Function:

The CNN model gains non-linearity via activation functions, which enables it to recognize intricate patterns.

Common activation functions include sigmoid, softmax, and ReLU (Rectified Linear Unit).

Equation (ReLU):

f(x) = max(0, x).

Pooling Operation:

Downsampling the feature maps by pooling layers results in smaller spatial dimensions while preserving crucial data.

Max pooling and average pooling are two common varieties of pooling processes.

Equation (Max Pooling):

Output = max(Input).

Fully Connected Layer:

Highlevel reasoning and decision-making are carried out by fully connected layers, which link every neuron in the preceding layer to the neurons in the current layer.

Equation:

Output = Activation(W * Input + b).

Loss Function:

During training, the learning process is guided by the loss function, which calculates the difference between the expected output and the actual label.

While categorical cross-entropy is utilized for multi-class classification, binary cross-entropy is a common loss function for binary classification.

Equation (Binary CrossEntropy):

$$\text{Loss} = -(y \cdot \log(p) + (1 - y) \cdot \log(1 - p)).$$

An efficient detection method for PHP web shells is produced by combining these equations, which are the constituent parts of a CNN model. Depending on the specifications of the task and the dataset being utilized, the CNN model is specific architecture and configuration may change [17].

2.8.3 Advantages of CNN model

1. Identification of local patterns: CNN models are particularly good at identifying local patterns and features in data. CNN models can be trained to identify certain code fragments or patterns that are suggestive of malicious scripts in the context of PHP webshell detection. This enables them to concentrate on the crucial local characteristics while disregarding unimportant code [18].

2. Learning appropriate features automatically from raw input data is possible with CNN models, negating the need for human feature engineering. The model can recognize hierarchical representations of features at various levels by utilizing convolutional layers, allowing it to capture both lowlevel and highlevel PHP webshell characteristics [18].

3. CNN models are translationally invariant, which means they can find patterns no matter where they are in the input. Given that malicious code can be injected into PHP scripts at various locations, this characteristic is helpful for webshell detection. No matter where they are in space, relevant patterns can be found by the CNN [18].

4. Noise and variation resistance: CNN models are renowned for their resistance to noise and variation in the input data. They are efficient at identifying various PHP webshell variants because they can handle subtle alterations or obfuscation techniques used on webshell programs. This robustness makes accurate detection possible even in the face of code updates or evasion attempts [18].

2.8.4 Disavantages of CNN model

1. Large volumes of labeled training data are frequently necessary for CNN models to perform at their best. It can be difficult and timeconsuming to gather and categorize a diverse collection of webshell scripts, especially in light of the continually changing nature of webshell threats [18].

2. Computer resources needed and computational complexity: CNN models can be resource-intensive to run, especially when working with deep architectures or big input volumes. It may be necessary to invest a significant amount of computing resources in training and running CNN models, requiring strong hardware and enough memory. In places with limited resources, this may be a restriction [18].

3. Interpretability: CNN models are frequently referred to as "black-box" models, which means it can be difficult to grasp how they make decisions and how they operate internally. When attempting to explain the circumstances around the detection or when there is a requirement for regulatory compliance, this lack of interpretability may be a disadvantage [18].

4. Only limited generalization to previously unknown code patterns is possible with CNN models, which are good at spotting patterns that were observed during training. They might have trouble extrapolating to previously undiscovered or vastly different code patterns, though. It is possible that the model won not be effective at detecting specific webshell versions if it hasn not seen such variants in the training data [18].

2.9 Long ShortTerm Memory

A recurrent neural network (RNN) architecture called an LSTM model for the efficient detection of PHP web shells is made to scan sequential data, like code snippets or series of PHP instructions, and spot dangerous web shell scripts [19].

2.9.1 The used of model

1. PHP code snippets or sequences are used as the model is input layer to represent the web shell script that will be examined. To prepare the input data for the LSTM model, it may be preprocessed, tokenized, or encoded.

2. LSTM Layers: The fundamental elements of an LSTM model are LSTM layers. These layers are made to manage sequential data and capture long-term dependencies. LSTM cells are appropriate for evaluating code snippets because they feature a memory state that enables them to retain and update information over time.

3. In an LSTM model, each LSTM cell keeps track of both a hidden state and a cell state. While the cell state uses gates within the LSTM cell to selectively remember and forget information, the hidden state carries information across the sequence.

4. Bidirectional LSTM: A bidirectional LSTM processes the input sequence using two LSTM layers in both the forward and backward directions. This enables the model to gather contextual data from both historical and upcoming inputs.

5. Fully Connected Layers: Fully connected layers can be used to carry out high-level deliberation and reasoning after the LSTM layers. These layers link the previously learned features to the output classes (web shell or non-web shell) by combining the learned features from prior layers.

6. Activation Functions: By introducing non-linearity to the model, activation functions enable it to learn intricate correlations between features. LSTM models frequently use ReLU (Rectified Linear Unit), sigmoid, or softmax as activation functions.

7. Output Layer: The output layer serves as the LSTM model is final layer and gives the results of any classification or prediction. It may produce probabilities or predictions that would indicate whether a web shell was there or not in the case of web shell detection.

8. Training: PHP code samples from a tagged dataset that have been classified as either web shells or innocuous code are used to train the LSTM model. Using methods like gradient descent and backpropagation across time, the model modifies its internal parameters during training to reduce the discrepancy between expected and real labels [19].

2.9.2 The main equation

uses a number of equations and processes to process sequential input and generate predictions. The following are the primary equations that an LSTM model employs for the efficient detection of PHP web shells:

1. Input Gate: . The amount of new data that should be saved in the cell state is decided by the input gate.

.Equation: $\mathrm{i}_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$

2. Forget Gate: . The forget gate regulates how much the prior cell state is remembered or not.

.Equation:

$\mathrm{f}_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$ 3. Cell State Update: . With the help of the input and forget gates, the cell state is updated by merging the new data with the prior cell state. .Equation: $\mathrm{c}_t = f_t \cdot c_{t-1} + i_t \cdot \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c)$ 4. The degree to which the cell state affects the output and hidden state is determined by the output gate. .Equation: $\mathrm{o}_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$

5. Hidden State Output: . The output gate directs the calculation of the hidden state using the modified cell state. .Equation: $\mathrm{h}_t = o_t \cdot \tanh(c_t)$ 6. Fully Connected Layers: . After the LSTM layers, fully connected layers can be added to do high-level decision-making and reasoning. The individual architecture and design decisions affect the equations for these layers. The fundamental functions of an LSTM model are described by these equations. The equations are used by the model as it iterates through the sequential input data, updating the cell state, hidden state, and predictions at each time step. In addition to x_t representing the input at time step t and h_t-1 representing the hidden state at the preceding time step, the parameters W, U, and b reflect the weight matrices and bias terms connected to each equation. The LSTM model is given non-linearity by using the sigmoid and tanh activation functions [19].

2.9.3 Advantages of LSTM model

1. LongTerm Relationships Can Be Captured: The LSTM model was created primarily to identify longterm relationships in sequential data. This qualifies it for identifying web shells that may display intricate patterns and dependencies in PHP code fragments.

2. PHP code snippets can come in different lengths, but LSTM models can handle variable-length sequences just as well. The memory cell of the LSTM enables it to store and update

data over time, making it suitable for processing sequences of various lengths.
3. Robust to Noise and Irrelevant Data: LSTMs are able to train themselves to disregard noisy or irrelevant input sequences. This can be helpful when detecting web shells because there might be other legitimate code there.
4. Contextual Understanding: By keeping hidden states that store data from earlier inputs, LSTMs are able to comprehend the context of the input sequences. This might increase the precision of web shell detection by allowing the model to take into account the full sequence when making predictions [19].

2.9.4 Disavantages of LSTM model

1. Computing Complexity: When compared to other models, LSTMs might be computationally more expensive, especially when working with huge datasets or sequences. An LSTM model may need a lot of computational time and resources to train and evaluate.
2. Need for Enough Training Data: In order to generalize successfully, LSTM models, like other deep learning models, often need a lot of training data. It can be difficult to gather a sizable and varied collection of tagged PHP code snippets, comprising both malicious and benign code.
3. Overfitting: When trained on small or unbalanced datasets, LSTMs are prone to overfitting. If the model is overfitted, it might not generalize well to brand-new web shell instances that haven't been seen before, and it might make wrong predictions.
4. Interpretability: LSTMs are regarded as "black-box" models, which makes it challenging to decipher the inner workings and comprehend how the model came to a specific prediction. In situations where explainability is essential, this lack of interpretability may be a disadvantage [19].

2.10 Features extraction

Datasets made up of formats like text and image can be used to extract features in a format that machine learning algorithms can understand using the sklearn.feature_extraction module [20].

Lexical, syntactical, semantical, statistical, and abstract information can be extracted from webshell scripts and categorized into five different classes based on their nature [20].

- **Lexical features** Describe the style of writing used in the script's sources. To conceal the precise purpose of the script, adversaries may employ specialized keywords as well as obfuscated instructions and arguments inside the code. Thus, scripts with a limited number of strings and precise patterns in tags or comments are produced. The quantity of strings and the use of malicious text patterns are examples of lexical traits found in comments [20].

- **Syntactical features** a script's use of expressions, variables, and functions. System calls and potentially hazardous functions can be used by adversaries to retrieve critical files or escalate their privileges. Additionally, loops can be used to reveal passwords while conditional statements can be used to adjust the webshell to the target server platform. Syntactical aspects include the frequency with which loops and conditional statements are used, as well as the use of potentially harmful functions and specialized language constructs [20].

- **Semantical features** Syntactic and lexical characteristics Semantical characteristics offer intentions that may be deduced from lexical and syntactical components, providing a basic picture of how webshells are created. For instance, one can just take into account the presence of loops over port scanning function calls or login access rather than the proportion of arbitrary loops included in scripts [20].

- **Statistical features** To get beyond firewalls, hostile webshells frequently use encrypted and obfuscated code. Malicious webshells may also employ encryption features for security purposes. Therefore, by comparing some statistical values of webshells with regular files, statistical features are helpful in differentiating malicious webshells. Information entropy and compression ratio are two examples of such characteristics [20].

- **Abstract features** transcend lexical, syntactic, and semantic characteristics. They are helpful in exposing concealed webshell components that are impossible to find via syntactical and semantic analysis. In this study, vectorized data such as source code, op-code, and web traffic are referred to as abstract features. Deep learning-based techniques are the ones that primarily employ these features [20].

2.11 Conclusion

In this chapter, we explored various machine and deep learning techniques for the detection of PHP webshells. We discussed the main models, their advantages, and inconveniences, as well as their applicability and features extraction.

We began by introducing the different models, including Naive Bayes (NB), Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Deep Neural Network (DNN), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). Each model was presented with an overview of its functionality and purpose in the context of PHP webshell detection.

Throughout our discussion, we highlighted the advantages and disadvantages of each model. Naive Bayes (NB) is known for its simplicity and efficiency, although it assumes independence between features. Logistic Regression (LR) offers interpretability and is suitable for binary classification tasks, but it may struggle with non-linear relationships. Random Forest (RF) combines decision trees to improve performance and handle non-linear data, but it can be computationally expensive. Support Vector Machine (SVM) achieves good generalization by finding an optimal hyperplane, yet it may be sensitive to parameter tuning. K-Nearest Neighbors (KNN) relies on local similarity, making it simple to understand, but it can be sensitive to noise and requires a larger dataset. Deep Neural Network (DNN) and Convolutional Neural Network (CNN) excel at learning complex patterns and hierarchical representations, but they may require large amounts of labeled data and extensive computational resources. Long Short-Term Memory (LSTM) is effective for sequential data analysis, particularly in capturing long-term dependencies, but it can be susceptible to overfitting and is computationally intensive.

CHAPTER 3

MACHINE AND DEEP LEARNING TECHNIQUES FOR DETECTION OF PHP WEBSHELLS

3.1 Introduction

This chapter focuses on implementing the proposed approach for detecting PHP web shells using machine and deep learning techniques within an ensemble learner model. It overviews the resources, programming language, and development environment used. The chapter outlines the step-by-step implementation process, including dataset preprocessing, machine and deep learning techniques selection, and ensemble model configuration. The second part presents the experimental results, evaluating the performance of the ensemble learner model and comparing different techniques. Overall, this chapter serves as a guide for implementing the proposed approach, providing insights into the methodology and enabling further research in webshell detection using machine and deep learning.

3.2 Related Work

In recent years, there has been a lot of interest in the machine and deep learning models for identifying PHP webshells. Numerous scholars have investigated various strategies to address this issue. According to features retrieved from the code, some studies have focused on using conventional machine learning methods, including Support Vector Machines (SVM) [21] or Random Forests [11], on categorizing PHP files as either webshells or legal code. These characteristics could include variable names, function calls, or typical webshell patterning. To identify intricate patterns and dependencies in PHP code, other researchers have dived into the field of deep learning and used methods like convolutional neural networks (CNN) [18] or recurrent neural networks (RNN) [22].

They seek to attain high accuracy in differentiating between the two by training these models on large datasets made up of well-known webshells and innocuous PHP code. In certain research, the usage of hybrid modelswhich include both conventional machine learning and deep learning techniques- has also been investigated to maximize each strategy is advantages. These models promise to have improved detection abilities that will allow for a strong defense against changing webshell threats. The work on the machine and deep learning for PHP webshell detection illustrates the continued efforts to create powerful and effective models that can improve the security of web applications.

3.3 The models used

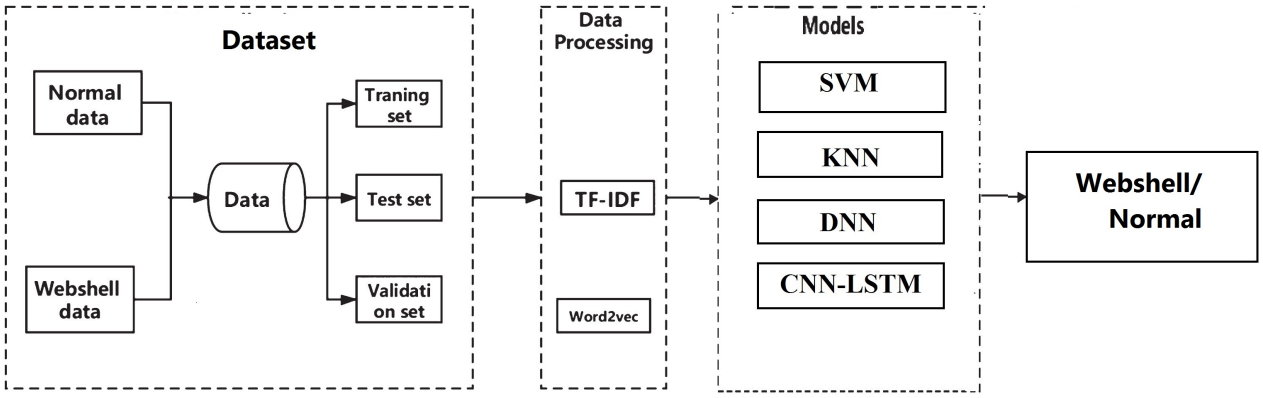

Figure 3.1: The global architecture

- **TF-IDF**

Algorithm 1 provides a general outline of the TF-IDF approach for webshell detection. The specific implementation details, such as the choice of machine learning model and threshold, may vary depending on the dataset and requirements of the detection system.

- **Word2vec** Algorithm 2 is used for learning word embeddings, which are dense vector representations of words in a continuous vector space. These word embeddings capture semantic and syntactic relationships between words and have been widely used in natural language processing (NLP) tasks such as language modeling, machine translation, and text classification.

- **Support Vector Machine** The job of Algorithm 3 is to develop a model that can identify malicious scripts called "webshells" in PHP files.To build an SVM classifier to detect PHP webshells, one starts by collecting a set of representative PHP files, including both files containing webshells and sane files. These files serve as training data for the model.Then, we extract features from PHP files, such as functions used, code patterns, variables, suspicious strings, etc. These characteristics make it possible to represent each PHP file in the form of a digital vector.Then, we train an SVM model using these feature vectors to distinguish files containing webshells from clean files. The SVM model learns to find a decision boundary that best separates the two classes.Once the SVM model is trained, it can be used to predict whether a new PHP file contains a webshell or not. We apply the same feature extraction steps to the file to be tested, then we use the SVM model to predict the class of the file (webshell or healthy).

Algorithm 1 TF-IDF Algorithm

1: **Imports:** `codecs` (for reading the PHP file) and `TfidfVectorizer` (from scikit-learn for TF-IDF conversion).
2: **Input:** PHP file path ('file.php').
3: **Output:** List of unique words extracted from the PHP code and the TF-IDF matrix representation.
4: **procedure** AnalyzePHPFile
5: Import necessary libraries: `codecs` (for reading the PHP file) and `TfidfVectorizer` (from scikit-learn for TF-IDF conversion).
6: Read the contents of the PHP file ('file.php') using `codecs.open` and store it in the variable `php_code`.
7: Create an instance of `TfidfVectorizer` for TF-IDF conversion.
8: Use the `fit_transform` method of the vectorizer to convert `php_code` into a TF-IDF matrix representation.
9: Print the list of words extracted from the text using the `get_feature_names` method of the vectorizer.
10: Print the TF-IDF matrix representation of the PHP code using the `toarray` method of the TF-IDF matrix.

Algorithm 2 Word2Vec Training Algorithm

Imports: `codecs` (for reading files), `CountVectorizer` (from scikit-learn for feature extraction), `Word2Vec` (from gensim.models for training Word2Vec models), `os` (for file-related operations), `re` (for regular expressions), and `numpy` (for numerical operations).
procedure LoadData(`webshell_dir`, `normal_dir`)
    Initialize `data` and `labels` lists.
    List the files in `webshell_dir` and `normal_dir`.
    for each webshell file **do**
        Read the content of the file and append it to `data`.
        Add the label `'webshell'` to `labels`.
    for each normal file **do**
        Read the content of the file and append it to `data`.
        Add the label `'normal'` to `labels`.
    return `data` and `labels`.
procedure ExtractFeatures(`file_content`)
    Apply regular expressions to remove comments and extra whitespaces.
    Find specific features like functions, variables, method calls, and certain function names.
    Join the extracted features into a single string and **return** it.
procedure Word2VecFunc
    `data`, `labels` ← LoadData(`webshell_dir`, `normal_dir`)
    `sentences` ← empty list
    for each `file_content` in `data` **do**
        `features` ← ExtractFeatures(`file_content`)
        Append `features` to `sentences`
    Create a Word2Vec model with `sentences` as input and a minimum count of 1 for word occurrence.
    Print the trained Word2Vec model.
    `vec_select` ← 2
    return `vec_select`
Specify the directories `webshell_dir` and `normal_dir`.
Word2VecFunc()

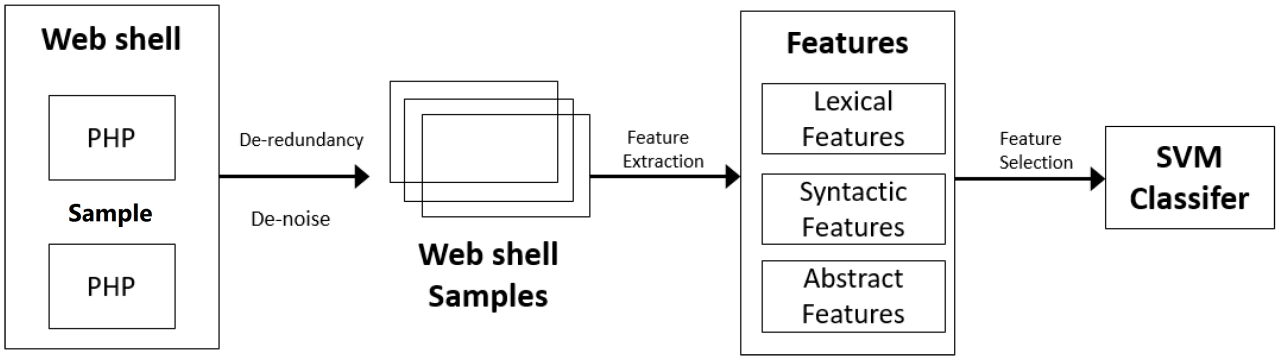

Figure 3.2: SVM model architecture.

• **Deep Neural Network** The job of the DNN Classifier for PHP webshell detection is to develop a deep neural network (DNN) based model to detect webshell files in PHP. Here is a summary of the process:

The DNN Classifier for PHP webshell detection uses the power of deep neural networks to learn the characteristic patterns of webshells and distinguish them from normal files. It is an advanced and accurate approach for automated webshell detection, helping to strengthen web application security.

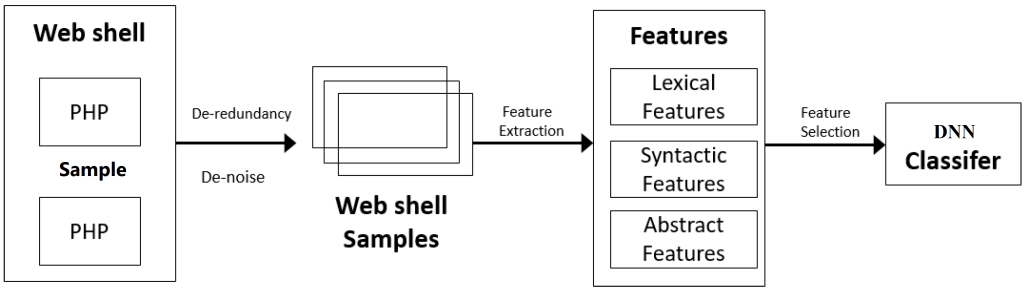

Figure 3.3: DNN model architecture.

• **K-Nearest Neighbors** The work of the KNN (K-Nearest Neighbors) classifier for the detection of webshells in PHP is to develop a model capable of identifying malicious scripts called "webshells" in PHP files based on the closest examples.
To build a KNN classifier to detect PHP webshells, one starts by collecting a set of representative PHP files, including both files containing webshells and sane files. These files will serve as training data for the model.Then, we extract features from PHP files, such as functions used, code patterns, variables, suspicious strings, etc. These characteristics make it possible to

Algorithm 3 SVM Algorithm

function LOAD_DATA(webshell_dir, normal_dir):
    Input: webshell_dir , normal_dir
    Output: data (list of file contents), labels (list of corresponding labels)
    Read file contents from webshell_dir and append to data list with label 'webshell'
    Read file contents from normal_dir and append to data list with label 'normal'
    Return data and labels
function EXTRACT_FEATURES(file_content):
    Input: file_content (content of a file)
    Output: features (extracted features as a string)
    Remove comments and excess whitespace from file_content
    Extract features using regular expressions and store in features
    Return features
function PREDICT_WEBSHELL(model, vectorizer, encoder, file_path):
    Input: model , vectorizer (TF-IDF vectorizer), encoder (label encoder), file_path
    Output: label (predicted label for the file)
    Read file content from file_path
    Extract features from the file content using extract_features
    Transform the extracted features using vectorizer.transform
    Predict the label using the model.predict method
    Inverse transform the predicted label using encoder.inverse_transform
    Return the predicted label
Set webshell_dir and normal_dir to the appropriate directory paths.
Load data using LOAD_DATA(webshell_dir, normal_dir) and assign the returned values to data and labels.
Extract features from data using EXTRACT_FEATURES(data) and store the result in data.
Initialize TfidfVectorizer as vectorizer.
Transform data using vectorizer.fit_transform and assign the result to X.
Initialize LabelEncoder as encoder.
Encode labels using encoder.fit_transform and assign the result to y.
Split the data into training and testing sets using train_test_split and assign the results to X_train, X_test, y_train, y_test.
Initialize the Support Vector Machine (SVM) model with linear kernel as model.
Train the model using model.fit with X_train and y_train.
Print "SVM Model".
Calculate accuracy using model.score on X_test and y_test and print the result.
Predict labels for X_test using model.predict and assign the result to y_pred.
Calculate accuracy, precision, recall, and F1-score using the respective metrics functions and print the results.
Classify a new file by calling PREDICT_WEBSHELL(model, vectorizer, encoder, file_path) with the model, vectorizer, encoder, and file path as arguments, and print the predicted label.
Repeat the previous step for another file.

Algorithm 4 DNN Algorithm

procedure DataCollection

PHP files, including webshells and normal files, are collected to form a training dataset.

procedure FeatureExtraction

Analyze the contents of PHP files and extract significant characteristics, such as function names, variables, and suspicious operations.

Eliminate comments and unnecessary spaces to obtain clean textual data.

procedure DataPreprocessing

Convert textual data into digital vectors using vectorization techniques (e.g., TF-IDF representation) to make them compatible with the DNN model.

procedure DNNConstruction

Build a deep neural network (DNN) using layers of neurons, such as dense layers and activation layers.

Train the model using the training data by adjusting the weights of the connections between neurons.

procedure ModelEvaluation

Evaluate the performance of the model using metrics such as accuracy, precision, recall, and F1 score.

These metrics measure the effectiveness of the model in detecting PHP webshells.

procedure WebshellDetection(unknown_file)

Once the model is trained and evaluated, use it to detect webshells in new PHP files.

Extract features from the unknown PHP file.

Predict whether the file is a webshell or a normal file using the trained model.

represent each PHP file in the form of a digital vector.

The KNN algorithm works by calculating the similarity between unknown files and training examples. To predict whether a new PHP file contains a webshell, one calculates the distance between that file and the training examples using an appropriate similarity measure, such as the Euclidean distance.Then, we select the K training examples closest to the unknown file in terms of distance. We use the labels of the K nearest neighbors to predict the class of the unknown file (webshell or sane). For example, if the majority of the K nearest neighbors are labeled as webshells, one can predict that the unknown file is also a webshell.

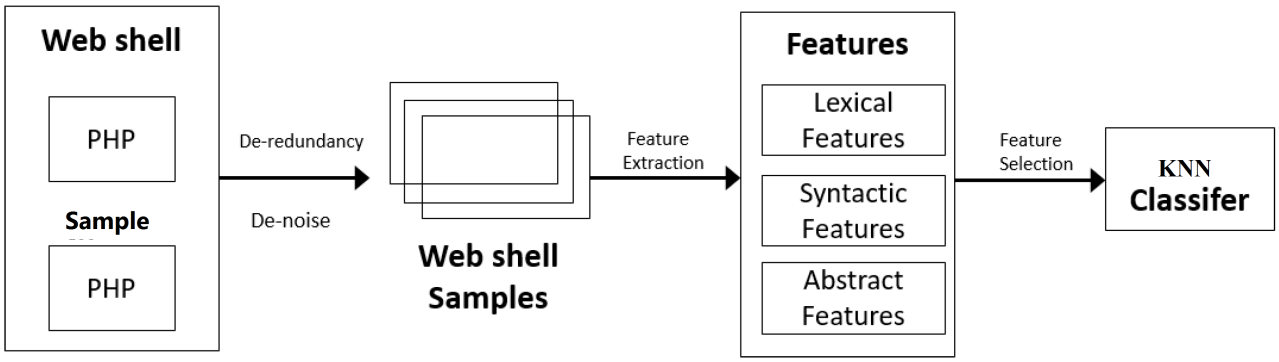

Figure 3.4: KNN model architecture.

• **CNN-LSTM** Algorithm 6 is a hybrid neural network architecture that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This architecture is commonly used for sequence data analysis, such as text or time series data, where both local and temporal dependencies need to be captured.

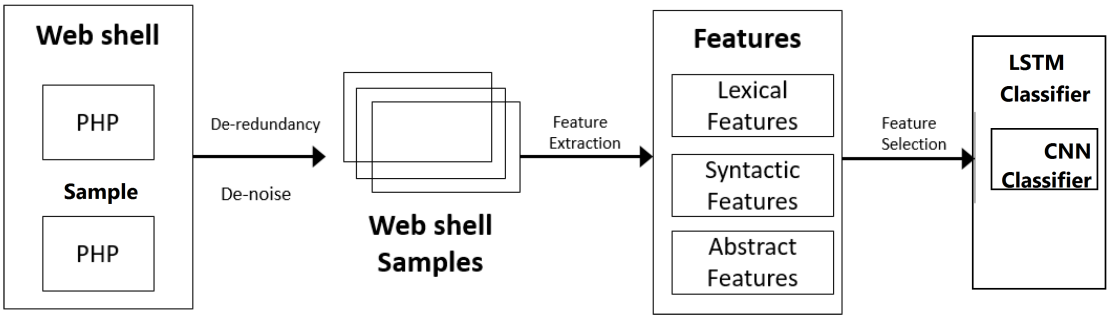

Figure 3.5: CNN-LSTM model architecture.

Algorithm 5 KNN Algorithm

procedure LoadData(webshell_dir, normal_dir)
    Initialize directories for webshell files (`webshell_dir`) and normal files (`normal_dir`)
    Call load_data with `webshell_dir` and `normal_dir` to load file contents and labels
    Store file contents in list `data` and labels in list `labels`
procedure ExtractFeatures(data)
    for each file content in `data` **do**
        Remove comments and extra whitespace using regular expressions
        Extract relevant features using regular expressions
        Join extracted features into a single string separated by spaces
        Store feature string in corresponding position in `data`
procedure VectorizeData(data)
    Create an instance of `TfidfVectorizer` with max features set to 10,000
    Use vectorizer to transform `data` into TF-IDF representation
    Convert transformed data to dense array using `toarray()` and store in variable `X`
procedure PrepareLabels(labels)
    Convert `labels` to NumPy array and store in variable `y`
procedure SplitData(X, y)
    Split `X` and `y` into training and testing sets using `train_test_split()`
    Set test size to 0.2 and random state to 42
    Store training and testing data and labels in variables `X_train`, `X_test`, `y_train`, `y_test`
procedure TrainKNNModel(X_train, y_train)
    Set number of neighbors (`k`) to 5
    Create instance of `KNeighborsClassifier` with `n_neighbors=k` and store in variable `model`
    Fit model to training data and labels using `fit()` method
procedure PredictAndEvaluate(model, X_test, y_test)
    Use trained model to predict labels for `X_test` and store predictions in `y_pred`
    Calculate accuracy, precision, recall, and F1-score using corresponding metrics
    Print evaluation metrics
procedure ClassifyNewFiles(file_paths, model)
    for each file in `file_paths` **do**
        Extract features from file content using ExtractFeatures
        Transform extracted features into TF-IDF representation using vectorizer
        Predict label for file using trained model and print result
Specify directories for webshell and normal files
LoadData(webshell_dir, normal_dir)
ExtractFeatures(data)
VectorizeData(data)
PrepareLabels(labels)
SplitData(X, y)
TrainKNNModel(X_train, y_train)
PredictAndEvaluate(model, X_test, y_test)
Specify paths of new files to be classified
ClassifyNewFiles(file_paths, model)

Algorithm 6 CNN-LSTM Algorithm

Require:

Import the necessary libraries `TfidfVectorizer`, `LabelEncoder`, TensorFlow.keras modules, and evaluation metrics from scikit-learn.
procedure LOAD_DATA(webshell_dir, normal_dir)
    data ← []
    labels ← []
    List files in `webshell_dir`
    for each webshell file **do**
        Read file content and append to *data*
        Append label 'webshell' to *labels*
    List files in `normal_dir`
    for each normal file **do**
        Read file content and append to *data*
        Append label 'normal' to *labels*
    return *data*, *labels*
procedure EXTRACT_FEATURES(file_content)
    Remove comments and extra whitespaces using regular expressions
    Extract specific features using regular expressions
    Join extracted features into a single string
    return extracted features
procedure CREATE_MODEL
    Create a deep learning model using TensorFlow.keras Sequential API
    Compile the model with categorical cross-entropy loss, Adam optimizer, and accuracy metric
    return the model
procedure PREDICT_WEBSHELL(file, model, vectorizer, encoder)
    Read file content
    Extract features from file content using EXTRACT_FEATURES
    Transform extracted features using the vectorizer
    Predict label using the model
    Inverse transform predicted label using the encoder
    return predicted label
Specify webshell and normal file directories
data, *labels* ← LOAD_DATA(*webshell_dir*, *normal_dir*)
for each file content in *data* **do**
    Apply EXTRACT_FEATURES to file content
Create a `TfidfVectorizer` with maximum 10,000 features and transform the data
Encode labels using `LabelEncoder` and convert to one-hot encoded form
Split data and labels into training and testing sets using `train_test_split`
Get input shape for the model from the shape of the training data
Create the model using CREATE_MODEL
Fit the model to the training data
Evaluate the model on the testing data and print the accuracy
Predict labels for the testing data and calculate evaluation metrics
Predict label for a new file using PREDICT_WEBSHELL and print the predicted label

3.4 Hardware and Software

3.4.1 Hardware environment (Hardware)

We present the detailed experiments and evaluation steps to test the effectiveness of our model. All experiments were performed on an HP PC equipped an Intel(R) Celeron(R) CPU N3060 @ 1.60GHz 1.60 GHz processor and 8 GB RAM and an Intel(R)HD Graphics GPU.

3.4.2 Software environment (Software):

- Python

  Is an interpreted high-level programming language for programming for general use. Created by Guido van Rossum, and first published in 1991. It is based on a design philosophy that emphasizes the readability of the code, in particular by using meaningful spaces. It provides constructs for programming clear at small and large scale. Python offers a dynamic type system and management automatic memory. It supports multiple programming paradigms,including object-oriented, imperative, functional and procedural, and has a library extended and complete standard. Python is an open-source, high-level programming language, developed for use with a wide range of operating systems
  It is called as the most powerful programming language due to its dynamic nature and diversified. Python is easy to use with a super simple syntax that is very encouraging to beginner learners, and very motivating for seasoned users. contains several libraries popular artificial intelligence: (Keras, Tensorflow, Numpy, Pandas, OPENcv, Pytorch, etc.. . .)[23].

- Tensorflow

  Is an end-to-end open source platform for building applications machine learning. It is a symbolic mathematical library that usesa flow of data and differentiable programming to perform various tasks focusedon the training and inference of deep neural networks. It allows developers to create machine learning applications using various tools, libraries andcommunity resources. Currently, the most famous deep learning library in the world is TensorFlow from Google. Google uses machine learning in all of its products tosearch engine optimization, translation, image captions or recommendations. TensorFlow offers multiple levels of abstraction so you can choose the right one best for your needs. Build

and train models using the high-level Keras API, making it easy to get started with TensorFlow and machine learning[24].

- Keras

  is a high-level neural network API, written in Python and capable of run on TensorFlow or Theano. It was developed with a focus on experimentation fast. Being able to go from idea to result with the least possible delay is the key to good research. It was developed as part of the research effort of theONEIROS project (Openended Neuro-Electronic Intelligent Robot Operating System), and its main author and maintainer is François Chollet, a Google engineer. In 2017, the team Google is TensorFlow has decided to support Keras in the core Tensor- Flow library. Chollet explained that Keras was designed as an interface rather than end-to-end learning framework. It presents a set of higher level abstractions and more intuitive facilitate the configuration of neural networks independent of the library backend computing. Microsoft is also working on adding a CNTK backend in Keras too[25].

- NumPy

  Is a fundamental package for scientific computing with Python. It provides powerful and efficient tools for handling large, multi-dimensional arrays and matrices, along with a wide range of mathematical functions to operate on these arrays. NumPy serves as a foundation for many other scientific libraries in Python and is widely used in various fields, including data analysis, machine learning, and numerical simulations. The bibliography of NumPy in Python refers to a collection of authoritative references, such as research papers, textbooks, documentation, and articles, that provide detailed information, explanations, and examples on the usage, features, and underlying algorithms of NumPy. These resources serve as essential references for developers, researchers, and learners interested in effectively utilizing NumPy is capabilities within the Python programming language[26].

- Pandas

  Is a robust and well-liked open-source Python data manipulation and analysis toolkit. It offers adaptable data structures that let users manage and analyze structured data effectively, like DataFrame and Series. For activities including data cleansing, filtering, grouping, merging, reshaping, and visualization, Pandas provides a large number of

methods and functions. The list of authoritative sources in the bibliography for Pandas in Python includes academic papers, books, government documents, online tutorials, and articles that provide in-depth explanations of the features, capabilities, and best practices of Pandas. These sources are helpful tools for Python data workers who want to learn more about Pandas and how to use its robust capabilities to glean insightful information from datasets[27].

- Sklearn

  A well-known machine learning library for Python is called scikit-learn, also known as sklearn. It offers a wide variety of tools and methods for jobs like model selection, regression, clustering, and dimensionality reduction. A user-friendly interface for developing and testing machine learning models is provided by scikit-learn, which is built on top of existing scientific Python tools like NumPy and SciPy. It refers to a collection of reliable sources that explore the ideas, algorithms, and usage patterns related to scikit-learn, including academic papers, textbooks, government documents, tutorials, and articles. These publications offer useful insights into the theory of machine learning, real-world implementation strategies, and best practices for using scikit-learn successfully. For developers, academics, and practitioners wishing to use scikit-learn to create accurate and reliable machine learning models in Python, the bibliography is an invaluable resource[28].

3.5 Evaluation scheme

3.5.1 Dataset

This dataset was collected from publicly accessible Github repositories. It contains an appropriate mix of PHP innocuous files and webshells. They are initially cleaned up using MD5, where duplicates are two files with the identical MD5 hashes. After that, the remaining files opcode sequences are formed, their sequences are subjected to MD5, and any duplicates of their opcode sequences are eliminated. This might offer more different samples that can be utilized to fairly assess automatic webshell detectors. There are 992 samples in total for each category in the final dataset [29].

3.5.2 Preprocessing

- **Extraction data with TF-IDF**

is a metric for determining how important a term is inside a document or group of papers. It combines the inverse document frequency (IDF) and term frequency (TF) variables.which are:
TF = (Number of occurrences of the term in the document) / (Total number of terms in the document)
IDF = log((Total number of documents) / (Number of documents containing the term))
$TF - IDF = TF * IDF$

Figure 3.6: Dataset window of the application.

- **Extraction data with word2vec**

is a word embedding paradigm that shows words as continuous vector spaces with dense vector representations of the words. A neural network design, such as continuous bag-of-words (CBOW) or skip-gram, is used to train it. By modifying the word vectors to minimize a selected loss function during training, the model develops the ability to correlate words with their context or target words. The Word2Vec model generates word vectors after training that represent the syntactic and semantic links between words. These vectors can be utilized for operations like determining word similarity, determining nearest neighbors, determining word relationships, and initializing further NLP models.

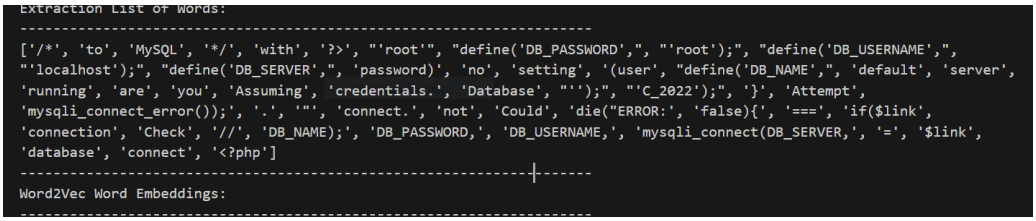

Figure 3.7: Dataset window of the application.

3.5.3 Evaluation metrics

We use 10-fold cross-validation [30] and well-known metrics to assess the proposed model. False Positive (FP), False Negative (FN), True Positive (TP), and True Negative (TN) numbers are the basic units used to measure these measures. TP stands for the quantity of correctly predicted webshell files, whereas TN stands for the quantity of correctly predicted benign files. False Negative FN denotes the number of false positives for benign files, whereas FP denotes the number of inaccurate predictions for webshell files. The approved list of evaluation metrics is as follows:

1. Recall: ratio of webshell samples that were accurately predicted.Utilizing the following formulas, it is calculated [30].

$Recall = \frac{TP}{(TP+FN)}$.

2. Precision: the proportion of webshell samples that were accurately predicted among samples that were positively predicted. This formula is used to calculate it.

$Precision = \frac{TP}{TP+FP}$.

3. Accuracy: the proportion of correctly predicted samples to all samples. The following formula is used to calculate a model is accuracy:

Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$.

3. Macro F1-score: the average F1 score for each class. By averaging the F1-scores calculated for each class i, we may obtain the macro F1-score. Macro F1-score is calculated using the formula below:

$MacroF1 - score = \frac{1}{N}\sum_{i=0}^{N} F1 - score_i$.

F1-score indicates the optimal balance between precision and class recall for each class. This formula is used to compute it:

$F1 - scorei = 2 * \frac{Precision_i * Recall_i}{precision_i + Recall_i}$.

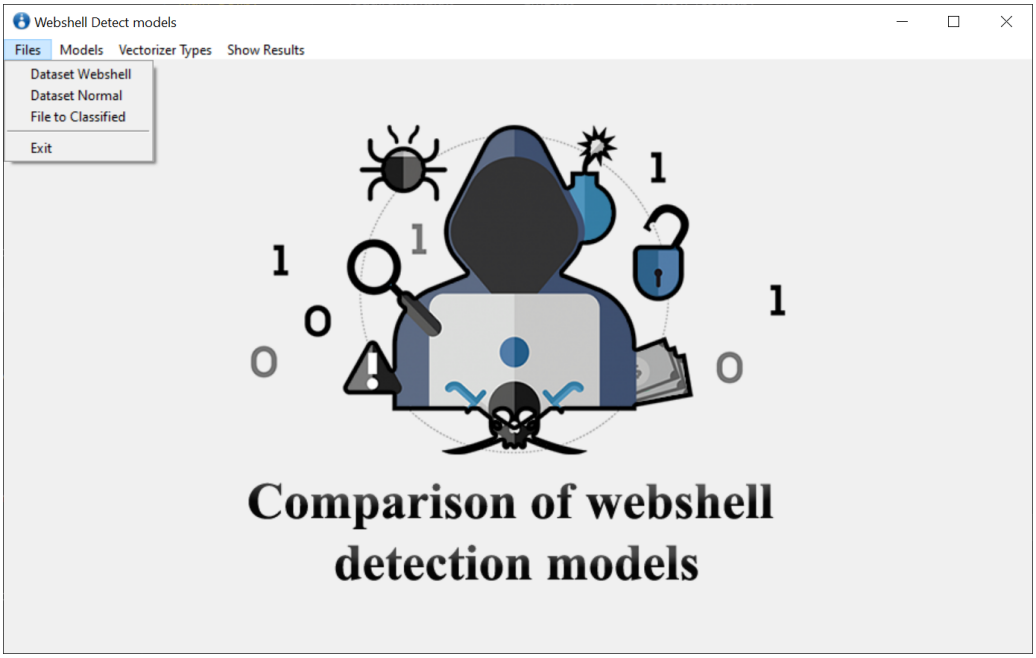

Figure 3.8: Dataset window of the application.

Figure 3.8 shows the application window that consists of three distinct sections: the webshell display, the normal dataset display, and the to-be-classified dataset display.

The webshell display section provides a user interface for interacting with a webshell. A webshell is a script or program that enables remote administration and control of a web server. It allows the user to execute commands and perform various actions on the server through a command-line interface. The webshell display typically includes a command prompt or input area where users can enter commands and an output area where the results of executed commands are displayed.

The normal dataset display section presents a visual representation or listing of the normal datasets. These datasets contain information that is known and considered safe or non-malicious. The display may include features such as a table or grid view, where each dataset is represented by rows and columns of data or graphical representations such as charts or graphs.

The to-be-classified dataset display section showcases datasets that require classification or analysis. These datasets are yet to be categorized or labeled as normal or potentially malicious. Similar to the normal dataset display, this section can also include a table or grid view, graphical representations, or any other visual representation that facilitates the analysis or classification process.

By organizing the application window into these three sections, users can efficiently manage and analyze data within the webshell environment, compare normal and potentially malicious

datasets, and perform necessary actions for classification or further investigation.

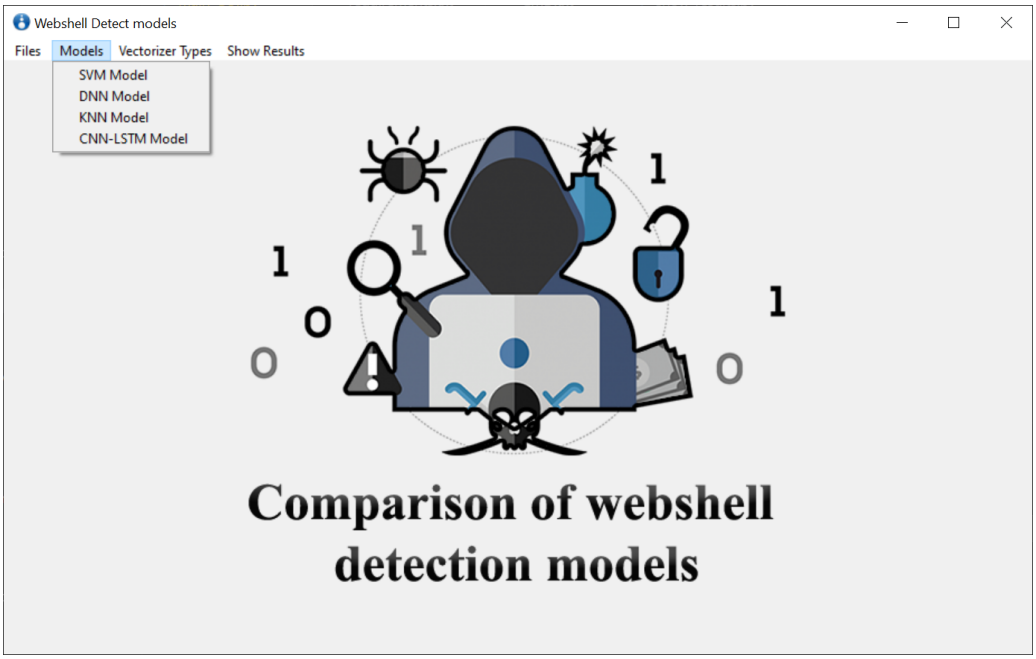

Figure 3.9: Menu of the used models.

Figure 3.9 shows the menu window which is designed to showcase the models used for training, including SVM, DNN, KNN, and CNN-LSTM.

The window typically includes separate sections or tabs for each model, allowing users to access and review the details of each model individually. Within each section, users can find relevant information and visualizations associated with the respective model.

For the SVM (Support Vector Machine) model, the window might display details such as the kernel type used (e.g., linear, polynomial, radial basis function), hyperparameter settings, accuracy metrics, and possibly a graphical representation of the decision boundary or support vectors.

The DNN (Deep Neural Network) section would provide information about the architecture of the neural network, including the number and type of layers (e.g., convolutional, dense), activation functions, optimizer choices, training parameters, and performance evaluation metrics like loss and accuracy. Additionally, visualizations such as the model architecture diagram or training/validation curves may be included.

In the KNN (K-Nearest Neighbors) section, the window would present the value of K chosen, any distance metrics employed (e.g., Euclidean, Manhattan), feature scaling or preprocessing techniques, and evaluation metrics like accuracy or precision-recall curves. It might also include a visualization of the decision boundaries created by the KNN algorithm.

Lastly, for the CNN-LSTM (Convolutional Neural Network - Long Short-Term Memory) model, the window would provide details about the combined architecture of convolutional and LSTM layers, the number of filters and kernel sizes used in the convolutional layers, LSTM cell configurations, activation functions, and optimizer choices. Similar to the DNN section, visualizations such as model architecture diagrams or training/validation curves could be included.

By organizing the window into sections for each model, users can easily navigate and access information related to the specific models used for training. This helps in understanding the model configurations, evaluating their performance, and making informed decisions in the context of the trained models.

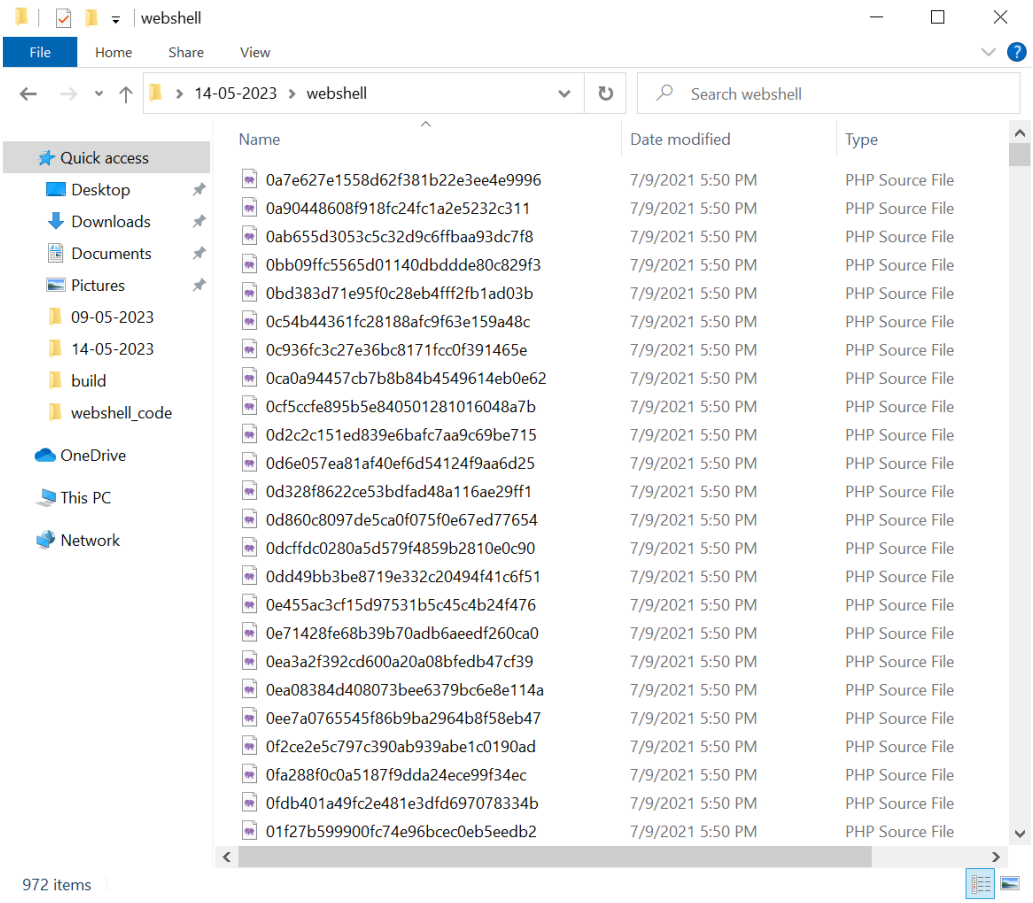

Figure 3.10: Pattern of the used webshell dataset.

Figure 3.10 depicts a pattern of the webshell dataset used during the training process. The figure provides a visual representation of the dataset is characteristics and reveals insights into the underlying patterns and trends within the webshell data.

In the figure, various elements are present to showcase the dataset pattern effectively. It might include a graph or chart, such as a line plot or histogram, representing the distribution of different features or variables within the webshell dataset. The x-axis of the graph would

typically represent the dataset instances or time intervals, while the y-axis represents the values or frequencies of the corresponding feature.

The figure may exhibit distinctive patterns that can be observed through the visualization. These patterns could include peaks or spikes, indicating occurrences of certain events or anomalies within the webshell data. Additionally, there may be regions of stability or fluctuations, providing insights into the stability or volatility of specific features over time or dataset instances.

Furthermore, the figure might include annotations or labels to provide additional information or context about the dataset pattern. These annotations could highlight important observations or specific data points of interest, aiding in the understanding and interpreting the pattern presented.

The illustrated figure serves as a valuable visual representation of the webshell dataset pattern, allowing researchers, analysts, or users to gain insights into the behavior, trends, and characteristics of the webshell data during the training process. It provides a clear and concise summary of the dataset's properties, enabling informed decision-making and further analysis based on the identified patterns.

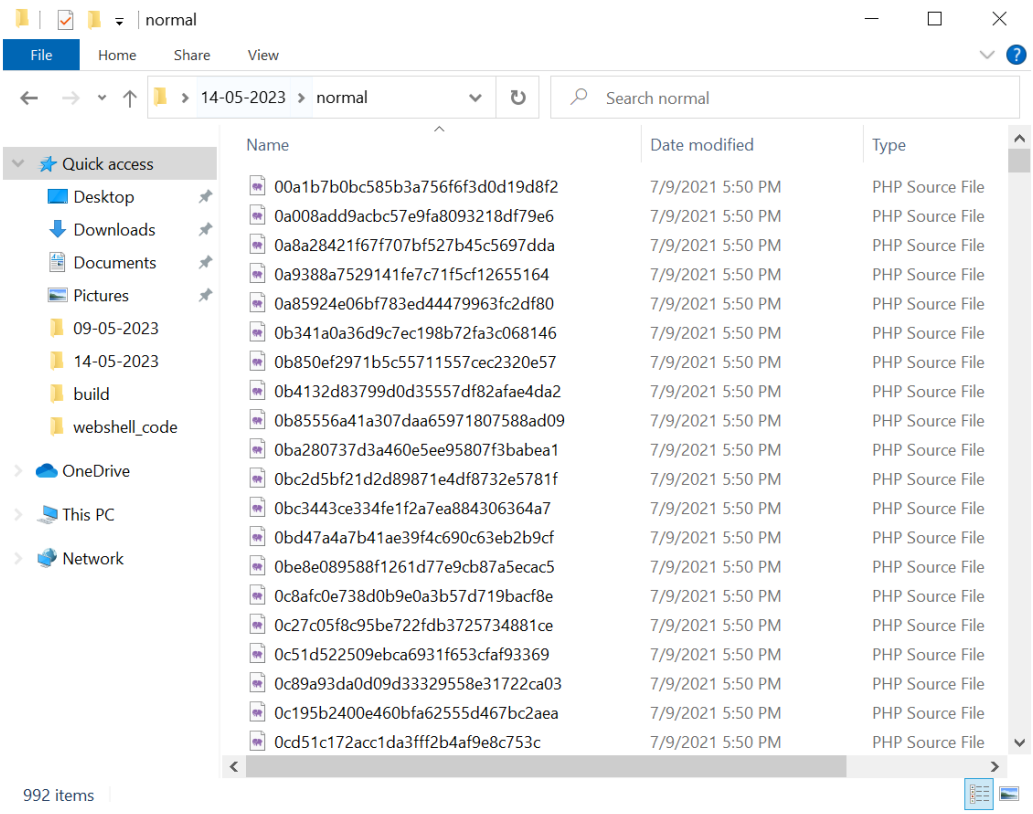

Figure 3.11: Pattern of the used normal dataset.

Figure 3.11 portrays the pattern observed within the normal dataset employed during the training phase. The figure offers a visual representation that enables the exploration of the dataset's inherent characteristics and discernment of underlying patterns and trends within the normal data.

To effectively showcase the normal dataset pattern, the figure may include graphical elements such as line plots, histograms, or scatter plots. These visualizations provide insights into the distribution, relationships, or variations of different features or variables within the normal dataset. The x-axis typically represents dataset instances or time intervals, while the y-axis represents the corresponding feature values or frequencies.

Within the figure, discernible patterns emerge, illuminating the behavior of the normal dataset. These patterns can manifest as consistent trends, cyclical variations, or distinctive clusters. They enable the identification of regularities and anomalies, facilitating a comprehensive understanding of the normal data's structure.

Moreover, the figure may incorporate annotations or labels to provide supplementary information and context about the observed dataset pattern. These annotations could highlight noteworthy observations and crucial data points or provide explanations for certain patterns, enhancing the interpretability of the visual representation.

The illustrated figure is a valuable tool for comprehending the pattern inherent in the normal dataset used for training. By visually capturing the dataset's characteristics, it empowers researchers, analysts, or users to gain insights into the normal data is dynamics, trends, and relationships. This understanding facilitates informed decision-making and further analysis based on the identified patterns within the normal dataset.

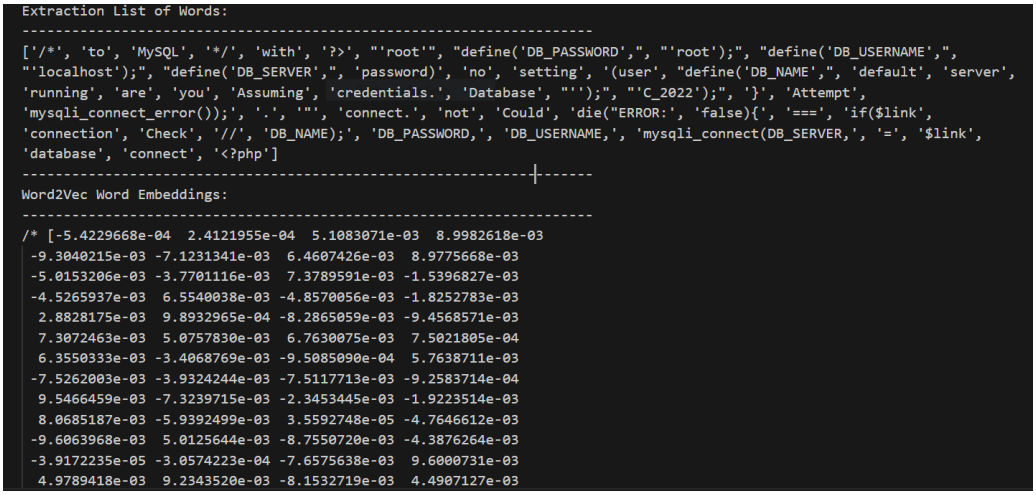

Figure 3.12: Pattern of word to vector process.

The Word to Vector (word2vec) process refers to a popular technique used to convert words or textual data into numerical vectors in natural language processing (NLP). It is a shallow neural network-based approach that learns distributed representations, also known as word embeddings, for words in a large corpus of text. Figure 3.15 depicts the pattern observed during the Word2Vec process, showcasing the transformation of words into vector representations. The figure provides a visual representation of the underlying steps and relationships involved in converting words into meaningful numerical embeddings.

In the figure, several key components are displayed to illustrate the Word2Vec process effectively. It may include a flowchart or a series of interconnected boxes and arrows to denote the sequential steps involved. The figure may also incorporate additional graphical elements, such as word clouds or scatter plots, to highlight the distribution or relationships between the word vectors.

The process begins with a large corpus of text, which serves as the input for the Word2Vec algorithm. The figure may indicate this corpus through a box labeled "Text Corpus" or a similar representation. From there, the Word2Vec algorithm applies either the CBOW or Skip-gram model to learn the distributed representations of words.

The figure might depict the training phase, showing how the algorithm processes the text corpus and updates the weights in the neural network. This could be represented by arrows connecting the input layer, hidden layer, and output layer, illustrating the flow of information and the learning process.

Once the training is complete, the figure may display a transition from the training phase to the embedding phase. It could represent this transition by showcasing the word embeddings as numerical vectors associated with each word in the vocabulary. These vectors are typically high-dimensional and capture semantic relationships between words.

To represent the pattern of the Word2Vec process, the figure might include visualizations of the word vectors in a reduced-dimensional space. This could be achieved through techniques like principal component analysis (PCA) or t-SNE (t-Distributed Stochastic Neighbor Embedding). The reduced-dimensional plot would allow for the visualization of clusters or patterns formed by semantically similar words, facilitating the understanding of the learned word embeddings.

Additionally, the figure may incorporate annotations or labels to provide additional information and context about the Word2Vec process. These annotations could explain the specific steps, algorithms, or parameters utilized, aiding in the interpretation of the figure.

Overall, the illustrated figure serves as a visual representation of the Word2Vec process, capturing the transformation of words into numerical vectors. It helps researchers, analysts, or

users understand the steps involved in generating word embeddings and provides insights into the patterns and relationships encoded within the learned word vectors.

```
Extraction List words
----------------------------------------------------------------------
['php', 'database', 'credentials', 'assuming', 'you', 'are', 'running', 'mysql', 'server', 'with', 'default', 'setting',
'user', 'root', 'no', 'password', 'define', 'db_server', 'localhost', 'db_username', 'db_password', 'db_name', 'c_2022',
'attempt', 'to', 'connect', 'link', 'mysqli_connect', 'check', 'connection', 'if', 'false', 'die', 'error', 'could',
'not', 'mysqli_connect_error']
Matrix TF-IDF:
----------------------------------------------------------------------
[[0.10846523 0.10846523 0.10846523 0.10846523 0.10846523 0.21693046
  0.10846523 0.10846523 0.10846523 0.21693046 0.21693046 0.21693046
  0.21693046 0.21693046 0.10846523 0.43386092 0.10846523 0.10846523
  0.10846523 0.10846523 0.21693046 0.10846523 0.21693046 0.10846523
  0.10846523 0.10846523 0.10846523 0.10846523 0.10846523 0.21693046
  0.10846523 0.10846523 0.10846523 0.21693046 0.10846523 0.21693046
  0.10846523]]
```

Figure 3.13: Pattern of TF_IDF process.

The TF-IDF (Term Frequency-Inverse Document Frequency) process is a widely used technique in information retrieval and text mining to determine the importance of a term within a document or a collection of documents. It aims to highlight terms that are both frequent within a document and relatively rare across the entire document collection. Figure 3.13 illustrates the pattern observed during the TF-IDF (Term Frequency-Inverse Document Frequency) process, depicting the calculation and utilization of TF-IDF scores for terms within a document collection. The figure provides a visual representation of the steps and relationships involved in determining the importance of terms in a text corpus. In the figure, various elements are present to showcase the TF-IDF process effectively. It may include boxes, arrows, and numerical values to denote the different stages and calculations.

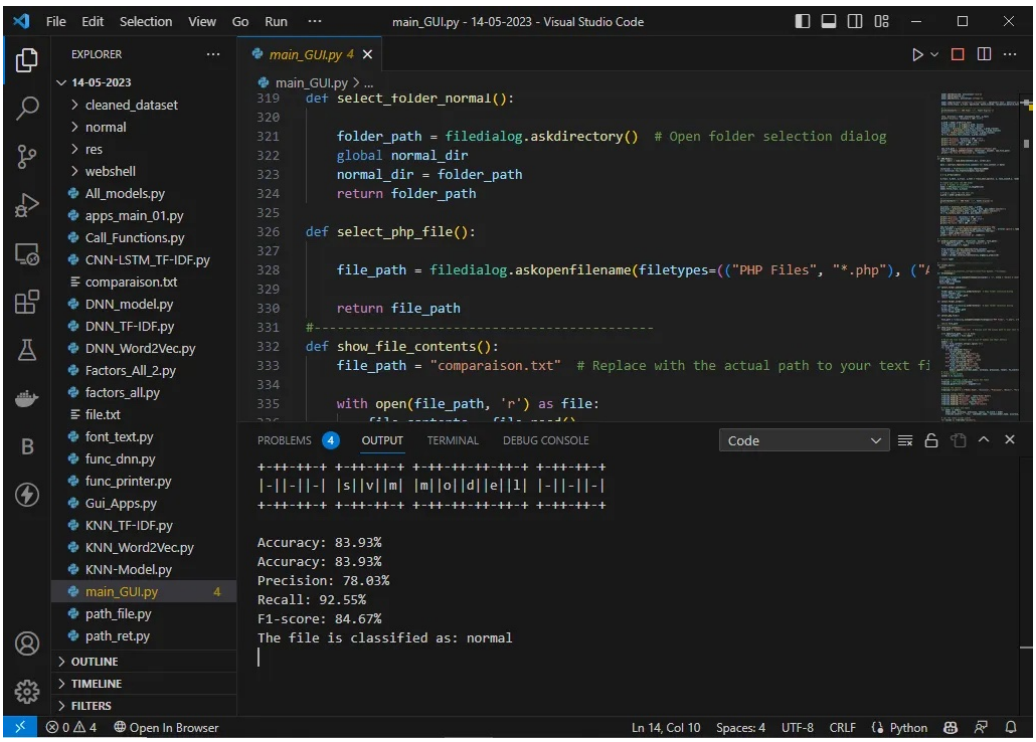

Figure 3.14: The training results using SVM model.

Figure 3.14 shows the results of the SVM model, where:

Accuracy: 85.75% Accuracy: 80.59% Recall: 95.02% F1 score: 87.21% When analyzing the source code file, according to the SVM model, it is classified as "normal" or ordinary source code. This means the model has concluded that the file is not a WebShell based on the code and the prediction information.

The results shown can be summarized. However, the SVM model shows good performance with an accuracy of 85.75%, a classification accuracy of 80.59%, a recall (ability to detect actual cases of Web Shell) of 95 .02%, and an F1 score of 87.21%. These values indicate the model is ability to differentiate and classify files accurately.

Based on this analysis, we can trust the classification of the SVM model and conclude that the mentioned source code file is an ordinary source code and not a WebShell.

The results of Figure 3.1 represent the evaluation of a set of models that have been trained and tested on web shell detection files or normal files. The results show how well each model classified files based on different metrics.

"Precision" (Accuracy) represents the percentage of correct cases (correctly classified files) compared to the total number of cases. Accuracy measures the general ability of the model to distinguish normal files from web shell files.

"Precision" is the ratio of correct cases divided by the total number of cases classified posi-

Table 3.1: Comparaison between models

Model Name	**Accuracy**	**Precision**	**Recall**	**F1-score**
SVM_Model	68.19%	61.73%	99.50%	76.19%
DNN_Model	90.20%	71.44%	92.23%	66.34%
KNN_Model	98.75%	94.55%	93.33%	91.34%
CNN-LSTM_MODEL	92.55%	88.54%	90.30%	93.28%

tively by the model. It measures the model's accuracy in correctly distinguishing normal files from web shell files.

The "recall" is the ratio of correct cases divided by the total number of truly positive cases among the files. It measures the ability to detect all real web shell file cases correctly.

The "F1-score" (F1-score) is a balanced measure considering precision and recall. It measures the model is ability to distinguish normal files from web shell files globally.

According to the given results, it seems that the DNN_Model model offers the best overall performance, with a high percentage of accuracy and high values for precision, recall, and F1 score. On the other hand, the KNN_Model model seems to have lower performance, with a low percentage of accuracy and low values for accuracy, but it has a high recall of 100%, which means that it can detect all real cases of files web shell, but it can also classify some normal files incorrectly.

CONCLUSION AND FUTURE WORK

In conclusion, the field of webshell detection techniques has seen significant advancements with the emergence of both machine learning (ML) and deep learning (DL) approaches. These techniques have proved to be effective in identifying and mitigating the threats posed by web shells, which are malicious scripts used by attackers to gain unauthorized access and control over web servers. ML-based webshell detection techniques leverage various algorithms and features to build models that can classify web shells accurately. These approaches have the advantage of detecting known and unknown web shells based on patterns and behavioral analysis. However, they often require large amounts of labeled training data and may face challenges in handling the evolving nature of webshell attacks.

In future studies, efforts will focus on detecting and analyzing Webshell files, as well as converting them into an opcode-like representation resembling executable files. The following steps will be followed to achieve this goal:

1. Conversion of source files and codes to opcode: Techniques will be explored and developed to convert PHP files and other source files to an opcode-like representation. This will allow codes to be analyzed and understood in a manner similar to executable files, making it easier to extract important features and information.

2. Use of text analysis techniques: Techniques such as TF-IDF and Word2Vec will be applied to extract distinctive features and attributes from the opcode representation. These characteristics represent the very essence of the opcode and allow us to detect suspicious behavior or files with security vulnerabilities.

3. Analysis and circumvention of encryption techniques: Analysis and understanding of encryption techniques used in Webshell files is essential. Different techniques, such as Base64 and Unicode, will be investigated, and tools and methodologies will be developed to deal with these

ciphers and circumvent them effectively.

Additionally, other methods can be used to extract features and test them for better results. It is possible to use other models or to merge two different models to improve performance. Increasing the dataset can also be done by introducing more samples. The results obtained from these different approaches can then be compared to assess their effectiveness and determine the best method to adopt. This exploration of different techniques and approaches in future studies will help improve the accuracy and reliability of Webshell file detection and strengthen the security mechanisms in place.

In this part of our work, we converted PHP files from a dataset including normal and Webshell files to opcode. This allowed us to build a new dataset composed of opcodes. Then we followed the general scheme of our work.

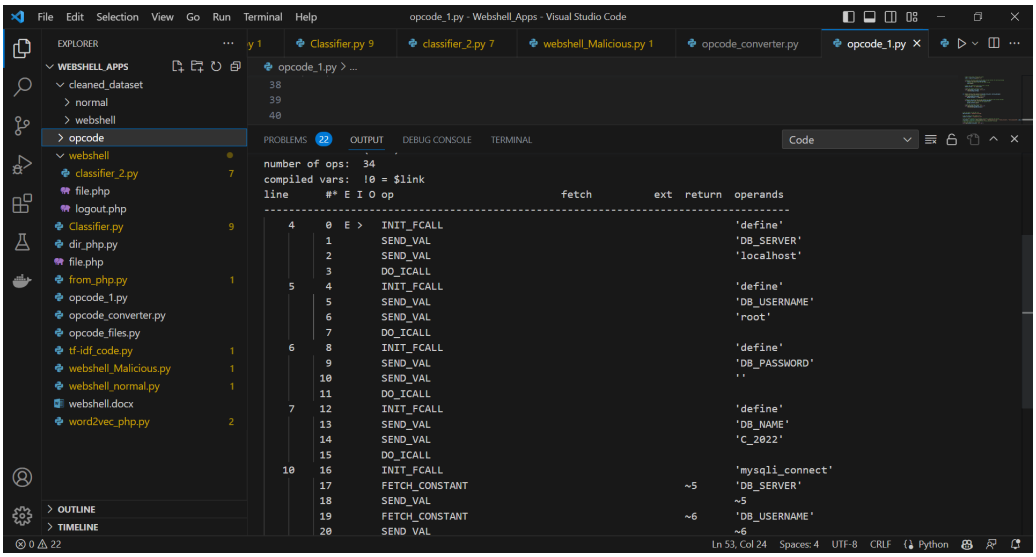

Figure 3.15: The process of converting a php file to an opcode.

In future work, efforts will be focused on converting webshell files to a representation similar to opcode and executable files. Text analysis techniques such as TF-IDF and Word2Vec will be used to extract the distinctive characteristics of the opcode, thus enabling the analysis and detection of suspicious files. The analysis and understanding of encryption techniques used in Webshell files will also be studied and dealt with effectively.

BIBLIOGRAPHY

[1] M. Yousefi kia, M. Saniei, and S. G. Seifossadat, "A novel cyber-attack modelling and detection in overcurrent protection relays based on wavelet signature analysis," *IET Generation, Transmission & Distribution*, 2023.

[2] J. Zhao, Y. Lu, X. Wang, K. Zhu, and L. Yu, "Wta: a static taint analysis framework for php webshell," *Applied Sciences*, vol. 11, no. 16, p. 7763, 2021.

[3] 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*, 2018, pp. 52–56.

[4] S. T. Z. Xuan and V. Selvarajah, "Web shell attack and mitigation," in *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*. IEEE, 2022, pp. 1–5.

[5] J. Petit and S. E. Shladover, "Potential cyberattacks on automated vehicles," *IEEE Transactions on Intelligent transportation systems*, vol. 16, no. 2, pp. 546–556, 2014.

[6] J. M. Biju, N. Gopal, and A. J. Prakash, "Cyber attacks and its different types," *International Research Journal of Engineering and Technology*, vol. 6, no. 3, pp. 4849–4852, 2019.

[7] J. Coe and M. Atay, "Evaluating impact of race in facial recognition across machine learning and deep learning algorithms," *Computers*, vol. 10, no. 9, p. 113, 2021.

[8] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[9] H. Kang, S. J. Yoo, and D. Han, "Senti-lexicon and improved naïve bayes algorithms for sentiment analysis of restaurant reviews," *Expert Systems with Applications*, vol. 39, no. 5, pp. 6000–6010, 2012.

[10] M. P. LaValley, "Logistic regression," *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.

[11] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, pp. 197–227, 2016.

[12] P. Prettenhofer and G. Louppe, "Gradient boosted regression trees in scikit-learn," in *PyData 2014*, 2014.

[13] C. Mood, "Logistic regression: Why we cannot do what we think we can do, and what we can do about it," *European sociological review*, vol. 26, no. 1, pp. 67–82, 2010.

[14] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[15] S. B. Imandoust, M. Bolandraftar *et al.*, "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," *International journal of engineering research and applications*, vol. 3, no. 5, pp. 605–610, 2013.

[16] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi, "Dnn-based prediction model for spatio-temporal data," in *Proceedings of the 24th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2016, pp. 1–4.

[17] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 29, no. 1, 2015.

[18] M. Jogin, M. Madhulika, G. Divya, R. Meghana, S. Apoorva *et al.*, "Feature extraction using convolution neural networks (cnn) and deep learning," in *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE, 2018, pp. 2319–2323.

[19] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: Lstm cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[20] A. Hannousse and S. Yahiouche, "Handling webshell attacks: A systematic mapping and survey," *Computers & Security*, vol. 108, p. 102366, 2021.

[21] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: a local svm approach," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3. IEEE, 2004, pp. 32–36.

[22] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[23] W. Python, "Python," *Python Releases Wind*, vol. 24, 2021.

[24] M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016, pp. 1–1.

[25] N. Ketkar and N. Ketkar, "Introduction to keras," *Deep learning with python: a hands-on introduction*, pp. 97–111, 2017.

[26] T. E. Oliphant *et al.*, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[27] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for high performance and scientific computing*, vol. 14, no. 9, pp. 1–9, 2011.

[28] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn," *Automated Machine Learning: Methods, Systems, Challenges*, pp. 97–111, 2019.

[29] "Dataset-cleaned," https://github.com/hannousse/Cleaned-PHP-Webshell-.

[30] T. Gunasegaran and Y.-N. Cheah, "Evolutionary cross validation," in *2017 8th International Conference on Information Technology (ICIT)*. IEEE, 2017, pp. 89–95.