Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Handling webshell attacks: A systematic mapping and survey

Abdelhakim Hannousse[a,*], Salima Yahiouche[b]

[a] Department of Computer Science, Université 8 Mai 1945, BP 401, Guelma 24000, Algeria
[b] LRS laboratory, Badji Mokhtar University, BP 12, Annaba 23000, Algeria

ARTICLE INFO

Article history:
Received 23 March 2021
Revised 3 May 2021
Accepted 2 June 2021
Available online 7 June 2021

Keywords:
Cybersecurity
Webshell attack
Systematic mapping
Survey

ABSTRACT

In recent years, there has been a significant increase in research interest in webshell attacks. Webshells are pieces of code that can be written in different scripting languages. They are uploaded to web servers after creating a breach making use of injection vulnerabilities. Webshells provide hackers a web interface to remotely execute commands, manipulate confidential data and invade web servers. The aim of this study is to provide researchers and practitioners with a holistic view of existing studies, approaches, techniques and tools for the detection of webshells and highlight potential gaps. To achieve this goal, a systematic mapping study is conducted. Forty-four primary studies are identified, surveyed and categorized following their scope of interest. Collections of malicious and benign webshells useful for validation and testing are identified. We also provide an overview of existing tools used for webshell detection with their limitations. Our findings revealed biases toward PHP webshells and machine learning technology as detection method. The study also revealed the need for more comprehensive studies and benchmark datasets for proper validation.

1. Introduction

Web servers are popular targets for common and recurrent cybersecurity attacks due to injection vulnerabilities. Injection flaws are at the head of the top ten list of the Open Web Application Security Project (OWASP) for the last few years (OWASP, 2017). Making use of injection flaws, malicious programs and scripts can be uploaded to web servers or injected into their exiting files in the aim to access data, exploit, compromise or destroy server systems. Webshells are typical examples of malicious scripts that make use of injection vulnerabilities. Webshell scripts enable the remote execution of commands and invading web servers. A webshell attack starts by writing malicious scripts and implanting them into a web server. Once the webshell script is set up, hackers may connect legitimately to the server by loading the webshell page in the browser. A webshell uses the interpreter of the web scripting language to communicate with the server's operating system (Kim et al., 2015). Through webshell pages, hackers may steal information, tamper databases, upload more dangerous malwares and escalate their privileges to cause more serious damages. The problem becomes severe when hackers use compromised servers to invade the internal network system and compromise more machines in the internal network. Webshells are also categorized as backdoors; they enable unauthorized access to web servers bypassing firewalls, making use of port 80, and other required authentication mechanisms. Webshells are easy to be encoded but hard to be detected (Qi et al., 2018). A webshell can be a single line of code enabling the remote execution of system commands given by users as described in the following PHP script. In the example, the `cmd` parameter value (system command) is remotely given by users through

* Corresponding author.
E-mail addresses: hannousse.abdelhakim@univ-guelma.dz (A. Hannousse), salima.yahiouche@univ-annaba.dz (S. Yahiouche).
https://doi.org/10.1016/j.cose.2021.102366
0167-4048/© 2021 Elsevier Ltd. All rights reserved.

HTTP requests to the web page that is already implanted in a web server.

```
<?php system($_GET['cmd']); ?>
```

Several threat actors rely on webshells to gain persistent access to network systems. Webshells are used for cyber espionage targeting many industries such as government, defense, financial, and telecommunications. China Chopper (Lee et al., 2013) is one of the most widely used webshells typically for cyber espionage. It has been used by several threat groups such as Leviathan (Plan et al., 2019) to remotely access enterprise networks and gain remote control of their systems. China Chopper has a GUI interface that provides powerful hacking facilities: (1) brute force password guessing against authentication portals, (2) upload, download, edit, delete, copy, rename and change the timestamp of files, (3) database management with SQL command runner, (4) command shell to interact with the operating system of the server.

The detection of webshells is a difficult task due to different factors: (1) malicious webshells use similar functionalities as those used by webmasters to remotely manage their own servers; (2) webshells generally do not leave full records in the system log and hackers want to have persistent access to the server they infected beforehand, to this end, they do not instantly and frequently access their implanted webshell pages; this makes them difficult to be tracked by inexperienced administrators; (3) webshells differ in size, they can range from a single line of code, as described in the above example, to sizable files with sophisticated graphical user interface items; (4) they can be written using several scripting languages or mix of different language scripts; (5) they use function hiding and string encoding to bypass rule and signature based matching tools.

In 2016, Starov et al. (2016) reported that despite the popularity of webshells and their real-world impacts shown through hacking large number of serves, they have not been paid sufficient attentions. Microsoft Detection and Response Team (DART) reported the steady increase of webshell attacks by an average 77,000 detections per month since 2019 (Detection, 2021). Since 2016, several studies are conducted addressing webshells' analysis and detection. To the best of our knowledge, this is a first exhaustive review focusing on webshells and their detection techniques. Gibert et al. (2020) have conducted a regular survey on machine learning detection and classification of general malwares. The focus of the present study is to shed the light on the different techniques and tools currently adopted for handling webshell attacks highlighting potential gaps. To this aim, we conduct a systematic mapping of existing studies, in literature, addressing webshells together with a comprehensive survey of existing endeavors. The contributions of the study can be summarized as:

1. present and categorize existing studies addressing webshell attacks
2. identify most adopted techniques used to detect, mitigate and prevent webshells
3. determine the set of techniques used to validate proposed solutions
4. identify potential gaps and limitations of the proposed solutions and their validation processes

The remainder of this paper is structured as follows: Section 2 details the research methodology adopted for this study; Section 3 presents and discusses the mapping results; Section 4 discusses identified gaps, challenges and issues; Section 5 discusses threats to validity and Section 6 concludes the paper.

2. Research methodology

In this section we present the details of the protocol adopted for conducting this mapping study. Following the guidelines of Petersen et al. (2008) and Felderer and Carver (2018), a systematic mapping study should include the following primary steps: a definition of research questions, search for relevant papers, selection process, data extraction and studies mapping. In the sequel, we describe the details of each step.

2.1. *Research questions*

The aim of this study is to review contemporary studies on malicious webshell detection. Thus, we formulate our research questions in light of the aims of the study and following the guidelines of Kuhrmann et al. (2017). This study is conducted with four main questions in mind:

RQ1. What type of researches are identified in the literature focusing on webshell attacks? This provides an overview of existing studies and their general focus on webshell attacks.

RQ2. How malicious webshells can be distinguished from benign webshells? This research question explores the set of features used to distinguish malicious from benign webshells.

RQ3. What type of solutions are proposed for the detection of webshells? This research question scrutinizes existing approaches and techniques used for securing web servers by detecting malicious webshells.

RQ4. Is there any publicly available datasets for webshells? This research question identifies all the sources of malicious and benign webshells that can be used to validate the proposed solutions.

RQ5. What kind of evidence is given regarding the evaluation and validation of proposed approaches and techniques for the detection of webshells? This research question evaluates the maturity of existing webshell detection techniques highlighting the set of empirical strategies used for verification and validation of solution proposals.

2.2. *Search process*

The search string used for the automatic search in digital libraries is constructed following the guidelines of Petticrew and Roberts in Petticrew and Roberts (2006). Therefore, we identified terms concerned with *population* and *intervention*. Population terms refer to the area of interest which is *"webshell attacks"* where intervention terms refer to any contribution to mitigate or prevent such attacks. Accordingly, final adopted search string is constructed from the word *"webshell"* or one of its variations and names in addition to terms describing the different potential contributions in the field. The final adopted search string is :

(”webshell” OR ”web shell” OR ”webpage backdoor” OR ”web backdoor”)

AND

(”analysis” OR ”experiment” OR ”search” OR ”detection” OR ”mitigation” OR ”prevention” OR ”taxonomy”)

To include studies with solutions for more generic attacks but tested with webshells, we did not restricted our search to titles, abstracts and keywords, instead full texts were searched for the correspondent terms. Moreover, for retrieving relevant studies, we followed the guidelines of Kuhrmann et al. (2017). Thus, we adopted the use of the following online academic libraries:

- IEEE Xplorer (https://ieeexplore.ieee.org)
- ACM Digital Library (https://dl.acm.org)
- SpringerLink (https://link.springer.com)
- ScienceDirect (https://www.sciencedirect.com/)
- Wiley Online Library (https://onlinelibrary.wiley.com)

To avoid missing relevant studies, we complement our automatic search by conducting recursive backward and forward snowballing on selected studies as suggested by Wohlin (2014, 2016). By backward snowballing, we check the relevance of references in approved papers. By forward snowballing, we check the relevance of papers citing approved papers. The snowballing is recursively applied to each newly approved paper. Google Scholar is used as a sole source for forward snowballing.

2.3. *Study selection process*

The set of retrieved papers by automatic search followed two screening stages. In the first stage, titles and abstracts are read to measure relevance. In the second stage, full texts of papers are examined to check if they meet our inclusion criteria. All the papers in the list are screened separately by the two authors; decisions are exchanged and conflicts are discussed and solved. Found papers from snowballing are also screened separately by the two authors before deciding whether to be included or excluded.

2.4. *Inclusion and exclusion criteria*

A set of inclusion and exclusion criteria are adopted to filter the set of retrieved papers by online academic libraries. In this review, only journal and conference papers are included. To avoid missing interesting papers, we did not restrict the search process to any period of time and we also consider early publications. The full lists of inclusion and exclusion criteria are described in Table 1.

2.5. *Data extraction process*

Following the guidelines of Petersen et al. (2008) a data extraction form is designed as illustrated in Table 2. Each paper is described in terms of its metadata such as year of publication, source and type. In addition, a set of required information for our analysis are extracted. These include type of webshells, proposed solutions, list of adopted features, datasets, and list of tests performed to validate the proposed solution.

3. Results of the search process

The search process is conducted in December 2020 and yielded 44 distinct papers. The designed query is formulated following the syntax of each search repository. Table 3 shows the number of returned papers by each library.

The set of 233 retrieved papers was subject for title and abstract screening. The screening was performed separately by both authors and conflicts are resolved by full papers screening. By checking the adopted inclusion and exclusion criteria, the number was reduced to 32 papers. Specifically, 4 papers were not selected due to E2 criterion where more recent publications of the same studies were identified and selected. By conducting recursive backward and forward snowballing, 12 new papers were identified. Figure 1 shows the distribution of selected studies according to their publication year and source. We notice that, although the prevalence of the attack in practice, the interest into webshells starts getting more attentions since 2016. Figure 1 also shows that the maximum number of publications come from IEEE and journals not indexed by considered digital repositories.

Table 4 shows the complete list of selected studies with their year and type of publication and how they were found (i.e.; automatic search or snowballing). Selected studies in Table 4 are firstly ordered following their type of retrieval (i.e.; those obtained by automatic search come first then come those found through snowballing); thereafter, the studies are ordered descendingly following their publication year.

Following the guidelines of Kuhrmann et al. (2017), we also experienced the use of Word Cloud to analyze the appropriateness of our search results. Figure 2 illustrates the most frequent words used in selected studies based on their titles and abstracts. The Figure illustrates that the most used words are webshell, detection, feature, PHP, method and model. These words are in fact consistent with the adopted search query and the set of conceived research questions. Moreover, this Word Cloud also provided a general overview of the focus of retrieved studies. Effectively, detection is the more frequent word compared with analysis (word in red ellipse); this may indicate that most retrieved studies are solution proposals. Moreover, since PHP is one of the most frequent script language name, it can be deduced that much focus is given to PHP webshells.

3.1. *What type of researches are identified in the literature focusing on webshell attacks?* (RQ1)

Identified studies can be classified into three main categories based on their underlying focus: *analysis, benchmarking*, and *detection*. Analysis studies focus is to examine malicious webshells and contemporary hacking tactics in order to provide a set of discriminative features to be used by tools for potential automatic detection. Benchmarking studies focus is to provide standard datasets for fair and proper evaluation of webshell detection techniques. Finally, detection studies focus is to propose practical solutions such as tools, algorithms or frame-

Table 1 – Inclusion and Exclusion criteria: I_i : inclusion criterion, E_i: exclusion criterion.

ID	Criteria
I1	papers written in English
I2	papers subject to peer reviews
I3	papers with webshell attack as a primary topic or papers addressing broader attacks with explicit refers and tests of webshells
E1	papers addressing broader attacks such as malwares, Remote Access Trojans (RAT) without explicit reference to webshells
E2	duplicate publications of the same study and authors; only the most complete publication is selected
E3	no peer review papers such as tutorials, editorials, and blogs
E4	papers not written in English
E5	papers without full text available

Table 2 – Data extraction form.

ID	Data item	Description	RQ
D1	Study ID	first author name + year	RQ1
D2	Year	year of the publication	RQ1
D3	Source	source of the publication	RQ1
D4	Type	conference or journal paper	RQ1
D5	Category	analysis, solution proposal, benchmarking	RQ1
D6	Webshell type	PHP, Mult-language, Language Independent	RQ1
D7	Features	list of features used for the solution proposal	RQ2
D8	Solution type	machine learning, technique, tool, heuristic, algorithm	RQ3
D9	Datasets	source and size of samples used for validation	RQ4
D10	List of tests	verification and validation techniques used to check the feasibility of the proposed solution	RQ5

Table 3 – Number of studies returned by each repository.

Repository	Search results
IEEE Xplorer	12
ACM Digital Library	33
SpringerLink	98
ScienceDirect	71
Wiley Online Library	19
Total	233

works for identifying malicious webshells. Figure 3 illustrates the distribution of studies regarding the three identified categories.

The search process revealed an extensive interest in detection proposals and less interest to other type of researches. Specifically, from the 44 selected studies, merely a single benchmarking study (2%) and two analysis studies (5%) are identified; all the 41 remaining studies (93%) are solution proposals. In this section we provide an overview of analysis studies. Detection and benchmarking studies are described later in response to RQ3 (Section 3.3) and RQ4 (Section 3.4) respectively:

- **p28** - Deng et al. (2016) analyzed the use of lexical features to detect real-world webshells. The study revealed that hacking techniques are rapidly evolving and the focus on looking for fixed patterns in webshell scripts is not sufficient for the detection of advanced webshells such as those using encryption functions.
- **p29** - Starov et al. (2016) performed a first comprehensive analysis of a large number of real-word webshells written

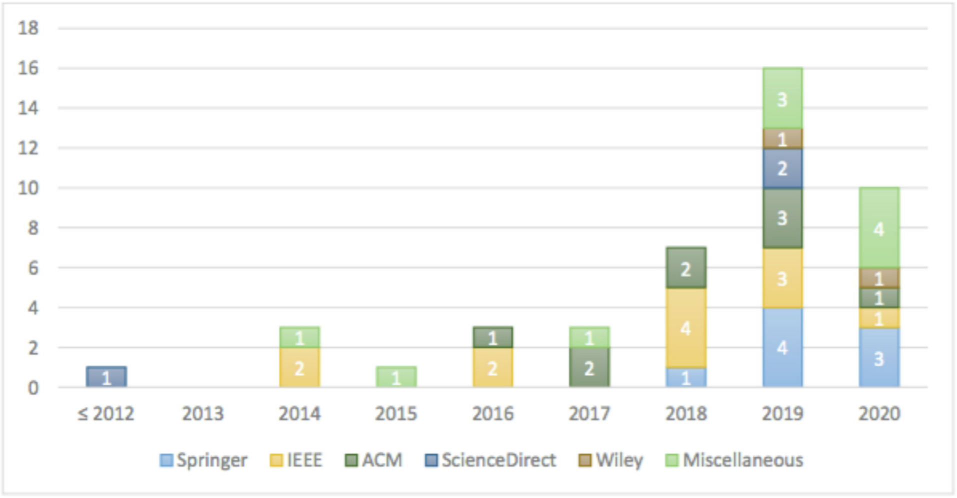

Fig. 1 – Distribution of selected studies by year and digital library.

Fig. 2 – Word cloud of titles and abstracts of selected studies.

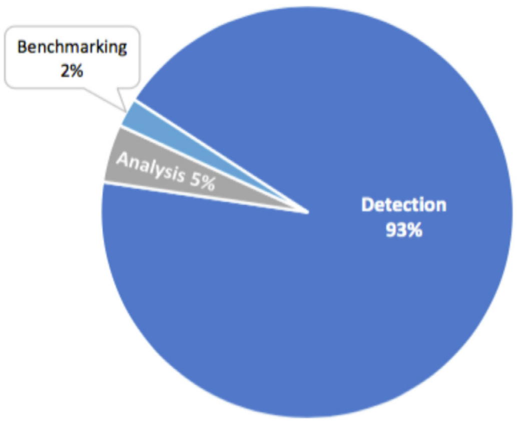

Fig. 3 – Distribution of selected studies per category.

in PHP language. Through static analysis of malicious webshell scripts (taint analysis), the study identified and quantified a set of common features and hacking tactics used by attackers to take over web servers and bypass antivirus engines. Through dynamic analysis (installing honeypots), the study quantified the number of webshell attacks and how webshells are used to control web servers.

Most studies are found focusing on PHP webshells. The choice is justified by the extensive use of PHP language in programming server-side applications as shown in Fig. 4.

Table 5 provides a closer overview of the interest of included studies in terms of webshell languages.

3.2. *How malicious webshells can be distinguished from benign webshells? (RQ2)*

Malicious webshells are very similar to benign webshells which make them hard to be distinguished. Identified studies used different sets of features to distinguish malicious webshells. Those features are based on experiencing and analyzing a large number of real-word webshells. Used features can be classified based on their source of extraction as described in Fig. 5: source code of scripts, their opcodes (i.e.; intermediate or bytecode), or web traffic analysis.

Investigating the set of selected studies, Table 6 reveals that webshell scripts are the most used source for the extraction of features followed by opcodes and HTTP-requests.

Adopted features extracted from webshell scripts can be classified following their nature into five distinct classes: *lexical, syntactical, semantical, statistical*, and *abstract* features. In the following we describe each class highlighting the rationale behind its adoption.

- **Lexical features:** indicate the way the script sources are written. Adversaries may use specific keywords and obfuscated instructions and parameters inside the code to hide the exact intention of the script. This results in scripts with small number of strings and specific patterns in tags or comments. Lexical features include the number of strings and use of malicious text patterns in comments.
- **Syntactical features:** refer to the expressions, variables and functions used in scripts. Adversaries may use system calls to elevate privileges, or some dangerous functions to upload other malwares or download important files. In addition, conditional statements can be used to adapt the webshell to the target server platform, while loops can be

Table 4 – List of selected studies: ST: Search Type, C: Conference paper, J: Journal paper, A: Automatic search, S: Snowballing.

ID	Cite	Year	Type	Source	ST	ID	Cite	Year	Type	Source	ST
p1	Ai et al. (2020a)	2020	J	IEEE	A	p23	Yong et al. (2018)	2018	C	IEEE	A
p2	Huang et al. (2020)	2020	C	Springer	A	p24	Zhang et al. (2018a)	2018	J	IEEE	A
p3	Jinping et al. (2020)	2020	C	ACM	A	p25	Zhang et al. (2018b)	2018	C	Springer	A
p4	Kang et al. (2020)	2020	C	Springer	A	p26	Sun et al. (2017)	2017	C	ACM	A
p5	Yong et al. (2020)	2020	J	Wiley	A	p27	Tian et al. (2017)	2017	C	ACM	A
p6	Zhao et al. (2020)	2020	C	Springer	A	p28	Deng et al. (2016)	2016	C	IEEE	A
p7	Croix et al. (2019)	2019	C	IEEE	A	p29	Starov et al. (2016)	2016	C	ACM	A
p8	Le et al. (2019)	2019	C	Springer	A	p30	Wrench and Ir-win (2016)	2016	J	IEEE	A
p9	Li et al. (2019a)	2019	J	IEEE	A	p31	Tu et al. (2014)	2014	C	IEEE	A
p10	Li et al. (2019b)	2019	J	ScienceDirect	A	p32	Mingkun et al. (2012)	2012	J	ScienceDirect	A
p11	Liu et al. (2019)	2019	J	ScienceDirect	A	p33	Ai et al. (2020b)	2020	J	MDPI	S
p12	Lv et al. (2019)	2019	C	Springer	A	p34	Guo et al. (2020)	2020	J	MDPI	S
p13	Naderi-Afooshteh et al. (2019)	2019	C	ACM	A	p35	Zhongzheng and Luk-tarhan (2020)	2020	J	IOS Press	S
p14	Naderi-Afooshteh et al. (2019)	2019	C	ACM	A	p36	Zhu et al. (2020)	2020	J	MDPI	S
p15	Nguyen et al. (2019)	2019	C	ACM	A	p37	Kurniawan et al. (2019)	2019	J	ICIC	S
p16	Tao et al. (2019)	2019	C	Springer	A	p38	Lian et al. (2019)	2019	J	AIRITI	S
p17	Tianmin et al. (2019)	2019	C	IEEE	A	p39	Wang et al. (2019)	2019	J	Francis Press	S
p18	Yang et al. (2019)	2019	C	Springer	A	p40	Wu et al. (2019)	2019	J	Wiley	S
p19	Cui et al. (2018)	2018	C	IEEE	A	p41	Wang et al. (2017)	2017	J	AIRITI	S
p20	Fang et al. (2018)	2018	C	ACM	A	p42	Kim et al. (2015)	2015	J	KIPS	S
p21	Li et al. (2018)	2018	C	ACM	A	p43	Wrench and Ir-win (2014)	2014	C	IEEE	S
p22	Qi et al. (2018)	2018	C	IEEE	A	p44	Jeong et al. (2014)	2014	J	RIP	S

used to reveal passwords. Typical examples of syntactical features are the ratio of using conditional statements and loops, invoking dangerous functions and calling specific language components.

- **Semantical features:** lexical and syntactical features provide a general overview of how webshells are written, semantical features provide intentions that can be derived from lexical and syntactical items. For example, instead of considering the ratio of arbitrary loops inside scripts, one can consider only the presence of loops over port scanning function calls or login access.
- **Statistical features:** malicious webshells often use encrypted and obfuscated code to bypass firewalls. Encryption functions can also be used by benign webshells for security reasons. Therefore, statistical features are useful in distinguishing malicious webshells by comparing some statistical values of webshells with normal files. Examples of such features are information entropy and compression ratio.

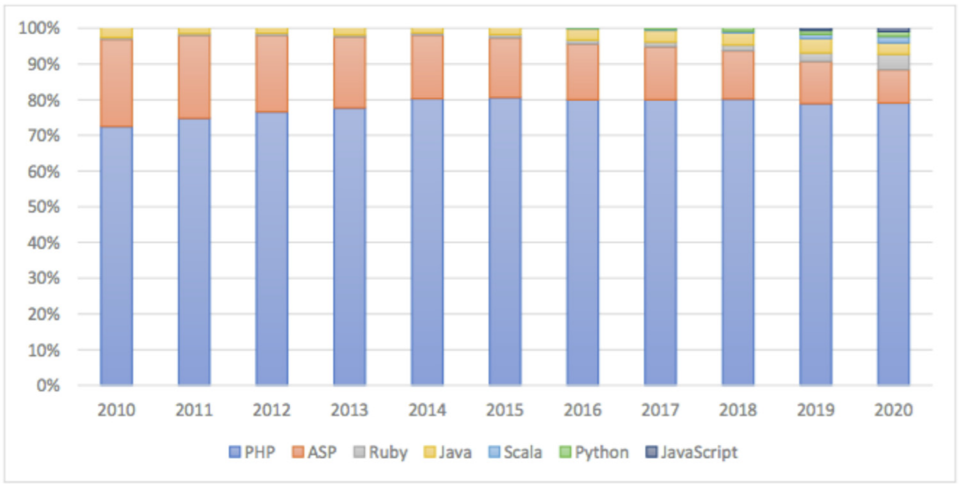

Fig. 4 – Top server-side programming languages: Web Technology Surveys report: https://w3techs.com.

Table 5 – Webshell language.

Webshell type	Studies	#	%
PHP	p1-5, p7, p8, p10, p13-21, p23, p25, p29, p30, p33-39, p41, p43	30	68.18%
ASP	p32	1	2.27%
Multi-Language	p6, p9, p12, p22, p24, p26, p32, p44	8	18.18%
Language Independent	p11, p18, p27, p40, p42	5	11.36%

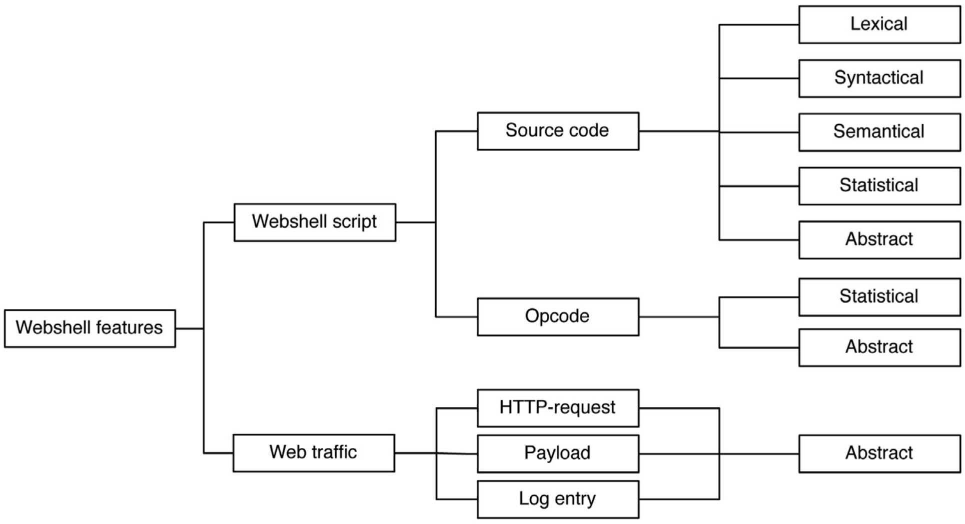

Fig. 5 – Classification of webshell features.

Table 6 – Feature extraction sources: Hybrid = source code + bytecode.

source of features	studies	#	%
Source code	p6-10, p12-14, p16, p21, p22, p26, p28-32, p36, p37, p41, p43, p44	22	51.16%
Bytecode	p1, p5, p15, p17, p25, p34, p39	7	16.28%
Hybrid	p2-4, p19, p20, p23, p33, p35, p38	9	20.93%
HTTP-requests	p18, p24, p27	3	6.97%
Log entries	p40	1	2.33%
Payloads	p11	1	2.33%

- **Abstract features:** go beyond lexical, syntactical and semantical features. They are useful to reveal hidden parts of webshells that cannot be detected through syntactical and semantical analysis. In this study, abstract features are used to refer to vectorized data such as source code, opcode and web traffic. These features are mainly used by deep learning based approaches.

Table 7 summarizes the set of used features with the list of studies adopting them. Features of each category are ordered following their adoption frequency. It can be derived from Table 7 that abstract features are the most used in the detection of webshells (26/41 studies) followed by statistical and syntactical features (16/41 studies). Semantical features are the less used ones (3/41 studies). Despite the fact that statistical features fail to detect webshells with clear codes and small webshells (Qi et al., 2018), they are used in conjunction with other features in several studies and tools. The increasing interest in adopting abstract features at the expense of other features is due to a combination of factors:

1. the diversity of webshell languages and their expressiveness.
2. the common tacit agreement on the ability of abstract features to detect hidden characteristics of webshells and hence detect unknown webshells.
3. attackers are aware of existing tools and hence they change webshell coding styles everytime which makes explicit features such as lexical and syntactical less efficient in detecting new webshells.
4. less attention is given to semantical features due to the complexity of their identification and location.
5. the interest in the use of deep learning approaches increased the use of such kind of features (see Section 3.3.1).

3.3. *What type of proposed solutions for the detection of webshells? (RQ3)*

Apart from the 2 analysis studies discussed in Section 3.1, 41 studies provided solutions for the detection of malicious webshells. Figure 6 shows the distribution of those studies regarding the type of their solution proposals. From those 41 studies, 29 of them (71%) adopted machine learning technology, only 12 studies (29%) proposed other kinds of solutions. In the sequel, we review each of those studies with respect to the type of their solutions.

3.3.1. *Machine learning based solutions*

Machine learning is an artificial intelligence technology that enables computers to learn from experienced data without having been explicitly programmed to do so. It is a modern tool for discovering patterns in one or more data streams and making predictions based on statistics. Machine learning reveals its full potential in situations where insights need to be spotted from large, diverse and varied datasets. With the increasing number of webshell attacks and detected malicious webshells, machine learning has become one of the important technologies used to distinguish malicious from benign webshells. Machine learning uses well-defined algorithms, named classifiers, to predict and classify input samples. Classifiers may need (*Supervised* or *Semi-supervised*) or need not to be (*Unsupervised*) trained on a labeled set of data samples. Different classifiers are proposed for the detection of webshells.

Table 8 describes the set of classifiers experimented (with normal face) and validated (with bold face) by identified studies. The results from Table 8 shows that SVM and RF are the most experimented classifiers (14 studies) followed by NB (12 studies), MLP and CNN (9 studies). Moreover, most validated

Table 7 – Features for webshell detection.

category	feature	studies
Lexical	number of strings	p10, p26, p29
	use of sensitive keywords	p20, p31
	number of different strings	p26
	number of special characters	p26
	total length of script	p26
	maximum length of lines	p36
	use of connecting symbols	p32
	use of dangerous functions	p2, p6, p7, p19, p20, p26, p28, p29, p32, p36, p44
Syntactical		
	use of obfuscation functions	p7, p26, p28, p32
	ratio of conditional statements	p10, p36
	ratio of loop statements	p10, p36
	number of global variables	p10, p36
	source code hash similarity	p6, p7
	number of script tags	p26
	average number of binary operations	p10
	number of notes	p26
	ASP components calls	p32
	file invocation	p26
	ActiveX control call	p26
	send emails function	p29
	use of char manipulation functions	p26
	use of dangerous variables	p2
	matching signature	p7
	malicious text patterns	p8
	implicit data flow	p10
Semantical		
	explicit data flow	p10
	loops over login or port scan	p29
	404-HTTP-user agent header	p29
	abnormal `#include` statement	p32
Statistical	longest word length	p2, p4, p6, p8, p10, p19, p20, p26, p31, p33, p36
	information entropy	p2, p4, p6, p7, p8, p19, p20, p33, p36
	index of coincidence	p4, p6, p8, p19, p20, p33
	amount of high risk functions	p4, p8, p13, p31, p33, p37
	compression ratio	p4, p6, p19, p33
	amount of `eval()` function	p19, p29
	special characters information entropy	p41
	quotation information entropy	p41
	maliciousness score	p13
	number of malicious signatures	p31
Abstract	vectorized opcode	p1, p2, p4, p5, p17, p19, p25, p33, p34, p38, p39
	vectorized source code	p3, p9, p12, p16, p22, p23, p24, p35, p38
	opcode sequences	p3, p15, p20
	vectorized opcode hash	p4, p19
	vectorized HTTP-request	p18, p27
	vectorized payload	p11
	vectorized log entry	p40
	vectorized system call sequence	p6

classifiers are CNN and LSTM (4 studies each) followed by SVM and RF (3 studies).

Single learning: traditional classifiers are used for the prediction of malicious webshells. 5 studies (17%) have used single model learners. All are supervised learners except K-Means which is experimented once in **p5**.

- **p18** - Yang et al. (2019) proposed a runtime webshell detection system based on the analysis of HTTP requests. An SVM classifier is trained on preprocessed and vectorized HTTP requests. The preprocessing step incorporated decoding and merging GET and POST parameters. Vectorization is performed by the extraction of bag of words using a 4-gram based algorithm. Using SVM, HTTP requests are

Table 8 – Summary of machine learning based detection models. lcr: limited computing resources.

type	classifier	description	experimented by
Single learning	SVM	Support Vector Machine	p2, p3, p5, p10, p11, **p18**, p22, p23, p24, p25, p27, p34, p35, **p36, p37**
	NB	Naïve Bayes	p3, p5, p11, p17, p18, p23, p24, p25, p27, **p34**, p35, p37, p39
	DT	Decision Tree	p1, p5, p11, p12, p25, p27, p37
	KNN	K-Nearest Neighbors	p1, p2, p5, p25, p27
	LR	Logistic Regression	p11, p24, p26, p33
	MatDec	Matrix decomposition	p9, **p26**
	K-Means	k-means clustering	p5
Ensemble learning	RF	Random Forest	p1, **p2**, p3, p4, **p5, p10**, p11, p17, p18, p19, p20, p23, p33, p34, p38
	XGBoost	eXtreme Gradient Boosting	p1, p3, p4, p9, **p17**, p38
	RF-GBDT	*stack*(2*RF, **GBDT**)	p4, p9, p15, **p19**
	FRF-WD	*stack*(FastText, **RF**)	p9, **p20**, p36
	AdaBoost	Adaptive Boosting	p1, p33, p38
	GBDT	Gradient Boost Decision Tree	p10, p19, p38
	RF-AdaCost	*stack*(2*RF, **AdaCost**)	**p4**
	Dynamic-Stack	*stack*(X,Y,Z, **DT**): X,Y,Z = 3 best classifiers	**p38**
	AdaCost	Cost-Sensitive Boosting	p4
	LightGBM	Light Gradient Boosting Machine	p33
	ET	Extremely randomized Trees	p5
	RF-DT	*stack*(2*RF, **DT**)	p19
Deep learning	CNN	Convolutional Neural Network	p3, p9, **p11, p12, p15**, p16, p24, p25, **p27**
	MLP	Multi Layer Perceptron	p1, p3, p5, p16, **p16**, p17, p24, p35, **p39**
	LSTM	Long Short-Term Memory	**p6**, p9, **p11, p16**, p24, **p40**
	GRU	Gated Recurrent Unit	**p9**, p11
	CNN-LSTM	CNN with LSTM layers	p3, **p24**
	ANN	Artificial Neural Network	p27, p37
	DNN	Deep Neural Network	**p23**
	DNN-LSTM	DNN with LSTM layers	**p22**
	SRNN-CapsNet	SRNN for reduction + CapsNet for classification	**p35**
	RNN	Recurrent Neural Network (standard)	p11
	SRNN	Structural Recurrent Neural Network	p35
	CapsNet	Capsule Neural Network	p35
	BPNN	Back Propagation Neural Network	p23
	HMM	Hidden Markov Model	p40
Mixed	Mix1	CNN ◁ *len*(opcode)>30 ▷ RF	**p3**
	Mix2	RF ◁ lcr ▷ *vote*(KNN,NB,DT,MLP,K-Means,SVM)	**p5**
	WSLD	Heuristics | RF | RNN	**p6**
	WS-LSMR	*weighted-vote*(LR,SVM,MLP,RF)	**p1**
	Deep Super Learner	*DeepSuperLearner*(RF,LR,MLP)	**p33**
	SmartDetect	*vote*(SVM,KNN,NB,DT,CNN)	**p25**
	Mix3	*stack*(SVM, MLP, RF, **LR**)	p1
	Mix4	*vote*(LR, SVM, MLP, RF)	p1
	Mix5	*vote*(K-Means,MLP,NB,DT,SVM,KNN,SVM)	p5

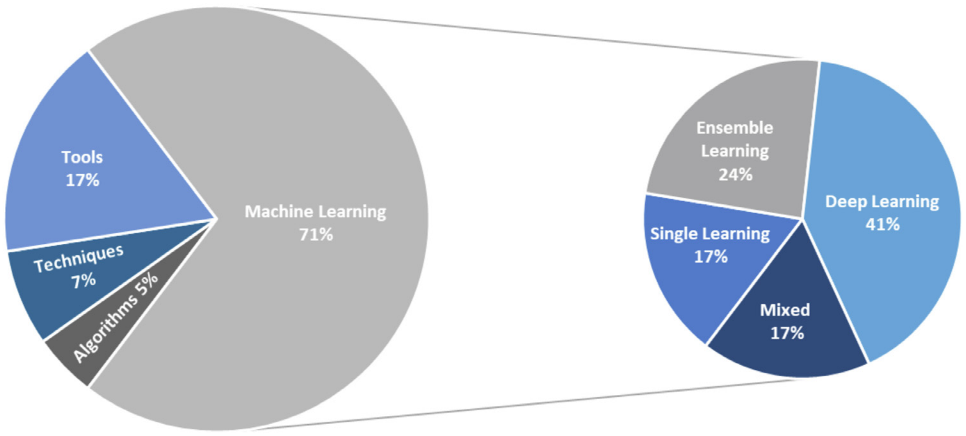

Fig. 6 – Distribution of proposed solutions.

classified into: suspicious, attack, and no attack. Suspicious requests are reported to the administrator to check the legitimacy of requested files.

- **p26** - Sun et al. (2017) proposed a new supervised machine learning based algorithm in the aim to identify unknown webshells. A set of source code statistical features extracted from known webshells are divided into different groups based on their counts and used to build a scoring matrix. The produced matrix is used to analyze data, get weights of features, and build a model to predict new webshell files.
- **p34** - Guo et al. (2020) adopted the use of vectorized opcode sequences extracted from PHP webshells to build and test a Naòve Bayes model. Vectorization is performed making use of 2-gram and TF-IDF models. Conducted experiments showed that the proposed model outperforms other classifiers such as SVM and RF.
- **p36** - Zhu et al. (2020) proposed a webshell detection system based on SVM. The SVM classifier is trained and tested on hybrid source code features (i.e.; lexical, syntactic and statistical). In total, 20 features were firstly measured and normalized, then filtered using the fisher score to designate the most important features. At the end, 16 features were selected to build the SVM model. The proposed model is compared with different anti-virus engines, tools and solutions proposed in **p20** and **p31**. The comparison was based on the detection rate of well-known webshell families.
- **p37** - Kurniawan et al. (2019) used source code based statistical features. They count the number of calls to 50 predefined sensitive functions in each file. Feature vectors are built from obtained counts and used to feed different classifiers. The experiment showed that SVM outperforms other classifiers such as DT and NB. However, the dataset used in the experiment is very small (i.e.; only 100 files), therefore, more investigation is required to validate the proposed model.

Table 9 sums up the number of malicious and benign webshells used for testing together with the validation method and the best performance of each model. Surprisingly, none of these works has used cross-validation as a validation method.

Ensemble learning models: multiple classifiers are trained to solve the same problem and combined to get better performance than that which could be obtained from any of the constituent classifiers taken individually. 7 studies (24%) adopted ensemble learning for automatic webshell detection. Different ensemble learners are proposed and tested. Table 10 describes the details of datasets used for testing existing ensemble models together with the validation method and the best performance of each model. In the sequel, we describe the structure of each of those models.

- **p2** - Huang et al. (2020) used 2 statistical features (i.e.; longest string, information entropy) and 21 high risk functions and variables together with TF-IDF vectorization of PHP opcode sequences. They found that RF ensemble classifier performs better than SVM and KNN. The peculiarity of this approach is the use of two kinds of opcode sequences: those that correspond to the execution paths in the code, and those that correspond to the code itself. The proposed model showed improved performances compared with the merely use of opcode sequences that correspond to the code.
- **p4** - Kang et al. (2020) proposed a stacking ensemble learner that combines two RFs and an AdaCost classifier. The proposed model is comparable to that proposed in **p19**. The first RF classifier is trained on vectorized PHP opcode sequences using a variant of TF-IDF. The second RF is trained on hashed opcodes. The predictions of the two RF classifiers are combined with 5 statistical features extracted from source file scripts and used as inputs to the AdaCost classifier. The conducted experiments showed that the proposed model outperforms existing tools such as D-shield (2021) and Webdir+ (2021) and the proposed model in **p19**.
- **p10** - Li et al. (2019b) experimented 8 syntactical and semantical features extracted from source codes considering taint analysis. Extracted features are used to feed several classifiers. It was found that RF outperforms SVM and GBT.

Table 9 – Datasets and results of single learning models.

study	classifier	#benign	#malicious	validation method	best performance
p18	SVM	3,500	1,743	Random sampling 8:2	F1-score = 98.90%
p26	MatDec	1,002	347	Random sampling 8:2	F1-score = 98.20%
p34	NB	1,200	500	Random sampling 7:3	F1-score = 97.00%
p36	SVM	1,056	1,056	Random sampling 6:4	Accuracy = 93.72%
p37	SVM	50	50	Random sampling 8:2	Accuracy = 92.00%

Table 10 – Datasets and results of ensemble learning models.

study	classifier	#benign	#malicious	validation method	best performance
p2	RF	11,397	912	Random sampling 7:3	F1-score = 98.40%
p4	*stack*(2*RF, AdaCost)	2,236	2,162	Random sampling 7:3	Accuracy = 95.30%
p10	RF	475	475	10-fold cross-validation	Accuracy = 91.70%
p17	XGBoost	10,121	11,778	Random sampling 7:3	F1-score = 97.80%
p19	*stack*(2*RF, GBDT)	2,388	2,232	10-fold cross-validation	F1-score = 99.09%
p20	*stack*(FastText, RF)	6,934	1,587	10-fold cross-validation	F1-score = 97.78%
p38	*stack*(X, Y, Z, DT)	3,478	99.39	10-fold cross-validation	F1-score = 99.10%

- **p17** - Tianmin et al. (2019) experienced different sizes of TF-IDF vectors obtained by analyzing 2- and 4-grams of PHP opcode sequences. Several classifiers are trained and tested in each case. It is found that XGBoost ensemble learner outperforms RF, NB, and MLP.
- **p19** - Cui et al. (2018) used a stacking ensemble learner that combines two RFs and a GBDT classifier. They used hybrid features. The first RF classifier is trained on vectorized PHP opcodes using TF-IDF. The second RF is trained on hash vectors of opcodes. The predictions of RF classifiers are combined with 6 statistical features extracted from source file scripts and used as inputs to the GBDT classifier. The model is found outperforming existing tools such as D-shield (2021) and PHP-Malware-Finder (2016).
- **p20** - Fang et al. (2018) proposed a stack based ensemble learner that combines FastText (Joulin et al., 2017) and RF classifiers. FastText classifier is trained on opcode sequences generated from PHP scripts; the prediction value of FastText is then combined with 5 statistical features and used as inputs for RF training and prediction.
- **p38** - Lian et al. (2019) proposed a framework for a stacking ensemble learner that is built on dynamic selection of top 3 best classifiers and a DT classifier. The framework used two feature vectors: one obtained through 2-gram and TF-IDF vectorization of PHP source scripts, the other is obtained through vectorization of opcode sequences. The top 3 best classifiers are chosen for each case and their predictions are fed to a DT classifier for final prediction.

Table 11 – Datasets and results of deep learning models.

study	classifier	#benign	#malicious	validation method	best performance
p9	RNN	7,400	4,500	Random sampling 7:3	F1-score = 98.95%
p11	CNN	1,104	528	Random sampling 7:3	F1-score = 94.11%
	RNN				F1-score = 92.55%
p12	CNN	8,361	3,944	10-fold cross-validation	F1-score = 99.30%
p15	CNN	6,057	1,324	Random sampling 7:3	F1-score = 99.41%
p16	MLP	-	-	Random sampling 8:2	F1-score = 99.50%
p22	DNN-LSTM	6,669	6,669	Random sampling 5:5	Accuracy = 98.54%
p23	DNN	1,728	1,728	Random sampling 7:3	F1-score = 98.08%
p24	CNN-LSTM	634,969	314,838	Random sampling 6:2:2	F1-score = 98.51%
p27	CNN	3,990	3,691	10-fold cross-validation	F1-score = 98.60%
p35	SRNN-CapsNet	-	-	Random sampling 6:2:2	F1-score = 99.20%
p39	MLP	3,000	3,000	10-fold cross-validation	Accuracy = 94.40%
p40	LSTM	13,986	57,160	10-fold cross-validation	F1-score = 93.17%

Deep learning models: Use of artificial neural network architectures of multiple layers, where each layer is endowed with a specific number of nodes and an activation function. 12 studies (41% of machine learning based solutions) have proposed deep learning models to detect webshells. Table 11 shows the details of datasets used for testing existing deep learning models with the best performances obtained by each model. In the following we discuss each proposed model with its underlying structure and experiment findings.

- **p9** - Li et al. (2019a) proposed an RNN-GRU model for the detection of malicious webshells. The peculiarity of the proposed model is the focus on capturing intra-line words association of source scripts. Therefore, the model is fed with vectorized words from each line in webshell sources. Their experiments showed that the model outperforms existing detectors of multi-language webshells including the matrix decomposition learner proposed in **p26**. However, the model has slightly low performance with PHP webshells than RF-GBDT and FRF-WD ensemble learners proposed in **p19** and **p20**, respectively.
- **p11** - Liu et al. (2019) proposed two distinct deep learning models (CNN and RNN-LSTM) for the detection of different types of network attacks including webshell attacks. The CNN model is build on preprocessed and encoded payloads making use of a word embedding technique. The RNN-LSTM model is build on the first sequence of characters in payloads. The number of characters is fixed to the average length of payload samples. The results showed that both proposed models outperform traditional models such as LR, SVM and RF. In addition, RNN-LSTM provided a slight improvement compared with CNN.
- **p12** - Lv et al. (2019) experimented different CNN models each is fed with a different vectorization of webshell source scripts: One-hot, bag-of-words, and Word2Vec. The model with bag-of-words vectorization provided the better accuracy score.

- **p15 -** Nguyen et al. (2019) proposed a CNN model fed with opcode sequences of 100,000 length max. Samples with opcode sequences less than 100,000 are padded to have the required length. The experiment showed that applying Yara rules to the original dataset has an effect on removing fake webshells and hence improving the performance of the model.
- **p16 -** Tao et al. (2019) experimented MLP, CNN, and LSTM models. They found that CNN and LSTM models converge better in time than MLP that outperforms the other models in terms of accuracy. Models were trained and tested on vectorized PHP source codes using optimized n-grams (1 and 2).
- **p22 -** Qi et al. (2018) proposed a DNN model. The model is fed with token streams that are firstly generated from source code scripts through lexical analysis, then transferred into a sequence of vectors, and finally converted into short streams using a down-sampling based compression algorithm. A comparison with other sampling methods, the down-sampling method provided the best accuracy score.
- **p23 -** Yong et al. (2018) experienced a DNN model with three different inputs: TF-IDF vectorized source codes, opcode sequences, and TF-IDF vectorized opcodes. The DNN fed with TF-IDF vectorized source codes outperformed the other models.
- **p24 -** Zhang et al. (2018a) proposed a CNN-LSTM model build on preprocessed HTTP requests. For each request, the URL and post body fields are transformed into 300 length character based vector. These vectors are used to feed the CNN-LSTM model. A comparison with traditional vectorization methods, such as TF-IDF and Word2Vec, showed the relative efficiency of the proposed model in the detection of unknown webshells.
- **p27 -** Tian et al. (2017) proposed a CNN model fed with vectorized HTTP requests using Word2Vec. The proposed model outperformed traditional machine learning models such as NB, DT and SVM.
- **p35 -** Zhongzheng and Luktarhan (2020) proposed a two step deep learning based detection model. First, a SRNN with attention mechanism was adopted to reduce the size of inputs. Second, a 4 dimension capsule neural network (CapsNet) is used to perform predictions. The model uses fused vectors generated from vectorizing both source and opcodes.
- **p39 -** Wang et al. (2019) proposed a two hidden layers MLP model with 5 and 2 nodes respectively trained on optimized bi-grams of PHP opcodes.
- **p40 -** Wu et al. (2019) proposed a model named SB-LSTM which consists of four LSTM layers with 60 neurons each and a dense layer. They adopted a session based detection method. Sessions are identified by statistical analysis and calculation of time intervals between each two entries in the log files. The SB-LSTM model is fed with 6 length vectors encoding each session log entry.

Mixed detection models: Mixed detection models combine different kinds of machine learning: single, ensemble and/or deep learning. 5 studies (17%) are found fitting this category. Table 12 shows the datasets and results of existing models of this category.

- **p1 -** Ai et al. (2020a) proposed a weighted voting ensemble model that incorporates 4 classifiers (2 single learners: LR, SVM, 1 ensemble learner: RF, and 1 deep learner: MLP). Weights of base classifiers are calculated using a well-defined formula based on the accuracy of each classifier on the training phase. Base classifiers are trained and tested on 4-gram based TF-IDF vectorization of opcodes within a well-defined feature selection algorithm.
- **p3 -** Jinping et al. (2020) proposed a solution to the detection of PHP webshells by combining an ensemble learning model (RF) and a deep learning model (CNN). First, text features are extracted from source scripts using a bag of words model and TF-IDF for optimization. Second, opcode sequences are generated from sources and used to form opcode feature vectors. Source code features are used to train and test the RF model, where opcode features are used for CNN. Accordingly, CNN model is used for the detection of samples with opcode sequences' length exceeding 30, where RF model is used for samples that fails at the opcode generation process or their opcode sequences length is less than 30.
- **p5 -** Yong et al. (2020) proposed two kinds of ensemble learners dedicated to the protection of IoT networks. A lightweight ensemble learner based on RF for devices with moderate computing resources. Another heavyweight ensemble learner based on votes of 6 classifiers (5 single learning classifiers: KNN, NB, DT, K-Means, SVM and a deep learning classifier: MLP) for devices with more powerful computing capabilities. Both learners are trained on 100-dimension feature vectors extracted using TF-IDF vectorization of PHP opcode sequences.
- **p25 -** Zhang et al. (2018b) proposed an easy voting based model that is built on four single learning classifiers (SVM, KNN, NB, DT), and a deep learning classifier (CNN). The five base classifiers are trained separately on different training sets. The proposed model used bi-gram sequences of opcodes to form input feature vectors.
- **p33 -** Ai et al. (2020b) proposed a deep super learner for webshell attack detection. The deep super learner implements a probabilistic stacking model built on 3 base classifiers: 1 single learner LR, 1 ensemble learner RF, and 1 deep learner MLP. A weight value for each base classifier is measured using Sequential Least Squares Programming algorithm (SLSQP). Those values are used as a base to predict new samples. The model used a combination of 5 statistical features and Word2Vec vectorization of opcodes. A genetic algorithm was applied to reduce feature vectors dimension.

3.3.2. *Non-Machine learning based solutions*

In this section, we discuss non-machine learning proposals including techniques, algorithms and tools.

Methods/Techniques: Only 3 studies (7%) proposed new techniques for the detection of webshells.

- **p14 -** Naderi-Afooshteh et al. (2019) proposed Cubismo to improve the detection rate of existing malware detection

Table 12 – Datasets and results of mixed models.

study	classifier	#benign	#malicious	validation method	performance
p1	*weighted-vote*(LR, SVM, MLP, RF)	5,379	566	Random sampling 8:2	Acc = 94.28%
p3	CNN ◁ len(opcode)>30 ▷ RF	11,161	4,406	Oversampling	F1 = 94.77%
p5	*vote*(KNN,NB,DT,MLP,K-Means,SVM)	2,593	1,551	Random sampling 8:2	F1 = 98.32%
	RF				F1 = 97.92%
p25	*vote*(SVM, KNN, NB, DT)	8,045	826	Bootstrap sampling	F1 = 85.60%
p33	*DeepSuperLearner*(LR, RF, MLP)	5,379	571	7-fold cross-validation	Acc = 98.90%

tools. Cubismo leverages counterfactual execution for exploring all potential execution paths and discovering hidden code in PHP scripts. Following Cubismo, original scripts are firstly normalized by removing extra lines, comments and white spaces. Afterwards, they are made subject for counterfactual execution. Therefore, the scripts are analyzed where exceptions, runtime errors and nested predicates are ignored to identify new paths, and recursive deobfuscation is applied to handle multi-layer encryption. New program files are created for each new explored path and encountered dynamic construct. New programs are executed in a sandbox environment for potential detection. The resulting program files are used as inputs to existing malware detection tools. A webshell is detected if one of the input files is reported as malware.

- **p21** - Li et al. (2018) proposed a diversification approach for the detection of known and unknown webshells. An uploaded PHP code is transformed into different versions to be executed in different application environments. To this aim, mapping functions that map PHP keywords and built-in functions into new ones are generated and applied. Therefore, every call to the original code triggers the execution of different versions of the code. A webshell is detected if any of the execution environments reports an abnormal result.
- **p31** - Tu et al. (2014) proposed a threshold based detection method. The method uses a database of malicious functions and signatures often used in webshells and their dangerousness score (i.e.; high, medium, low). All files of the web server are scanned, matching malicious functions and signatures in the database. In addition, the length of the longest word in header tags is determined for each file and blacklist keywords in comment lines are counted. An optimal threshold value for each feature is identified by scanning a set of benign files. Therefore, any file with the number of malicious functions, malicious signatures, or string word length exceeding the correspondent identified threshold value is considered as suspicious and reported to the administrator for further checks. Otherwise, scanned files are compared with their original ones and if any one does not match, the file is reported as suspicious.

Algorithms: Two new algorithms (5%) are proposed to detect new webshells.

- **p7** - Croix et al. (2019) proposed a training algorithm for the aggregation operator WOWA (Weighted Ordered Weighted Averaging). WOWA is proposed as a PHP webshell detector that combines five heuristics: signature matching, fuzzy hashes, dangerous routines, obfuscation degree and information entropy. A genetic algorithm is used to identify the best parameter values of the operator.
- **p41** - Wang et al. (2017) proposed an algorithm for the detection of unknown webshells. The proposed algorithm measures two entropy values of special characters and quotes used in normal files. Threshold values are determined and used for the detection of webshells. A script file is reported as webshell if the entropy values of both special characters and quotes exceed the correspondent threshold values.

Tools: We distinguish two kinds of tools: tools that are used by selected studies as base lines for their proposals, and new proposed tools by identified studies. Table 13 summarizes the set of baseline tools used by selected studies together with a reference and a short description to each tool. Tools can be classified according to their underlying detection mechanisms into six different classes:

1. **Database matchers:** different information about known webshells are collected into databases and used for identifying and locating them potentially with their variants. These information include signatures, sensitive keywords,

Table 13 – Webshell detection tools.

class	tool	ref.	description	used in
Database matcher	Web Shell detector	Emposha (2016)	A PHP/Python tool that enables the detection of webshells. It uses a signature database for matching webshells. It enables sending malicious files to the team members for manual inspection; one may receive a report and the database is updated accordingly.	p6, p12, p19, p31, p34, p42
	PHP Malware Finder	PHP-Malware-Finder (2016)	A tool that enables scanning web servers and identifying PHP webshells based on matching a set of semantic Yara rules. It provides support for several deobfuscation techniques for encoded files.	p13, p15, p19
	Grep	TitanWolf (0000)	A Unix text search command that uses regular expressions for matching texts in files. For webshell detection, regular expressions are used to match suspicious keywords and functions.	p31
	D-Shield	D-shield (2021)	An active defense security protection tool that prevents intruders from uploading and executing malicious scripts. For webshell detection, D-Shied scans uploaded scripts and identifies calls to dangerous functions.	p2, p4, p12, p19, p34, p38
	Webshellpub	Web (2021)	Online webshell killing engine that uses a large database of known webshells.	p4, p38
	clamAV	Clamav (2002)	Free antivirus engine that uses a huge database of signatures.	p10, p13, p36
	findbot.pl	findbot.pl (2015)	A sensitive keyword and hash comparator based detector written in Perl	p10
	FindWebshell	Findwebshell (2018)	A sensitive keyword based detector written in Python	p38
ML	Chaitin Cloud-Walker	Chaitin Cloud-Walker (2020)	A neural network based detection system	p2, p6, p36
Taint Checker	RIPS	Rips (2020)	A static source code analyzer for vulnerabilities in PHP scripts. RIPS tokenizes and parses PHP files and transforms them into a program model. Taint analysis is used to analyze the data flow of user inputs that the application receives. The data flow of a user input is traced throughout the complete code base, including all files, functions, classes, and methods. If the user input is used in a security sensitive operation, a security vulnerability is reported.	p8
	THAPS	Jensen et al. (2012)	A source code analyzer for vulnerability detection in PHP scripts. THAPS uses symbolic execution on AST (Abstract Syntax Tree) generated through a taint analysis. By taint analysis, all possible user inputs are generated and their propagations are controlled. If the input is able to alter the outcome of the application, a vulnerability is reported.	p8

(*continued on next page*)

rules, entire codes or file hashes. Those tools alone are known to be less effective in the detection of new, advanced and unknown webshells. This is due to the diversity and the continuous evolvement of hacker coding tactics and the high expressiveness of script languages. Effectively, they found producing a large number of false positives (Deng et al., 2016; Kim et al., 2015).

2. **Machine learning based tools:** some open access tools adopted machine learning technology without further ex-

Table 13 (*continued*)

Hybrid	Linux Malware Detect	Networks (2013)	A malware scanner that has the ability to spot webshells in Linux operating system. It uses signature and statistical feature matching for the detection of webshells. It updates its signature database not only by user submissions but also by regular check of open data sources such as antiviruses.	p13, p42
	VirusTotal	VirusTotal (2012)	An online services that enables scanning files for malwares using more than 50 antivirus systems. The suspiciousness of files can be determined depending on the number of antiviruses reporting the file as suspicious.	p10, p13, p31, p36
	Loki	GmbH and Loki (0000)	A free and simple tool for compromise and detection. It provides a support of different detection techniques: hash based detection of known webshells, Yara semantic rule matching and signature based matching.	p10, p36
	BackdoorMan	Backdoorman (2016)	A Python open source toolkit for malicious PHP webshells. It uses different methods and technologies: UnPHP for code deobfuscation, shells signature database, external services such as VirusTotal and shellray PHP webshell detector.	p13
	WebshellKiller	Webshellkiller (2019)	A web backdoor killing tool that incorporates multiple detection engines and different detection methods: signature matching, machine learning, simulation execution, dynamic analysis and monitoring.	p19, p34, p38
	Webdir+	Webdir+ (2021)	Online webshell detection engine that uses advanced dynamic monitoring technology and combines three engines to detect and kill different kinds of Trojans.	p4, p6
	AVAST	Avast (1995)	Free antivirus engine that uses 6 layers of deep protection including: signature matching, machine learning, runtime monitoring and emulators. Incorporated database is automatically updated whenever a malware is reported by an AVAST user in the network.	p38
	360 Total Security	Total security (2014)	A free malware scanner that uses five different antivirus engines including a cloud based heuristic engine. It also incorporates a machine learning engine to monitor activities and ensure that threats are captured even before a common definition is established.	p6, p12, p38
Helper	NeoPI	NeoPI (2014)	Statistical analyzer of text and script files written in Python. It ranks files based on their contents regarding five measured statistical values: information entropy, longest word, coincidence index, signature, and compression ratio.	p31

planation of their structures or accuracies. Chaitin Cloud-Walker (2020) fits to this category.

3. **Taint checkers:** these tools use taint analysis to detect any injection vulnerability pattern in source codes and warn that a script is using potentially dangerous tainted variables. For this sake, taint checkers parse codes by traversing abstract syntax trees and identifying the information flow of untrustworthy inputs that may have potential security risks (Kurniawan et al., 2018).
4. **Sandboxies:** sandboxing is a security mechanism enabling isolate execution of suspicious programs in order to mitigate actual failures and attacks from spreading. For web-

shell detection, suspicious scripts are isolated and executed in virtual platforms aiming to detect any suspicious behaviors that cannot be detected through static analysis.

5. **Hybrid detection tools:** to increase the detection rate, hybrid tools use either different techniques or different technologies and engines. The effectiveness of such tools depends on the individual effectiveness of their incorporated techniques, technologies and engines.
6. **Helper tools:** some tools do not fit to any of the above categories. Instead, they are used as part of other tools for the detection of webshells. Specifically, NeoPI (2014) is not a detection tool by itself, it ranks files based on five statistical values. It is up to the user to determine some thresholds or manually investigate files to validate the maliciousness of webshells. Therefore, NeoPI can be used in conjunction with other established detection methods. Effectively, NeoPI has been used in **p2** for the extraction of statistical features that are combined with other opcode sequence features to form input feature vectors. However, NeoPI is found performing better with obfuscated rather than clear scripts (Qi et al., 2018) and suffering from high false positive rates (Tu et al., 2014).

In addition to the aforementioned tools used for validation, this review identified 7 new tools (17%):

- **p6 [Hybrid] -** Webshell Detector (WSLD) (Zhao et al., 2020) is a prototype system implemented for runtime detection and prevention of webshell attacks. The prototype combines three heuristic methods (fuzzy hash similarity, signature matching and statistical feature based similarity) and deep learning. A synchronized webshell feature library is used by heuristic methods. Input scripts are firstly classified into high risk and suspicious webshells using the three different heuristic models. An input script is considered of high risk if it is detected by one of the three heuristic models as a webshell, otherwise the script is classified as suspicious. Suspicious scripts are preprocessed by extracting all their system call sequences and made subject for a second round detection using LSTM. The webshell is isolated and preprocessed by a cloud analysis module to update the feature library.
- **p8 [Hybrid] -** GuruWS (Le et al., 2019) is a hybrid platform designed for the detection of both malicious webshells and web application vulnerabilities. The runtime environment of GuruWS incorporates two distinct modules: *grMalwrScanner* for webshells detection and *grVulnScanner* for web application vulnerabilities detection. The *grMalwrScanner* module performs taint-analysis for simple PHP scripts to determine dangerous function calls and their arguments; for more complex scripts, YARA rules are used for matching malicious scripts. Moreover, a statistical based analysis (i.e.; ranking files based on 5 statistical features) is optionally provided to users. The *grVulnScanner* module extends the THAPS engine (Jensen et al., 2012) to perform web application vulnerabilities detection such as injection and Cross-Site Scripting vulnerabilities.
- **p13 [Sandboxie] -** PHPMalScan (Naderi-Afooshteh et al., 2019) is designed as a malware detection tool including webshells. The proposed tool follows the Cubismo (Naderi-Afooshteh et al., 2019) (**p14**) technique in analyzing PHP scripts. Therefore, PHPMalScan explores every possible path using counterfactual execution of code in a sandbox and virtual environment. Sandboxes may cooperate with each other by sharing artifacts needed for their executions. For the detection of malwares (e.g.; webshells), PHP functions are classified into safe and potentially malicious, and two metrics are proposed based on the number and intensity of malicious functions: maliciousness score (MS), and potentially malicious functions ratio (PMFR). Thresholds are used to determine the maliciousness of scripts based on the values of MS and PMFR respectively.
- **p30 [Database Matcher] -** A webshell classification tool is proposed in Wrench and Irwin (2016). The proposed tool uses similarity analysis to detect derivatives of well-known webshells. Following the proposed method, PHP scripts need to be decoded first in order to reveal any deobfuscation layer of code. Second, all user-defined function names and bodies are extracted by analyzing PHP script tokenization. Third, scripts are fuzzy hashed and stored within the source files. Finally, similarity matrices of function names, function bodies, file hashes are generated in request. Visualization tools such as heatmaps, dendograms are incorporated and used to discover the similarity among samples.
- **p32 [Helper] -** A search software for ASP webshells is proposed in Mingkun et al. (2012). The proposed tool has the ability to recognize several features of ASP webshells and report suspicious files to the administrator for further examination. Specifically, the tool recognizes calls to specific ASP component and functions, suspicious statements and customized encryption functions. The tool is a language dependent and follows a semi-automatic approach for the detection of webshells.
- **p43 [Sandboxie] -** A sandbox based environment is proposed in Wrench and Irwin (2014) for the static and dynamic analysis of PHP scripts in the aim to semi-automatically detect webshells. In the proposed environment, PHP shells are firstly deobfuscated and normalized. Deobfuscated scripts are statically analyzed for speciousnesses. Malicious scripts are indexed and saved in a database and safely executed in a sandbox environment for behavioral and dynamic analysis. Sandbox execution enables reporting calls to exploitable functions (i.e.; command and code execution, information disclosure and filesystem functions) and where they were called. The proposed environment is found failing in the execution of PHP files incorporating other scripts such as javascripts and CSS. Moreover, the deobfuscation process is restricted to *eval()* and *preg_replace()* functions with explicit string arguments which restricts it detection ability to specific kinds of webshells.
- **p44 [Database Matcher] -** Web Shell Scanner (WSS) (Jeong et al., 2014) uses two main databases. A first database (MDB) for system call functions and commands used by web programming languages, and a second database (SDB) for common keywords used in webshells. WSS performs two kinds of pattern matching, it uses MDB to match 34 keywords and hence detect well-known malicious webshells, and uses SDB to match 51 system function calls and commands to detect suspicious webshells.

Table 14 – Available datasets for webshell detection.

dataset	year	available at	type	cited/used in
CNTC-2017	2017	https://www.kesci.com/	Payloads	p11
Li et al. (2019a)	2019	GitHub - leett1/Programe/	Webshells	p9
WebSHArk 1.0	2015	Available in request from the authors of p42	Webshells	p42, p44

WSS also performs a web log analysis to trace the effects of webshells and check the accuracy of the detection. The proposed tool has a limitation due to its dependency on databases with small number of patterns. This makes the tool unable to detect all real world webshells.

3.4. *Is there any publicly available datasets for webshells? (RQ4)*

Availability of datasets plays a foremost role in identifying malicious webshells, recent coding techniques and validating proposed solutions. Few ready to use datasets incorporating both malicious and benign webshells are found available in the literature as described in Table 14.

- **CNTC-2017:** is a collocation of payloads that are gathered from the Internet. The dataset is provided through the 2017 China Network Security Technology Competition. The dataset contains 1104 normal payloads and 528 webshell payloads.
- Li et al. (2019a) dataset: Li et al. (2019a) used a collection of malicious and benign webshells written in PHP, ASP, ASPX and JSP. The dataset is made available to users. Webshell sources are collected from Github. The dataset constitutes of 7400 benign and 4500 malicious webshells.
- **WebSHArk 1.0:** Kim et al. (2015) proposed WebSHArk 1.0 as a benchmark dataset for malicious webshell detection. The dataset includes 13810 multi-language webshells in which 809 are malicious and 13001 are benign webshells. The included samples are collected from different sources and de-duplicated using MD5 based similarity and made ready to use by request from the authors.

This review also identified several publicly collections available as open source projects. Those collections are used by identified studies for the validation of their proposals. Tables 15 and 16 summarize respectively the set of benign and malicious webshell collections together with lists of studies citing them. Most of those collections contain an arbitrary number of webshells and a single webshell can also be present with several copies or versions. Therefore, pre-processing steps such as language based filtering and de-duplication are required prior to their exploitation. Furthermore, Huang et al. (2020) (**p2**) performed a manual analysis of malicious webshell collections from GitHub and found that some of them include benign webshells.

3.5. *Webshell detection V&V methods (RQ5)*

We distinguish the use of several verification and validation methods used by primary studies. These techniques are shown in Table 17 with their adoption frequencies. Most studies used several methods for the verification and validation of their solution proposals.

- **Test multiple classifiers:** this verification technique is mainly adopted by machine learning based solutions. Different classifiers are trained and tested using the same dataset. The performance of the proposed model is compared with those of other classifiers. This approach is the most adopted verification technique due to the large number of machine learning based solutions 71% (see RQ3).
- **Comparison with existing tools:** the set of tested webshells are passed through existing tools (see Table 13). Afterwards, the detection rate of the proposed solution is compared with the reported rates from selected tools.
- **Comparison with similar works:** results of the proposed solutions are compared with the results obtained by applying other proposals in similar works. This is performed either by making use of available implementations of other solutions or by self-implementing based on their descriptions in published papers.
- **Runtime analysis:** the response time of proposed solutions is compared with other tools or similar works to check the suitability of proposed solutions to runtime detection or to low resource-environments.
- **Test case based validation:** a test case describes a particular execution scenario by specifying inputs, execution conditions, and expected results. For webshell detection, webshell attacks are simulated, modeled as test cases and applied to the proposed solution platforms or tools. Test cases are generally used to verify specific software requirements. This method is known to be unable to identify all the fails of systems unless all potential scenarios are described by test cases.

Table 15 – Malicious webshell collections.

Source	Available at	Studies
GitHub	tennc/webshell	p1, p6, p9, p8, p10, p12, p15, p25, p28, p29, p33, p34, p36, p38
	JohnTroony/php-webshells	p1, p8, p10, p15, p19, p25, p29, p33, p36
	ysrc/webshell-sample	p6, p12, p15, p19, p20, p34, p36
	xl7dev/webshell	p6, p12, p15, p19, p34, p36
	tutorial0/webshell	p8, p12, p19, p34
	bartblaze/PHP-backdoors	p15, p20, p36
	BlackArch/webshells	p8, p15, p34
	nikicat/web-malware-collection	p25, p29, p36
	fuzzdb-project/fuzzdb	p8, p15
	lcatro/PHP-webshell-Bypass-WAF	p15
	linuxsec/indoxploit-shell	p15
	b374k/b374k	p15
	LuciferoO/webshell-collector	p15
	malwares/webshell	p34
	tanjiti/webshell-Sample	p1
	tanjiti/webshellSample	p33
	webshellpub/awsome-webshell	p15
	xypiie/webshell	p34
	lhlsec/webshell	p8
	http://insecurety.net	p30, p43
Insecurety.net		
Certified Ethical Hacker	https://ceh.vn/[FORUM]	p31
Google code archive	https://code.google.com	p31
Irongeek webshells and RFIs Collection	http://www.irongeek.com	p29
Laudanum collection	https://sourceforge.net/p/laudanum/	p8
VirusTotal online analysis service	https://www.virustotal.com	p30
	http://c99shell.gen.tr	p30
c99shell.gen.tr		
	http://r57shell.net	p30
r57shell.net		
r57.gen.tr	http://r57shell.net/	p30
hoco.cc	http://hoco.cc/	p30

Only seven studies have performed runtime analysis of their proposed solutions. Table 18 shows the best average runtime test per sample of each of those studies. The presented values in Table 18 only provide an overview of the runtime performance of proposed solutions. These values mainly depend on (1) the number and size of samples in test datasets used for each study, and (2) the hardware configuration used to perform the runtime analysis. As expected, the cost detection time of **p6** and **p24** are the highest due to the adoption of deep learning models. These models are known to be expensive in terms of computation. The cost detection time of **p6** is also higher due to the adoption of code analysis where all the executable statements of each sample are explored which is also a time consuming task.

4. Research issues and challenges

In this section, we highlight and discuss the set of challenges and issues related to the detection of webshells observed from primary studies of this review. We also identify general research directions to tackle them.

4.1. *Interpretability issue of machine learning based models*

This review revealed a bias toward the adoption of machine learning based techniques for the detection of webshells, 71% of identified solution proposals are machine learning based models. Although the capability and the widespread use of such technology, machine learning have widely been criti-

Table 16 – Benign webshell collections.

Source	Link	Studies
GitHub	WordPress/WordPress	p6, p15, p25
	yiisoft/yii2	p1, p25, p33
	phpbb/	p15, p31, p36
	WordPress	p1, p33
	johnshen/PHPcms	p1, p33
	joomla/joomla-cms	p6, p15
	laravel/laravel	p15, p25
	learnstartup/4tweb	p1, p33
	phpmyadmin/phpmyadmin	p15, p25
	rainrocka/xinhu	p1, p33
	octobercms/october	p6
	alkacon/opencms-core	p6
	craftcms/cms	p19
	croogo/croogo	p19
	doorgets/CMS	p19
	phppgadmin/phppgadmin	p15
	smarty-php/smarty	p38
	source-trace/phpcms	p38
	symfony/symfony	p25
	typecho/typecho	p25
Open source projects		
WordPress	https://wordpress.org/	p2, p8, p10, p13, p20, p31, p34, p36, p38
	https://www.phpmyadmin.net/	p2, p13, p34, p38
PHPMyAdmin		
PHPCMS	https://sourceforge.net/projects/phpcms	p2, p34, p38, p39
Joomla	https://www.joomla.org/	p13, p31, p36
Yii	https://www.yiiframework.com	p2, p20, p34
Smarty	https://www.smarty.net	p2, p34
PHPNuke	https://phpnuke.org	p31, p36
	https://www.oscommerce.com	p31, p36
Oscommerce		
CakePHP	https://cakephp.org	p13
	https://www.codeigniter.com/	p20
CodeIgniter		
Alexa	https://www.alexa.com	p31

cized for their lack of interpretability (Gibert et al., 2020; Gilpin et al., 2018; Shirataki and Yamaguchi, 2017). Effectively, machine learning models are often considered as black boxes and their predictions cannot easily be interpreted compared with traditional techniques such as signature and rule based matching. Interpretability of predictions is important to cybersecurity analysts to take appropriate actions. Therefore, machine learning are less suitable for cybersecurity analysts unless more works will be provided investigating their interpretability for malware detection (Gibert et al., 2020). To compromise, we advocate more researches on combining machine learning with traditional techniques. Traditional techniques can be used first and only suspicious files are forwarded to machine learning models. The review, recognized only a single study (**p6**) tested such combination, more investigations are

Table 18 – Cost detection time per sample.

study	avg runtime test (ms)
p5 (model for moderate computing resources)	0.01928
p5 (model for high computing resources)	1.58797
p6	58.5714
p11	2.75813
p13	6.21559
p14	54.1071
p24	71.0657
p44	0.46767

Table 17 – V&V techniques.

technique	studies	#
Test multiple classifiers	p1, p3-p5, p9, p10, p11, p16-18, p22-27, p33, p34, p37-40	22
Compare with existing tools	p2, p4, p6, p8, p10, p12, p13, p15, p19, p31, p34, p36, p38, p42, p44	15
Runtime analysis	p5, p6, p11, p13, p14, p24, p44	7
Compare with similar works	p4, p9, p11, p15, p44	5
Test case based validation	p8, p43	2

required to tackle the interpretability issue of machine learning toward webshell detection.

4.2. *Diversity of webshell languages*

This review also marked a bias regarding specific languages of webshells; 68.18% of identified studies have focused on PHP webshells and a single study (**p32**) was found focusing on ASP webshells. Most of PHP based detection solutions cannot be easily leveraged to other languages. The choice has mainly been justified by the extensive use of PHP language in the development of server-side applications. However, this same reason may encourage hackers to develop more webshells using different languages to bypass existing tools. Effectively, several other languages have been used in coding webshells. Moreover, the recent Web Technology Surveys report[1] indicated that Java, Ruby, and Javascript are the fastest growing server-side programming languages since January 2021. Therefore, more attention need to be paid to other webshell languages, multi-language or language independent based solutions.

4.3. *Selection of distinguishing features*

Different kinds of features have been used to distinguish malicious from benign webshells. Lexical and syntactical features look for predefined patterns or rules. This may be efficient for the detection of already known webshells and their variants. However, the set of patterns or rules require to frequently be updated to follow the rapid evolvement in coding tactics and hence enable the detection of new and more advanced webshells. Semantical features are more meaningful but require experts and domain knowledge to identify them, hence, few of them have been identified and tested. Statistical features are easy to be bypassed by attackers. Attackers may use small webshells, incorporate an extensive number of comment lines and useless variables, rename, split or encode functions, and split webshells into different files. These are examples of bypassing techniques but more can be used getting benefits from available scripting languages' facilities. The use of abstract features may solve some of those problems but not all. Within abstract features, there will be no need to feature engineering. Their extraction is performed with few preprocessing steps or not at all and without the need for human domain knowledge. This reduces the effort and enables the identification of hidden features. Abstract features are suitable for deep learning architectures that can learn directly from raw data. However, experiences show that deep learning models are not suitable for runtime detection and lower resource platforms such as IoT (Yong et al., 2020). Therefore, more attention needs to be paid to the analysis of new webshell coding tactics and the identification of more semantic patterns.

4.4. *Lack of benchmark datasets*

Although the importance of standard datasets, benchmarking has not received as much attention as other types of researches such as detection and analysis. Only a single study (**p42**) has been carried with a primary focus to propose a benchmark dataset for webshell detection. The dataset is not

[1] Web Technology Surveys report: https://w3techs.com/

made available due to security and legal issues as reported by the authors. Several benign and malicious webshell collections can be found in public platforms as described in Section 3.4 and partially or totally used by identified studies after different selection processes. However, final used samples are not made available to readers except **p9**. Moreover, Huang et al. (2020) have examined several malicious webshell collections from GitHub and discovered that they include a considerable number of benign samples. This questions the results obtained by studies who have used those collections. Despite the complexity of manual labeling webshell samples, validated and fair in size datasets need to be available for accurate and fair evaluation of solution proposals. Moreover, those datasets should be extended by new samples to follow the evolvement and get more insights about recent hacking tactics. This goal can be achieved getting benefits from public platforms with versioning facilities such as GitHub[2] or academic repositories such as Mendeley[3].

4.5. *Proper validation techniques*

The review shows that most selected studies compare their solution proposals with existing tools mainly with antivirus engines. Antivirus engines may change their behaviors due to the submission of requests. Effectively, new malware samples can be learned from submitted samples Naderi-Afooshteh et al. (2019). Therefore, comparing new proposed solutions with antivirus engines should be carefully performed. Moreover, comparing with machine learning models requires the presence of original trained models that are not often made available by authors. Training the same classifiers with new datasets is not considered as a viable comparison. Furthermore, performance analysis is very important for checking the suitability of proposed solutions to particular environments such as IoT Yong et al. (2020). However, few studies have performed runtime analysis (17%). Therefore, we advocate incorporating performance analysis in validation techniques and share trained models and tools for further examination and comparison.

5. Threats to validity

This section discusses the validity threats of the present review following the guidelines of Ampatzoglou et al. (2019). Therefore, we report three kinds of threats to validity: threats related to study selection, threats related to data validity, and threats related to research validity.

5.1. *Study selection validity*

For identifying maximum number of relevant studies, well-known academic libraries are queried following the guidelines of Kuhrmann et al. (2017) without restricting to a particular period of time. Early published studies are also considered. Moreover, the automatic search is completed with the conduction of recursive backward and forward snowballing on already identified papers according to the guidelines of Kitchenham et al. (2015). However, due to the adoption of strict inclusion and exclusion criteria, several papers were excluded. Specifically, 27 papers were identified within the snowballing process but excluded with respect to I2 and E4 since they are found written in non-English language. In addition, 4 other studies were excluded according to E2 since more recent and complete versions are published by the same authors and included in the review.

5.2. *Study data validity*

Data required for research questions are extracted manually and reported following a concise template checked by both authors at the planning phase. To avoid any inconsistency in data extraction, the process is conducted separately by both authors and conflicts are discussed and solved.

5.3. *Study research validity*

Both authors are familiar with secondary studies and cybersecurity fields. The first author has been involved in a number of secondary studies as both author and reviewer. A single research validity concerns the generalizability of obtained results. Due to the adopted inclusion and exclusion criteria, the results presented in this study concerns only English and peer-reviewed papers, but not grey literature and non-English papers.

6. Conclusion

This paper presents a detailed overview and state-of-the-art review of existing endeavors in handling webshell attacks. The paper reviews contemporary studies with the primary focus on tacking webshell attacks. The systematic search identified 44 distinct papers classified into analysis, benchmarking and detection. Reviewed papers are summarized, analyzed and compared regarding the details of contributions, webshells of interests, proposed detection methods and adopted validation techniques. The study identified the list of features used to distinguish malicious from benign webshells classified into lexical, syntactical, semantical, statistical and abstract features. Existing tools for the detection of webshells are reviewed and categorized into six different classes: database matching, machine learning, taint checkers, sandboxies, hybrid and helper tools. Datasets and webshell collections available for experimentation are located and validation methods are identified and discussed. The study revealed biases toward PHP webshell detection and machine learning technology. It also identified the limitation of currently adopted features for distinguishing malicious webshells. As a result, the study advocates more analysis and benchmarking studies and adoption of validated datasets and proper validation techniques.

Declaration of Competing Interest

No conflict of interest exists.

[2] GitHub: https://github.com/
[3] Mendeley: https://www.mendeley.com

REFERENCES

Ai Z, Luktarhan N, Zhao Y, Tang C. WS-LSMR: Malicious webshell detection algorithm based on ensemble learning. IEEE Access 2020a;8:75785–97. doi:10.1109/ACCESS.2020.2989304.

Ai Z, Luktarhan N, Zhou A, Lv D. Webshell attack detection based on a deep super learner. Symmetry 2020b;12(9):1–16. doi:10.3390/sym12091406.

Ampatzoglou A, Bibi S, Avgeriou P, Verbeek M, Chatzigeorgiou A. Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Inf. Softw. Technol. 2019;106:201–30. doi:10.1016/j.infsof.2018.10.006.

Avast. 1995. Available online. https://www.avast.com/(accessed jan 2021).

Backdoorman. 2016. https://github.com/cys3c/BackdoorMan.

Clamav. 2002. Available online. https://www.clamav.net/(accessed jan2021).

Cloudwalker. 2020. Available online.https://github.com/chaitin/cloudwalker (accessed jan 2021).

Croix A, Debatty T, Mees W. Training a multi-criteria decision system and application to the detection of PHP webshells. In: 2019 International Conference on Military Communications and Information Systems (ICMCIS). IEEE CS; 2019. p. 1–8. doi:10.1109/ICMCIS.2019.8842705.

Cui H, Huang D, Fang Y, Liu L, Huang C. Webshell detection based on random forest–gradient boosting decision tree algorithm. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). IEEE CS; 2018. p. 153–60. doi:10.1109/DSC.2018.00030.

D-shield. Available online. http://www.d99net.net/(accessed jan 2021).

Deng LY, Lee DL, Chen Y, Yann LX. Lexical analysis for the webshell attacks. In: 2016 International Symposium on Computer, Consumer and Control (IS3C). IEEE CS; 2016. p. 579–82. doi:10.1109/IS3C.2016.149.

Detection M.. R. T. (DART). 2021;Web shell attacks continue to rise. https://www.microsoft.com/security/blog/2021/02/11/web-shell-attacks-continue-to-rise/.

Emposha. 2016;Php web shell detector. https://www.shelldetector.com/.

Fang Y, Qiu Y, Liu L, Huang C. Detecting webshell based on random forest with fasttext. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, ICCAI 2018. New York, NY, USA: ACM; 2018. p. 52–6. doi:10.1145/3194452.3194470.

Felderer M, Carver JC. Empirical research for software security: foundations and experience. In: Ch. Guidelines for systematic mapping studies in security engineering. CRC Press; 2018. p. 47–69.

findbot.pl. 2015. Available online.https://gist.github.com/tonit22/8205047 (accessed jan 2021).

Findwebshell. 2018. Available online. https://github.com/he1m4n6a/findWebshell (accessed jan 2021).

Gibert D, Mateu C, Planes J. The rise of machine learning for detection and classification of malware: research developments, trends and challenges. J. Netw. Comput. Appl. 2020;153:1–22. doi:10.1016/j.jnca.2019.102526.

Gilpin LH, Bau D, Yuan BZ, Bajwa A, Specter M, Kagal L. Explaining explanations: an overview of interpretability of machine learning. In: 5th IEEE International Conference on Data Science and Advanced Analytics (DSAA). IEEE; 2018. p. 80–9. doi:10.1109/DSAA.2018.00018.

GmbH N.S., Loki. open-source ioc scanner. https://www.nextron-systems.com/loki/.

Guo Y, Marco-Gisbert H, Keir P. Mitigating webshell attacks through machine learning techniques. Future Internet 2020;12(1):1–16.

Huang W, Jia C, Yu M, Chow KP, Chen J, Liu C, Jiang J. Enhancing the feature profiles of web shells by analyzing the performance of multiple detectors. In: Proceedings of the 16th IFIP WG 11.9 International Conference on Advances in Digital Forensics XVI. Springer International Publishing; 2020. p. 57–72. doi:10.1007/978-3-030-56223-6_4.

Jensen T, Pedersen H, Olesen MC, Hansen RR. THAPS: automated vulnerability scanning of PHP applications. In: Proceedings of the Nordic Conference on Secure IT Systems. Springer Berlin Heidelberg; 2012. p. 31–46. doi:10.1007/978-3-642-34210-33.

Jeong K, Jung H, You DH, Kim J. Web shell scanner (WSS): a high-performance detection tool for malicious web shells. Int. J. Appl. Eng. Res. 2014;9(22):14619–32.

Jinping L, Zhi T, Jian M, Zhiling G, Jiemin Z. Mixed-models method based on machine learning in detecting webshell attack. In: Proceedings of the 2020 International Conference on Computers, Information Processing and Advanced Education. ACM; 2020. p. 251–9. doi:10.1145/3419635.3419716. CIPAE 2020

Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers; 2017. p. 427–31.

Kang W, Zhong S, Chen K, Lai J, Xu G. RF-AdaCost: webshell detection method that combines statistical features and opcode. In: Proceedings of the 3rd International Conference on Frontiers in Cyber Security, FCS 2020. Singapore: Springer Singapore; 2020. p. 667–82. doi:10.1007/978-981-15-9739-8_49.

Kim J, Yoo DH, Jang H, Jeong K. Webshark 1.0: a benchmark collection for malicious web shell detection. J. Inf. Process. Syst. 2015;11(2):229–38. doi:10.3745/JIPS.03.0026.

Kitchenham BA, Budgen D, Brereton P. Evidence-Based Software Engineering and Systematic Reviews. Chapman & Hall/CRC 2015.

Kuhrmann M, Fernández DM, Daneva M. On the pragmatic design of literature studies in software engineering: an experience-based guideline. Empir. Softw. Eng. 2017;22(6):2852–91. doi:10.1007/s10664-016-9492-y.

Kurniawan A., Abbas B.S., Trisetyarso A., Isa S.M.. Static taint analysis traversal with object oriented component for web file injection vulnerability pattern detection, Procedia Comput. Sci.2018. 135, 596–605, 10.1016/j.procs.2018.08.227

Kurniawan A, Abbas BS, Trisetyarso A, Isa SM. Classification of web backdoor malware based on function call execution of static analysis. ICIC Express Lett. 2019;13(6):445–52. doi:10.24507/icicel.13.06.445.

Le VG, Nguyen HT, Pham DP, Phung VO, Nguyen NH. GuruWS: A Hybrid Platform for Detecting Malicious Web Shells and Web Application Vulnerabilities. Berlin, Heidelberg: Springer Berlin Heidelberg; 2019. p. 184–208.

Lee T., Ahl I., Hanzlik D.. 2013;Breaking down the China chopper web shell. https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html.

Li T, Ren C, Fu Y, Xu J, Guo J, Chen X. Webshell detection based on the word attention mechanism. IEEE Access 2019a;7:185140–7. doi:10.1109/ACCESS.2019.2959950.

Li W, Zhang Z, Wang L. A dynamic and heterogeneous web application to defense webshell attacks by using diversified PHP code. In: Proceedings of the 4th International Conference on Communication and Information Processing, ICCIP '18. ACM; 2018. p. 107–11. doi:10.1145/3290420.3290438.

Li Y, Huang J, Ikusan A, Mitchell M, Zhang J, Dai R. Shellbreaker: automatically detecting PHP-based malicious web shells. Comput. Secur. 2019b;87:1–11. doi:10.1016/j.cose.2019.101595.

Lian W, Qi F, Dandan S, Qili A, Bin J. Webshell detection based on multi-classifier ensemble model. J. Comput. 2019;31(1):242–52. doi:10.3966/199115992020023101022.

Liu H, Lang B, Liu M, Yan H. CNN AND RNN based payload classification methods for attack detection. Knowl.-Based Syst. 2019;163:332–41. j.knosys.2018.08.036

Lv ZH, Yan HB, Mei R. Automatic and accurate detection of webshell based on convolutional neural network. In: Proceedings of the 15th International Annual Conference on Cyber Security, CNCERT 2018. Springer Singapore; 2019. p. 73–85. doi:10.1007/978-981-13-6621-5_6.

Mingkun X, Xi C, Yan H. Design of software to search asp web shell. Procedia Eng. 2012;29:123–7. doi:10.1016/j.proeng.2011.12.680.

Naderi-Afooshteh A, Kwon Y, Nguyen-Tuong A, Bagheri-Marzijarani M, Davidson JW. Cubismo: Decloaking server-side malware via cubist program analysis. In: Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC '19. ACM; 2019. p. 430–43. doi:10.1145/3359789.3359821.

Naderi-Afooshteh A, Kwon Y, Nguyen-Tuong A, Razmjoo-Qalaei A, Zamiri-Gourabi MR, Davidson JW. Malmax: multi-aspect execution for automated dynamic web server malware analysis. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19. ACM; 2019. p. 1849–66. doi:10.1145/3319535.3363199.

Neopi. Detection of web shells using statistical methods. 2014. https://github.com/CiscoCXSecurity/NeoPI.

Networks R.f.. Linux malware detect. 2013. https://www.rfxn.com/projects/linux-malware-detect/.

Nguyen NH, Le VH, Phung VO, Du PH. Toward a deep learning approach for detecting PHP webshell. In: Proceedings of the Tenth International Symposium on Information and Communication Technology, SoICT 2019. ACM; 2019. p. 514–21. doi:10.1145/3368926.3369733.

OWASP. In: Tech rep. Owasp top 10: The ten most critical web application security risks. OWASP Foundation; 2017. https://owasp.org/www-project-top-ten/

Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE'08, Swindon, UK; 2008. p. 68–77.

Petticrew M, Roberts H. Systematic Reviews in the Social Sciences: A Practical Guide. John Wiley & Sons, Ltd; 2006.

Plan F., Fraser N., O'Leary. J., Cannon V., Read B.. Apt40: examining a China-nexus espionage actor. 2019. https://www.fireeye.com/blog/threat-research/2019/03/apt40-examining-a-china-nexus-espionage-actor.html.

Qi L, Kong R, Lu Y, Zhuang H. An end-to-end detection method for webshell with deep learning. In: 2018 Eighth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC), IEEE CS; 2018. p. 660–5. doi:10.1109/IMCCC.2018.00143.

Rips. 2020. Available online https://github.com/robocoder/rips-scanner (accessed jan 2021).

Shirataki S, Yamaguchi S. A study on interpretability of decision of machine learning. In: 2017 IEEE International Conference on Big Data (Big Data); 2017. p. 4830–1. doi:10.1109/BigData.2017.8258557.

Starov O, Dahse J, Ahmad SS, Holz T, Nikiforakis N. No honor among thieves: a large-scale analysis of malicious web shells. In: Proceedings of the 25th International Conference on World Wide Web, WWW '16. ACM; 2016. p. 1021–32. doi:10.1145/2872427.2882992.

Sun X, Lu X, Dai H. A matrix decomposition based webshell detection method. In: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy, ICCSP '17. ACM; 2017. p. 66–70. doi:10.1145/3058060.3058083.

Systems N.. Php malware finder. 2016. https://github.com/nbs-system/php-malware-finder.

360 Total security J. (2014).available online. https://www.360totalsecurity.com (accessed jan 2021).

Tao F, Cao C, Liu Z. Webshell detection model based on deep learning. In: Proceedings of the 5th International Conference on Artificial Intelligence and Security, ICAIS 2019. Springer International Publishing; 2019. p. 408–20. doi:10.1007/978-3-030-24268-8_38.

Tian Y, Wang J, Zhou Z, Zhou S. CNN-webshell: malicious web shell detection with convolutional neural network. In: Proceedings of the 2017 VI International Conference on Network, Communication and Computing, ICNCC 2017. New York, NY, USA: ACM; 2017. p. 75–9. doi:10.1145/3171592.3171593.

Tianmin G, Jiemin Z, Jian M. Research on webshell detection method based on machine learning. In: 2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE). IEEE CS; 2019. p. 1391–4. doi:10.1109/EITCE47263.2019.9094767.

TitanWolf. Find webshell with grep. https://titanwolf.org/Network/Articles/Article?AID=89dd30a6-6e3d-40ec-9764-74a64f79b22fgsc.tab=0.

Tu TD, Guang C, Xiaojun G, Wubin P. Webshell detection techniques in web applications. In: Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT). IEEE CS; 2014. p. 1–7. doi:10.1109/ICCCNT.2014.6963152.

VirusTotal. Free online virus, malware and url scanner.;https://www.virustotal.com/. 2012

Webshellpub. available online. https://www.shellpub.com/(accessed jan).2021

Wang C, Yang H, Zhao Z, Gong L, Li Z. The research and improvement in the detection of PHP variable webshell based on information entropy. J. Comput. 2017;28:62–8. doi:10.3966/199115992017102805006.

Wang Z, Yang J, Dai M, Xu R, Liang X, et al. A method of detecting webshell based on multi-layer perception. Acad. J. Comput. Inf. Sci. 2019;2(1):81–91.

webdir+ B.. Available online. https://scanner.baidu.com/(accessed jan 2021).

Webshellkiller. 2019. https://github.com/fragileeye/WebshellKiller.

Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA; 2014. p. 1–10. doi:10.1145/2601248.2601268.

Wohlin C. Second-generation systematic literature studies using snowballing. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE '16, Association for Computing Machinery, New York, NY, USA; 2016. p. 1–6. doi:10.1145/2915970.2916006.

Wrench P, Irwin B. Detecting derivative malware samples using deobfuscation-assisted similarity analysis. SAIEE Afr. Res. J. 2016;107(2):65–77. doi:10.23919/SAIEE.2016.8531543.

Wrench PM, Irwin BVW. Towards a sandbox for the deobfuscation and dissection of PHP malware. In: 2014 Information Security for South Africa. IEEE CS; 2014. p. 1–8. doi:10.1109/ISSA.2014.6950504.

Wu Y, Sun Y, Huang C, Jia P, Liu L, Schrittwieser S. Session-based webshell detection using machine learning in web logs. Secur. Commun. Netw. 2019;2019:1–11. doi:10.1155/2019/3093809.

Yang W, Sun B, Cui B. A webshell detection technology based on http traffic analysis. In: Barolli L, Xhafa F, Javaid N, Enokido T, editors. In: 12th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Vol. 773 of IMIS-2018. Springer International Publishing; 2019. p. 336–42. doi:10.1007/978-3-319-93554-6_31.

Yong B, Liu X, Liu Y, Yin H, Huang L, Zhou Q. Web behavior detection based on deep neural network. In: 2018 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE CS; 2018. p. 1911–16. doi:10.1109/SmartWorld.2018.00320.

Yong B, Wei W, Li KC, Shen J, Zhou Q, Wozniak M, Połap D, Damaševičius R. Ensemble machine learning approaches for webshell detection in internet of things environments. Trans. Emerging Telecommun. Technol. 2020:1–12. doi:10.1002/ett.4085.

Zhang H, Guan H, Yan H, Li W, Yu Y, Zhou H, Zeng X. Webshell traffic detection with character-level features based on deep learning. IEEE Access 2018a;6:75268–77. doi:10.1109/ACCESS.2018.2882517.

Zhang Z, Li M, Zhu L, Li X. SmartDetect: a smart detection scheme for malicious web shell codes via ensemble learning. In: Proceedings of the Third International Conference on Smart Computing and Communication, SmartCom 2018. Springer International Publishing; 2018b. p. 196–205. doi:10.1007/978-3-030-05755-8_20.

Zhao Z, Liu Q, Song T, Wang Z, Wu X. WSLD: detecting unknown webshell using fuzzy matching and deep learning. In: Proceedings of the 21st International Conference on Information and Communications Security, ICICS 2019. Springer International Publishing; 2020. p. 725–45. doi:10.1007/978-3-030-41579-2_42.

Zhongzheng X, Luktarhan N. Webshell detection with byte-level features based on deep learning. J. Intell. Fuzzy Syst. 2020:1–12. doi:10.3233/JIFS-200314.

Zhu T, Weng Z, Fu L, Ruan L. A web shell detection method based on multiview feature fusion. Appl. Sci. 2020;10(18):6274. doi:10.3390/app10186274.

Abdelhakim Hannousse is a senior lecturer at Guelma university, Algeria since 2011. He has earned his PhD degree from Ecole des Mines de Nantes, France in 2011, and his Magister degree from Annaba university, Algeria in 2006. He earned a fellowship from UNU-IIST, Macao, China in 2007. His research interests include: cybersecurity, formal methods, software engineering paradigms, architectural styles and data mining.

Salima Yahiouche received the Magister degree on Artificial intelligence from Badji Mokhtar University, Annaba, Algeria in 2006. She is currently working towards her PhD in Grid security at the same university. She is an Assistant Master in the computer science department of Annaba university. Her research interests include: cybersecurity, cloud and grid computing