Journal of Network and Computer Applications 179 (2021) 103009

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches

Hazim Hanif [a,b], Mohd Hairul Nizam Md Nasir [b], Mohd Faizal Ab Razak [c], Ahmad Firdaus [c], Nor Badrul Anuar [d,*]

[a] Department of Computing, Faculty of Engineering, Imperial College London, London, SW7 2AZ, United Kingdom
[b] Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, 50603, Malaysia
[c] Faculty of Computing, Universiti Malaysia Pahang, Pekan, Pahang, 26600, Malaysia
[d] Department of Computer System and Technology, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, 50603, Malaysia

ARTICLE INFO

Keywords:
Software vulnerability detection
Software security
Computer security
Machine learning
Deep learning

ABSTRACT

The detection of software vulnerability requires critical attention during the development phase to make it secure and less vulnerable. Vulnerable software always invites hackers to perform malicious activities and disrupt the operation of the software, which leads to millions in financial losses to software companies. In order to reduce the losses, there are many reliable and effective vulnerability detection systems introduced by security communities aiming to detect the software vulnerabilities as early as in the development or testing phases. To summarise the software vulnerability detection system, existing surveys discussed the conventional and data mining approaches. These approaches are widely used and mostly consist of traditional detection techniques. However, they lack discussion on the newly trending machine learning approaches, such as supervised learning and deep learning techniques. Furthermore, existing studies fail to discuss the growing research interest in the software vulnerability detection community throughout the years. With more discussion on this, we can predict and focus on what are the research problems in software vulnerability detection that need to be urgently addressed. Aiming to reduce these gaps, this paper presents the research interests' taxonomy in software vulnerability detection, such as methods, detection, features, code and dataset. The research interest categories exhibit current trends in software vulnerability detection. The analysis shows that there is considerable interest in addressing methods and detection problems, while only a few are interested in code and dataset problems. This indicates that there is still much work to be done in terms of code and dataset problems in the future. Furthermore, this paper extends the machine learning approaches taxonomy, which is used to detect the software vulnerabilities, like supervised learning, semi-supervised learning, ensemble learning and deep learning. Based on the analysis, supervised learning and deep learning approaches are trending in the software vulnerability detection community as these techniques are able to detect vulnerabilities such as buffer overflow, SQL injection and cross-site scripting effectively with a significant detection performance, up to 95% of F1 score. Finally, this paper concludes with several discussions on potential future work in software vulnerability detection in terms of datasets, multi-vulnerabilities detection, transfer learning and real-world applications.

1. Introduction

The advancement of the technological world rapidly increases the amount of software developed across various platforms. Some software is able to communicate with each other to increase the efficiency of a system. This is further extended with the recent introduction of the 4th Industrial Revolution (IR 4.0), which promotes automation of software systems by allowing them to communicate with each other automatically with less human supervision (Vaidya et al., 2018). However, the implementation of automation into the software system attracts more attackers to hack and gain control of the system. For example, in 2017, software failure caused 1.7 million USD in financial losses, with a cumulative total of 268 years of downtime (Matteson, 2018). In terms of cybercrime, organisations spent an average of 1.4 million USD in 2017

* Corresponding author.
E-mail addresses: m.md-hanif19@imperial.ac.uk (H. Hanif), hairulnizam@um.edu.my (M.H.N. Md Nasir), faizalrazak@ump.edu.my (M.F. Ab Razak), firdausza@ump.edu.my (A. Firdaus), badrul@um.edu.my (N.B. Anuar).

https://doi.org/10.1016/j.jnca.2021.103009
Received 6 August 2020; Received in revised form 19 November 2020; Accepted 29 January 2021
Available online 4 February 2021

and 13 million USD in 2018 to deal with cyber-attacks (Bissell and Cin, 2019). Furthermore, the National Institute of Standards and Technology (NIST) also reported the exponential increase of software vulnerabilities since 2016 (Technology, 2020). This situation raises concerns regarding software security, especially in terms of software vulnerabilities.

The rise in software vulnerability attacks over the years raises the concern regarding the state of the current software vulnerability detection ecosystem. It attracts more researchers to study and devise better approaches in detecting software security vulnerabilities. The existing approaches to detect software vulnerabilities are categorised into two different categories, namely: a) conventional and b) data-mining and machine learning. The conventional approaches consist of: a) static analysis b) dynamic analysis and c) hybrid analysis. However, these conventional approaches suffer from limitations, such as high false positives and high computational costs (Shin and Williams, 2013). Therefore, security researchers have developed a new approach which utilises the predictive power of machine learning. According to Ghaffarian and Shahriari (2017), the data-mining and machine learning approaches are categorised into three main categories: a) vulnerability prediction models based on software metrics b) anomaly detection and c) vulnerable code pattern recognition. The increasing number of works on detecting software vulnerabilities encourages more study and discussion to understand the existing software vulnerability detection ecosystem.

Several studies focus on discussing the conventional and current approaches in software vulnerability detection. Shahriar and Zulkernine (2012) discussed the conventional approaches to detect software vulnerabilities comprehensively by categorising past works into three categories: a) testing b) static analysis and c) hybrid analysis. Furthermore, Kulenovic and Donko (2014) also discussed using static code analysis to detect software security vulnerabilities. Following the application of data mining and machine learning into various fields, including software vulnerability detection, Ghaffarian and Shahriari (2017) proposed a taxonomy of detection approaches based on these new techniques. They categorised the approaches above into four categories, namely: a) software metrics-based b) anomaly detection c) vulnerable code pattern recognition and d) miscellaneous. Similarly, Sarmah et al. (2018) also focused on the vulnerability detection method with the emphasis only on cross-site scripting (XSS) vulnerability.

Following the growing interest of the software vulnerability detection community towards machine learning approaches, especially the breakthrough of deep learning approaches, several studies have discussed the potential of deep learning for vulnerability detection. Li et al. (2019b) performed a comparative evaluation of different deep learning approaches for vulnerability detection, such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) across SARD and NVD datasets. The results showed that code representation with data and control dependency with Bi-directional Recurrent Neural Network achieved the best detection performance. While deep learning approaches are at the peak of interest of the detection community, Zheng et al. (2020) decided to perform a comparative evaluation between traditional machine learning and deep learning approaches in software vulnerability detection. They chose several widely used traditional machine learning algorithms, such as Random Forest, Gradient Boosting Decision Tree and Support Vector Machine, to compare with CNN and RNN algorithms. The evaluation showed that Bidirectional Long Short-term Memory from RNN achieved the best detection performance across all vulnerabilities (buffer errors, resource management errors, improper control of resources). This confirms that deep learning approaches are indeed better at detecting software vulnerabilities as compared to traditional machine approaches. However, even though these approaches are better, there are still gaps that need to be addressed. Lin et al. (2020a) highlighted that there needs a more effective embedding technique that is able to provide richer semantic information for learning purpose. Richer semantic information allows for a better representation of the source code when projected in the feature space. They also highlighted the importance of model interpretability for human understanding so that we are able to make sense of the decision made by the detection model. The current technique visualises the gradient of each token embeddings with respect to the prediction made by the model. Nevertheless, this technique is still lacking in interpretation as it shows only the surface decision made by the detection model without any context. Therefore, there is also in need of more work to look into model interpretability and attribution for software vulnerability detection.

In this study, we approach the topic of software vulnerability detection differently by looking at the growing research interests based on the fundamental research problems and vulnerability detection approaches. This allows us to provide an overview of the current state of research in software vulnerability detection. Therefore, this study presents a taxonomy on software vulnerability detection with an emphasis on the research interests. In addition, this study also presents a taxonomy of machine learning approaches used in the detection of software vulnerabilities, including deep learning techniques. The following are the contributions of this study:

a) Presents a taxonomy of research interests in software vulnerability detection.
b) Presents a taxonomy of machine learning approaches used to detect software vulnerabilities.
c) Highlights the potential future work in software vulnerability detection in terms of datasets, multi-vulnerabilities detection, transfer learning and detection performance.

The remainder of this study is structured as follows. Section 2 provides an overview of the methodology and data analysis approach used in this study. Section 3 presents the taxonomy of research interests in software vulnerability detection. Section 4 presents the taxonomy of machine learning approaches used in the detection of software vulnerability. Section 5 discusses the potential future work in software vulnerability detection. Finally, Section 6 concludes this study with a summary of the work. Table 1 provides a mapping for all the abbreviations used in this study, along with its description.

Table 1
Abbreviations used in this study.

Abbreviation	Description
ANN	Artificial Neural Network
AUC	Area Under the Curve
CFA	Confirmatory Factor Analysis
CNN	Convolutional Neural Network
CPE	Common Platform Enumeration
CVE	Common Vulnerability Exposure
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DNN	Deep Neural Network
FSS	Feature Subset Selection
IR	Industrial Revolution
LSTM	Long Short-term Memory
MLP	Multilayer perceptron
NIST	National Institute of Standards and Technology
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OS	Operating Systems
OSCI	Operating System Command Injection
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SAMATE	Software Assurance Metrics and Tool Evaluation
SARD	Software Assurance Reference Dataset
SMOTE	Synthetic Minority Over-sampling Technique
SPCA	Sparse Principal Component Analysis
SQL	Structured Query Language
SQLI	Structured Query Language Injection
VCC	Vulnerable Code Changes
XSS	Cross-site scripting

2. Software vulnerability detection

Software vulnerability detection is quickly gaining popularity in the research community and industry, especially among those who are in the computer security area. The spike of interest regarding this topic is due to the rise of cybersecurity attacks over the past years, which demand more research in software vulnerability detection. This section presents an overview of the methodology used in this study, including the process of sourcing the research papers and categorising them. Furthermore, this section also discusses the growing research interests in software vulnerability detection based on the collected research papers in terms of research problems and detection approaches.

2.1. Methodology

This study collects and analyses past papers from the year 2011 until 2020 that focus on detecting software vulnerabilities across various problems, programming languages and source codes. We also analyse papers using machine learning approaches to detect software vulnerabilities as this study plans to investigate more into the implementation of these approaches in software vulnerability detection. Fig. 1 illustrates the flowchart of the data collection process.

In the process of obtaining the papers used for this study, we search for relevant research papers from three available databases, which are ScienceDirect, IEEE Xplore and Google Scholar using "software vulnerability" as our main keyword. We choose these databases due to the presence of an abundance of research papers on various topics in these databases, which increases the visibility of research papers in vulnerability detection (Razak et al., 2016). In total, this study found a total of 816,524 records from different types of articles such as journals, conference, books, and news. However, many of these records come from different research areas which are indirectly related to vulnerability detection. To overcome this, we further narrow our findings by using the keyword "software vulnerability detection" and filter any duplicates, unrelated topics and select only journal and conference papers to become our final data.

In total, this study analyses 90 highly relevant and quality experimental papers on software vulnerability detection, which becomes the foundation of this study. Upon finding all the relevant papers from the sources, we read and analyse each paper thoroughly while extracting details from the paper, including problem statement and the detection approach used. The results of our analysis are displayed and discussed in the next subsection.

2.2. Data analysis

The analysis of the collected papers allows this study to categorise the research problems and vulnerability detection approaches into several categories. The categorisation approach allows for a better understanding of the software vulnerability detection paradigm in terms of trending research problems and detection approaches. This study divides the data analysis into two: a) Research problems, b) Vulnerability detection approaches.

a) Research Problems

In this analysis, this study explores and analyses all the research problems to view the current interest in the software vulnerability detection community. Based on the research problems extracted from the papers, we are able to categorise them into several categories based on the fundamental research problems that they are addressing. This categorisation allows for a better understanding of software vulnerability detection paradigm in terms of current research problems. The research problems are divided into five categories, which are methods, detection, features, datasets, code and misc. Fig. 2 visualises the distribution of papers based on different categories of research problems.

Based on Fig. 2, the most significant distribution shows that the majority of the papers, 43.3%, focus on addressing research problems related to the method for vulnerability detection, such as the use of conventional techniques and manual analysis. Meanwhile, the second biggest distribution, 25.6%, belongs to papers that focus on the problem of detection performance of previous approaches or tools. The third biggest distribution, 13.3%, consists of papers that address issues such as the use of old and classic software engineering metrics to detect software vulnerability. The fourth-biggest distribution focuses on papers that address problems related to datasets (8.89%) such as the quality and size of gold standard dataset. The remaining distributions, like code (4.44%) and miscellaneous (4.44%) focus on problems related to code complexity and a combination of several ungrouped research problems. To better understand the growth of the research problem over the years, Fig. 3 visualises the cumulative total of different categories of research

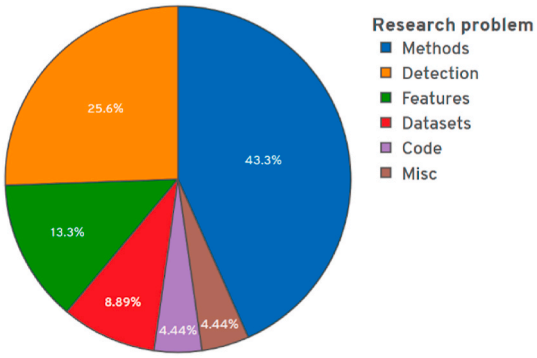

Fig. 2. Distribution of papers collected based on different categories of research problems.

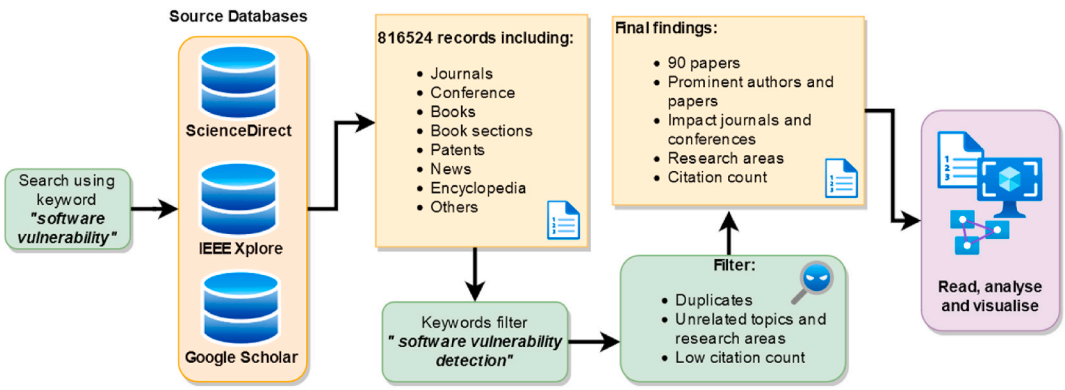

Fig. 1. Flowchart of the data collection process.

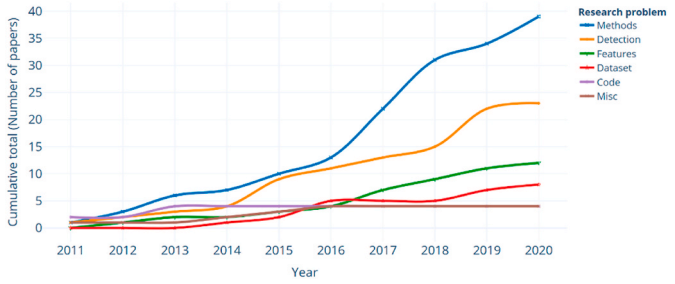

Fig. 3. Cumulative total of different categories of research problems from the year 2011 until 2020.

problems from the year 2011 until 2020.

In general, Fig. 3 shows the increasing interest of methods, detection and features research problems over the years. These rising interests are due to the growing number of papers that address research problems in these categories. For example, in the methods category, a total of 39 papers addressed the problem of manual code analysis and the use of conventional techniques for software vulnerability detection. As for the detection category, a total of 23 papers focused on solving the problem of detection performance as many existing solutions suffer from a high number of false positives and negatives during detection. Some of them also focus on detection class, as binary detection is still widely implemented, as opposed to multiclass detection. In the features category, the interest is a lot less significant than the previous two categories, as only 12 papers tackled the problem of classical metrics and overfitting. Meanwhile, the dataset, code and miscellaneous research problems suffer from only a minor increase over the years, which, in total, ranges between four and eight papers. Therefore, these analyses show that research problems that are related to methods and detection categories are the current focus in most of the work in software vulnerability detection. On the other hand, the research interests in vulnerability detection approaches are also essential to explore.

b) Vulnerability Detection Approaches

In this analysis, this study explores and analyses all the vulnerability detection approaches to view the current interest in the software vulnerability detection community. The approaches are divided into two, which are machine learning and conventional approaches. We divide the approaches into these two categories to show the relevancy of this study and how it fits in the current interest of the software vulnerability detection. Table 2 shows the distribution of papers based on different categories of vulnerability detection approaches, along with its description.

The majority of the works use machine learning approaches as compared to conventional approaches for vulnerability detection. The conventional approaches reflect the traditional solution when dealing with vulnerability detection, while the machine learning approaches reflect the current interest in software vulnerability detection. Fig. 4 visualises the cumulative total of research papers for different categories of vulnerability detection approaches from the year 2011 until 2020.

The trend shows the consistently increasing number of papers that implemented machine learning approach to detect software vulnerability. In total, 67 papers implemented machine learning approaches such as supervised learning, semi-supervised learning, ensemble learning and deep learning from the year 2011 until 2020. Out of the 67 papers, 23 of them implemented deep learning, which is the current interest in artificial intelligence. In terms of the conventional approach, the increasing interest is less significant as the machine learning approach with 23 papers in total. The trends show that existing work in software vulnerability detection made the switch to use the machine learning approach instead of conventional approaches like static, dynamic and taint analysis to detect vulnerabilities. Based on the analysis,

Table 2
Distribution of papers based on different categories of vulnerability detection approaches.

Vulnerability detection approach	Name	Description	Distribution
Machine learning (74.4%)	Deep learning	A type of machine learning technique that uses artificial neural networks as the core foundation in its architecture.	31.11%
	Supervised learning	A type of machine learning technique that learns to perform classification or regression task based on a labelled input-output dataset.	28.89%
	Ensemble-learning	A type of meta machine learning technique that combines multiple different machine learning techniques into a single architecture.	5.56%
	Natural language processing	An artificial intelligence technique able to analyse and make decisions from human language.	3.33%
	Semi-supervised learning	A type of machine learning technique that learns to solve a classification or regression task using a limited number of labelled data.	3.33%
	Regression	A technique that involves predicting a continuous outcome value.	1.11%
	Tree-based	A technique that follows the structure of a tree to perform a prediction task with high stability and accuracy.	1.11%
Conventional (25.6%)	Static analysis	A technique that examines the source code of a program without executing it.	8.89%
	Hybrid analysis	A combination of static and dynamic analysis to examine and analyse a program.	5.56%
	Pattern matching and searching	A technique that involves the matching of specific patterns and parameters using searching techniques and regular expressions.	3.33%
	Graph-based	A technique which involves the analysis of graph structures of a source code such as control-flow graphs and data-flow graphs.	3.33%
	Taint analysis	An analysis technique that identifies the source of users' data and follows the flow of the data throughout the program to make sure it is sanitized before being used.	1.11%
	Dynamic analysis	A technique that examines and analyses the behaviour of the program by executing it.	1.11%
	Formal models	A technique which uses formal methods to perform formal verification of a program.	1.11%
	Statistical analysis	A technique that utilises statistical methods to	1.11%

(*continued on next page*)

Table 2 (*continued*)

Vulnerability detection approach	Name	Description	Distribution
		analyse quantitative data and derive conclusions from it.	

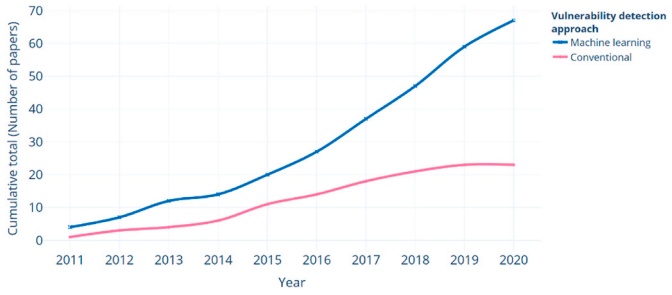

Fig. 4. Cumulative total of different categories of vulnerability detection approaches from the year 2011 until 2020.

machine learning approaches are likely to be used and explored further in the future since the field of artificial intelligence is also advancing rapidly over the years. Therefore, this exploration provides ground for this study to focus on machine learning approaches in software vulnerability detection.

c) Common Vulnerabilities

Throughout the analysis, we found that most existing works targeted specific types of vulnerabilities for detection. This study identifies these vulnerabilities as common vulnerabilities because they are frequently being targeted by vulnerability detection systems. Table 3 presents the common vulnerabilities found in this study.

In this section, this study presents an overview of the methodology used in this study from data collection until data categorisation. In total, this study collected 90 research papers that are related to software vulnerability detection from the year 2011 until 2020. The papers are categorised into several categories based on their fundamental research problem, like methods, detection, features, code, dataset and miscellaneous. Based on the analysis of the collected papers, this study also discusses several interesting statistics in terms of research problems, vulnerability detection approaches and common vulnerabilities, which provides a solid foundation for this study, thus showing its importance to the field of software vulnerability detection.

3. Taxonomy of research interests in software vulnerability detection

Research interest is the main element that motivates researchers to conduct research work on a particular topic. In the previous section, the categorisation process of all the collected papers allows this study to view the research problems as a form of research interest for the research community. Future work will be able to focus more on which problem has less work done, such as problems related to datasets and code. This insight allows the gap between trending and non-trending research problems to be filled and addresses the issues behind vulnerability detection work as a whole. Therefore, this study proposes a taxonomy of research interests in software vulnerability detection based on the fundamental research problems that existing works have addressed. In this study, we discuss six research interests that encourage the work in software vulnerability detection, which are methods, detection, features, code, dataset and miscellaneous. Fig. 5 shows the taxonomy of research interests in software vulnerability detection with its subcategories.

Table 3
Common vulnerabilities found in this study.

CWE ID	Name	Vulnerability	Description
CWE-78	Improper neutralisation of special elements used in an OS command	OS command injection	The program builds all or part of an OS command using externally-influenced feedback from the upstream component but does not neutralise or incorrectly neutralise special elements that may change the intended OS command when it is sent to the downstream component.
CWE-79	Improper neutralisation of input during web page generation	Cross-site scripting	The program does not neutralise or wrongly neutralise the user-controllable input before it is inserted in the output that is used as a web page by other users.
CWE-89	Improper neutralisation of special elements used in a SQL command	SQL injection	The program builds all or part of the SQL command using externally-influenced feedback from the upstream component but does not neutralise or incorrectly neutralise special elements that may change the intended SQL command when it is sent to the downstream component.
CWE-119	Improper restriction of operations within bounds of a memory buffer	Buffer errors	The program runs on a memory buffer, but it can read from or write to a memory location outside the intended buffer boundary.
CWE-120	Buffer copy without checking size of the input	Buffer overflow	The software copies the input buffer to the output buffer without checking that the input buffer size is less than the output buffer size, leading to a buffer overflow.
CWE-190	Integer overflow or wraparound	Integer overflow	The program conducts a calculation that can trigger an integer overflow or wrap-around as the logic assumes that the resulting value will always be greater than the original value. This can contribute to additional vulnerabilities when the equation is used for resource management or execution control.
CWE-306	Missing authentication for critical function	Missing authentication for critical function	The program does not perform any feature authentication that involves a proven user identity or consumes a large number of resources.

The methods category focuses on the methods implemented for vulnerability detection, including manual analysis and conventional techniques. Meanwhile, the detection category discusses the work that concerns with the vulnerability detection performance and the types of

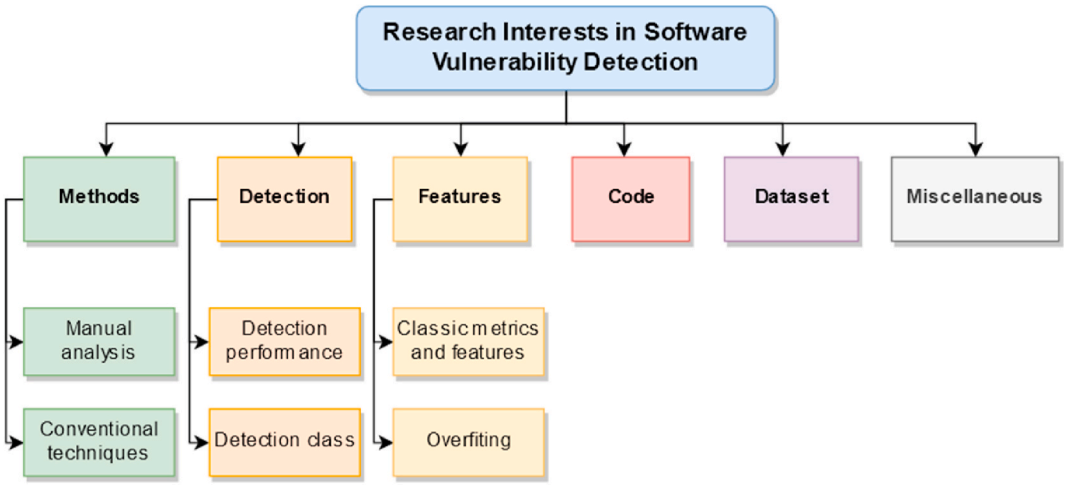

Fig. 5. Taxonomy of research interests in software vulnerability detection.

vulnerability detected. The features category focuses on the usage of classical features and overfitting. On the other hand, the code category explains code activities during the software development phase and the code complexity level. Furthermore, the dataset category discusses existing works that highlight the problem of datasets, including the lack of standard benchmark datasets. Finally, the miscellaneous category gathers all work that focuses on non-major interests into a single group. The six categories of research interests mentioned above highlight the current trends and issues that researchers are focusing on in the area of software vulnerability detection.

3.1. Methods

The methods research interest groups existing works on software vulnerability detection that address the problems related to the methodology. Different researchers adopt different methodologies and approach to achieve their objective, which comes with advantages and disadvantages. Therefore, recent works focus on improving the disadvantages of these methodologies by proposing a better approach. This study divides the methods research interest into two subcategories: a) manual analysis and b) conventional techniques.

a) Manual Analysis

Manual analysis is the most common interest in the domain of software vulnerability detection. The analysis uses a human expert to perform testing and debugging of software during the software development phase for security vulnerability detection. This method requires extensive human resources and is time-consuming for a large-scale software development project and prone to human error. Work to address the problem of manual analysis was started as early as in 1998 by Cowan et al. (1998) and Ghosh et al. (1998). Since then, numerous works have addressed the problem of manual analysis using a variety of different approaches. The problem of this analysis relies on the following weaknesses:

i. Tedious and time limitation

The analysis requires detailed work and may take a longer time in analysing the existing codes. Li et al. (2017b) and Wang et al. (2018) highlight that manual analysis is very time-consuming and requires a tremendous amount of resources to work.

ii. Quality of the detection performance

The quality and performance of detection are also questionable when it comes to manual analysis as different security experts have a different level of knowledge in software security vulnerability (Yamaguchi et al., 2014).

Therefore, manual analysis is often deemed unreliable to be considered as a solution for software vulnerability detection in a large-scale project, which demands a lot of testing and checking. Table 4 summarises the manual analysis problem cited from previous studies.

Conventional techniques are another common research interest in software vulnerability detection. This study defines conventional techniques as any traditional technique used in detecting software vulnerabilities, including static analysis, dynamic analysis, fuzz testing and traditional analysis. These types of techniques are often constituted with low detection performance and produce a high number of false positives (Jingling and Rulin, 2015; Seokmo Kim et al., 2016). The problem of these techniques relies on the following:

i. Accuracy of detection

Russell et al. (2018) and Kronjee et al. (2018) highlighted the disadvantages of static and dynamic analysis that leads to a high percentage of errors and false positives when detecting software vulnerabilities. Similarly, Seokmo Kim et al. (2016) also mentioned the low detection accuracy problem in static analysis techniques for vulnerability detection.

ii. Reliability of the approach

Conventional techniques become less reliable as vulnerabilities keep evolving. Shuai et al. (2015) stated that fuzz testing is unreliable for large scale applications as it is poor at exposing certain types of vulnerabilities as most generated inputs fail to satisfy the syntactic and semantic constraints of the fuzzing system. Moreover, the existing vulnerability prediction model suffers from functionality limitations, such as redundant inspection and lack of training dataset (Yu et al., 2018).

Therefore, this shows that conventional techniques pose a great problem in the area of software vulnerability detection, and new detection techniques are required to address this problem promptly. Table 5 summarises existing works that address conventional techniques as their main interest in detecting software vulnerabilities.

3.2. Detection

The detection research interest groups existing works on software

Table 4
Manual analysis research interest in software vulnerability detection.
b) Conventional Techniques

Reference	Problem	Proposed Approach	Common Vulnerabilities	Dataset
Niu et al. (2020)	Traditional human experts-based approaches are labour-consuming and time-consuming	Deep learning	• Buffer overflow	NVD, SARD
Guo et al. (2020)	Existing solutions heavily rely on human experts to extract features and many security vulnerabilities may be missed	Deep learning	• SQL injection • Cross-site scripting	NVD, SARD
(Li et al., 2020a)	Traditional vulnerability detection methods rely on manual participation and incur a high false positive rate	Deep learning	• Buffer overflow • Resource management error	NVD, SARD
Zagane et al. (2020)	Manual detection of vulnerable code is very difficult and very costly in terms of time and budget	Deep learning	• Buffer overflow • Resource management error	Vuldeepecker (NVD and SARD)
Zhou et al. (2019b)	A challenging and tedious manual process which requires specialised security expertise	Deep learning	Vulnerability information is unavailable.	Linux kernel, QEMU, Wireshark and FFmpeg
Ban et al. (2019)	Most approaches demand experts to locate where the software vulnerabilities are, which requires a lot of manual work and time	Deep learning	• Buffer overflow • Resource management error	FFmpeg, Asterisk, LibPNG and SARD
Li et al. (2019a)	High time consumption due to manual code auditing and analysis	Deep learning	Vulnerability information is unavailable.	QEMU, ImageMagick, multidiff, lrzip, KLEE, vim, LibTIFF 4.0.6
Liu et al. (2019)	Manual code analysis is time-consuming and labour-intensive	Deep learning	• Buffer overflow	Common Vulnerability Exposure (CVE), FFmpeg 0.6, LibTIFF 4.0.6
Li et al. (2018b)	Relies on human experts to define features and often misses many vulnerabilities	Deep learning	• Buffer overflow	National Vulnerability Database (NVD), Software Assurance Reference Dataset (SARD), National Institute of Standards and Technology (NIST)
Li et al. (2018a)	Human experts have to manually define features for vulnerability detection model	Deep learning	• SQL injection • OS command injection • Buffer overflow • Cross-site scripting	NVD, SARD
Liu et al. (2018)	Unpractical to detect and exploit vulnerabilities by manual construction	Dynamic analysis	• Buffer overflow	20 binaries from Robot Hacking Game competition
Wang et al. (2018)	Vulnerability analysis and confirmation rely on security researchers' manual analysis	Pattern matching	• Cross-site scripting	TOM mail and Winmail projects
Jurn et al. (2018)	Current method relies on manual analysis resulting in slow response speed	Hybrid analysis	• Buffer overflow • OS command injection	Lighttpd, libhttpd, Wsmp3d, Kritton
Gawron et al. (2017)	Current situation demands manual interaction in the classification process.	Supervised learning	• SQL injection • OS command injection • Buffer overflow • Cross-site scripting	CVSS, CPE
Wu et al. (2017)	Current analysis tool based on tedious manual auditing, which requires a considerable amount of expertise and resources.	Deep learning	Vulnerability information is unavailable.	Private dataset by analysing 9872 binaries in 32-bit Linux OS
Yulei Pang et al. (2017)	Manual patching of vulnerabilities is very time-consuming and expensive	Deep learning	Vulnerability information is unavailable.	BoardGameGeek, Connectbot, CoolReader, AnkiDroid
Han et al. (2017)	Requires expert knowledge to manually determine intricate vulnerability metrics	Deep learning	Vulnerability information is unavailable.	CVE
Ma et al. (2017)	Expensive and impractical to manually create specific template or repair rules for all kinds of vulnerabilities	Tree-based	• SQL injection	48 Java applications with more than 400 stars from Github
Li et al. (2017b)	Manual auditing of codes is very difficult and time-consuming	Statistical analysis	• Buffer overflow • SQL injection	Common Weakness Enumeration (CWE), CVE, NVD
Medeiros et al. (2016)	Requires explicit coding knowledge about how to discover each kind of vulnerability	Supervised learning	• SQL injection • Cross-site scripting	Private dataset
Yamaguchi et al. (2015)	Require a security expert to manually model, manually audit and specify appropriate patterns in practice	Graph-based	• Buffer overflow	Linux kernel 3.11.4, OpenSSL 1.1.0, Pidgin 2.10.7, VLC 2.0.1, Poppler 0.24.1
Yamaguchi et al. (2014)	Requires manual auditing of source codes and significant expert knowledge	Graph-based	• Buffer overflow	Linux kernel in 2012
Yamaguchi et al. (2013)	The vast majority of vulnerabilities is still identified by tedious manual auditing of source codes	Natural language processing	• Permissions, Privileges, and Access Control (CWE-264)	Firefox 4.0, Linux 2.6.34.13, LibPNG 1.2.44, LibTIFF 3.9.4, Pidgin 2.7.3
Yamaguchi et al. (2012)	Requires tedious manual auditing of source codes, which requires a lot of time and expertise	Natural language processing	• Buffer overflow • OS command injection	LibTIFF 3.8.1, FFmpeg 0.6, Pidgin 2.10, Asterisk 1.6.1

vulnerability detection that address the problems related to the detection aspects of the vulnerabilities. Detection aspect is one of the important problems in software vulnerability detection as it is the main aim of the research. Different approaches target different detection issues, such as performance, reliability and class. This study divides the detection research interests into two subcategories: a) detection performance and b) detection class.

a) Detection Performance

Detection performance concerns about the rate of false positives and false negatives that occur during the detection of software vulnerabilities. This is very important as a high rate of misclassifications shows that the proposed approach is unreliable for the detection of software vulnerabilities. The problem of detection performance is widely known in computer security fields, such as in malware and opinion spams

Table 5
Conventional techniques research interest in software vulnerability detection.

Reference	Problem	Proposed Approach	Common Vulnerabilities	Dataset
Fidalgo et al. (2020)	Gap in detecting PHP vulnerabilities using machine learning	Deep learning	• SQL injection	SARD
Kronjee et al. (2018)	Static and dynamic analysis requires knowledge of the whole code structure and rule sets	Supervised learning	• SQL injection • Cross-site scripting	Software Assurance Metrics and Tool Evaluation (SAMATE), NVD
Yu et al. (2018)	Existing vulnerability prediction model has functionality limitations	Supervised learning	Vulnerability information is unavailable.	Firefox, Mozilla Security Advisories blog, Bugzilla
Grieco & Dinaburg (2018)	Hard to obtain optimal configuration for a vulnerability detection tool	Supervised learning	Vulnerability information is unavailable.	288 DARPA Cyber Grand Challenge (CGC) binaries
Russell et al. (2018)	Existing tools detect a limited subset of possible errors based on pre-defined rules (static analysis)	Deep learning	• Buffer overflow	Static Analysis Tool Exposition (SATE) IV test suite, Debian Linux, ManyBugs, public Github repositories
Kim et al. (2017)	Lack of scalability prevents software developers from readily managing code clones and associated vulnerabilities	Static analysis	• Buffer overflow	Github repositories, firmware of Android smartphones
Catal et al. (2017)	Existing approaches rely on the conventional offline analysis tool	Supervised learning	• OS command injection • Cross-site scripting	Drupal, Moodle, PHPMyAdmin
Seokmo Kim et al. (2016)	Static analysis produces low accuracy and many false positives	Hybrid analysis	• SQL injection	Private dataset
Grieco et al. (2016)	Static analysis suffers from high false positives and is time-consuming needed for large scale applications	Supervised learning	• Buffer overflow	Private dataset called VDiscovery based on test case from Debian Bug Tracker
Jingling & Rulin (2015)	Static and dynamic analysis suffer from high number of false positives	Hybrid analysis	• SQL injection • Cross-site scripting	110 small hand-written PHP programs
Shuai et al. (2015)	Fuzz testing is poor at exposing certain vulnerabilities, as most generated inputs fail to satisfy syntactic or semantic constraints	Static analysis	• Buffer overflow	Winamp 5.5.7, MP3 CD Ripper 2.6, Storm 3.10.4
Lwin Khin Shar and Tan (2013)	Existing approaches locate vulnerable code only at software components or file levels	Supervised learning	• SQL injection • Cross-site scripting	Schoolmate 1.5.4, FaqForge 1.3.2, WebChess 0.9.0, Utopia News Pro 1.1.4, Yapig 0.95b, PHPMyAdmin 2.6.0, PhpMyAdmin 3.4.4, CuteSite 1.2.3
Lekies et al. (2013)	The increasing number of client sides vulnerabilities that challenges both static and dynamic security testing approaches	Taint analysis	• Cross-site scripting	Private dataset
L. K. Shar and Tan (2012)	Existing solution that implements conditional branching ignores control flow and thus prone to errors	Supervised learning	• SQL injection • Cross-site scripting	Geccbblite 0.1, Schoolmate 1.5.4, FaqForge 1.3.2, WebChess 0.9, Utopia News Pro 1.1.4, Yapig 0.95b, PHPMyAdmin 2.6.0

detection (Afifi et al., 2016; Hazim et al., 2018; Razak et al., 2018). Detection performance problem focuses upon the following objective:

i. Rate of false positives and false negatives

False-positive occurs when a normal sample is falsely classified as vulnerable, while false negative occurs when a vulnerable sample is incorrectly classified as normal. The problem of detection performance and high rate of false positives in vulnerability detection is still being addressed (Chernis and Verma, 2018; Ren et al., 2019), even with the rise of machine learning approach to detect software vulnerabilities. Conventional approaches suffer the most in terms of having detection performance problems. Similarly, Morrison et al. (2015) also highlight the detection performance issues of vulnerability prediction models and whether it is reliable to be used for software development.

Therefore, this shows that detection performance plays an essential role in software vulnerability detection. There are several evaluation metrics reported in the existing works. Table 6 presents the evaluation metrics used in this study.

To further describe this research interest, this study extracts the evaluation result of existing works to show the improvement of their approaches. Table 7 shows the list of current works that address detection performance as their main research problem in detecting software vulnerabilities.

b) Detection Class

Detection class problem focuses on the type of vulnerabilities that are being detected by the proposed approaches. This includes different types of vulnerabilities, web vulnerabilities and zero-day vulnerabilities.

Table 6
The evaluation metrics used in this study.

Evaluation metric	Description
True positive rate	The number of outcomes where the detection model correctly predicts the vulnerable samples.
False positive rate	The number of outcomes where the detection model incorrectly predicts the non-vulnerable samples as vulnerable
Accuracy	The number of correctly predicted samples either non-vulnerable or vulnerable.
Precision	Measures the ability of the detection model to present relevant prediction. The higher the score, the higher its ability to present relevant prediction. In this study, the relevant prediction is predicting if a vulnerable sample is vulnerable.
Recall	The rate of correctly predicted samples as vulnerable. Since the objective of the study is to detect vulnerable samples, the objective of the evaluation is also focusing on vulnerable samples.
F1	The test for accuracy that considers both recall and precision when calculating the score. It calculates the harmonic average of both metrics. F1 score helps to determine how precise and robust is the detection model in predicting samples correctly while reducing false prediction.

Certain existing works focus only on several or limited types of vulnerabilities, making granular level analysis harder for security researchers. Most of the existing works focus on doing binary detection, which predicts whether a sample is either vulnerable or non-vulnerable without knowing what type of vulnerability it is. In terms of multiclass detection, it allows the detection model to predict different types of vulnerability based on its class. However, the problems in detection class are the following:

Table 7
Detection performance research problem in software vulnerability detection.

Reference	Problem	Detection performance	Common Vulnerabilities	Dataset
Li et al. (2020b)	Current detectors still offer inadequate detection capability and inadequate locating precision	Accuracy up to 96%	• Buffer overflow	SARD, NVD
Tian et al. (2020)	Existing works suffer deficiencies such as high false positives rate	F1 scores up to 89.9% with precision of up to 91%	• Buffer overflow • Numeric handling	SARD
Duan et al. (2019)	Some vulnerable and non-vulnerable code is hardly distinguishable, resulting in low detection accuracy.	F1 scores up to 80.6	• Buffer overflow • Resource management error	SARD
Ren et al. (2019)	Existing techniques suffered from low testing efficiency and accuracy	Accuracy up to 82.55%	• Buffer overflow	NIST (Juliet test suite)
Saccente et al. (2019)	Software security vulnerability detection tools suffer from high false positive reports	Accuracy up to 92.1 average across different vulnerabilities	Multiple vulnerabilities (29 types)	SARD
Chernis & Verma (2018)	Difficult to scan for bugs thus leads to low detection performance	Accuracy up to 75%	• Buffer overflow	NVD, Github, Top 20 Linux utilities from ftp.gnu.org
Pechenkin & Demidov (2018)	Existing works suffer from low accuracy of detection	Accuracy up to 92%	Vulnerability information is unavailable.	Private dataset
Phan et al. (2017)	Existing features and tree structures often fail to capture the semantics of program	Accuracy gain up to 11% compared to tree-based approach	Vulnerability information is unavailable.	CodeChef
Li et al. (2017a)	Previous methods less accurate during detection	Accuracy ranges from 69% to 100% across different vulnerabilities	• SQL injection • Cross-site scripting	Private dataset
Thomé et al. (2017)	Existing constraint solvers are limited in effectiveness in detecting vulnerabilities	Accuracy ranges from 33% to 100% across different software	• SQL injection • Cross-site scripting	NVD, WebGoat, Roller, Pebble, Regain, Pubsubhubbub-java, Rest-auth-proxy
Saleh et al. (2015)	Existing vulnerability scanners have high false positives	False positives rate of 0–10% across different vulnerabilities	• Buffer overflow • SQL injection • Cross-site scripting	Private dataset
Goseva-Popstojanova and Perhinschi (2015)	Static code analysis tools unable to detect all vulnerabilities in source codes and leads to false positives	Accuracy up to 90%	Multiple types of vulnerabilities	SARD
Morrison et al. (2015)	Current vulnerability prediction models give low prediction accuracy	Precision up to 75%	Vulnerability information is unavailable.	Windows 7 RTM, Windows 8 RTM, CODEMINE repository
Y. Zhang et al. (2015)	Effectiveness of prediction models is still relatively low	F1 scores up to 75%	• SQL injection • Cross-site scripting • OS command injection	Drupal, PHPMyAdmin, Moodle
Perl et al. (2015)	Code metric analysis tools have a very high false-positive rate	Precision up to 60%	• Buffer overflow	66 C and C++ projects that use Git
Scandariato et al. (2014)	The performance of detecting vulnerabilities are low	Precision up to 99% for single Android application	• SQL injection • Cross-site scripting	AnkiDroid 8, BoardGameGeek 8, Connectbot 12, CoolReader 13, Crosswords 17, FBReader 14, etc

i. Multiclass detection of vulnerabilities

Some works focus on detecting certain types of vulnerabilities, such as in Ren et al. (2019), where they aim at detecting Buffer overflow vulnerabilities only. This limits the detection capability of vulnerability prediction models or tools as it is only able to successfully detect one or a certain type of vulnerabilities just like in Zhou et al. (2019a) and Mouzarani and Sadeghiyan (2016).

Table 8
Detection class research problems in software vulnerability detection.

Reference	Problem	Detection Class	Common Vulnerabilities	Dataset
Zou et al. (2019)	Current approaches incapable of multiclass detection	Multiclass	• 40 vulnerabilities	SARD, NVD
Zhou et al. (2019a)	Only a few kinds of vulnerability are covered by the detection tool	Multiclass	• Unused variable (CWE-563), • Use of uninitialised variable (CWE-457), • Use after free (CWE-416)	Private dataset (30 random applications)
Singh et al. (2019)	Traditional security approaches are unreliable in detecting zero-day vulnerabilities	Multiclass	• OS command injection • Missing authentication for critical functions	Apache HTTP server, OpenSSH
Khalid et al. (2019)	Majority of existing vulnerability prediction models unable to detect all type of web vulnerabilities	Single-class	• SQL injection • Cross-site scripting	Public dataset by Walden et al. (2014) for PHPMyAdmin, Moodle and Drupal
Mouzarani & Sadeghiyan (2016)	Most existing methods suggest for detecting one or a limited number of vulnerability classes	Multiclass	• Integer overflow (CWE-190) • Division by zero (CWE-369),	NIST SAMATE test suite collections, Juliet Suite v1.2
Shahmehri et al. (2012)	Users having a problem which type of vulnerabilities can be detected by vulnerability prediction tools	Single-class	• Buffer overflow	Xine-lib-1.1.15
S. Zhang et al. (2011)	The effect of zero-day vulnerabilities on network security evaluation	Regression	Vulnerability information is unavailable.	NVD, CPE, CVSS

ii. Zero-day vulnerabilities

A zero-day vulnerability is a term used to describe a type of vulnerability that has not been discovered. Singh et al. (2019) and S. Zhang et al. (2011) highlights the problem of detecting zero-day vulnerabilities and the importance of having vulnerability detection models that able to predict zero-day vulnerabilities. It allows for a more pre-emptive approach in vulnerability discovery rather than post-production detection based on bug reports.

Therefore, reliable vulnerability detection and prediction models must have the capability to detect different types of software vulnerabilities as zero-day vulnerabilities are unknown and vary from time to time. Table 8 summarises existing works that focus on solving detection class research problems in software vulnerability detection.

3.3. Features

The features research interest groups existing works on software vulnerability detection which address the problems related to the features or metrics used in detecting the vulnerabilities. Features and metrics are regarded as one of the important components in vulnerability prediction models or tools. The reliability and performance of the prediction models greatly depend on the quality of the features or metrics used in the models. This study divides the features research interests into two subcategories: a) classic metrics and features b) overfitting.

a) Classic Metrics and Features

Classic metrics and features highlight the problem of incorporating less meaningful attributes into vulnerability detection models or tools. Consequently, these less meaningful attributes directly affect the detection performance of vulnerability detection models. Classic metrics and features include static code level features, code descriptions and traditional software engineering metrics. The problems in classic metrics and features are:

i. Semantic and syntactic information of the source code

Source codes always contain semantic and syntactic structural information that depends on the programming language. Dam et al. (2018) highlighted that simple code-level features are unable to capture the actual semantic and syntactic representations of the source code. Capturing the true meaning of the source codes is very important as it is able to distinguish between part of codes that are vulnerable and non-vulnerable.

ii. Source code characteristics

Source codes have their own characteristics depending on the granularity level of the source codes. Sultana and Williams (2017) highlight the ineffectiveness of traditional software metrics and the inability of those metrics to specify any particular granularity level of the source codes. Similarly, traditional static attributes also lose information regarding specific source code characteristics (L. K. Shar et al., 2013).

This shows that a high-quality set of features is required as it is able to capture the actual representation of the source codes and distinguish between vulnerable and non-vulnerable source codes effectively. Table 9 describes the list of existing works that focus on the problem of classical metrics and features in detection software vulnerabilities.

b) Overfitting

Overfitting highlights the problem of the high amount of noise in a dataset where it is used to train a vulnerability detection model. In other words, overfitting is the use of models that include more terms than necessary or more complex than it should be (Hawkins, 2004). The

Table 9
Classic metrics and features research problems in software vulnerability detection.

Reference	Problem	Type of Features	Common Vulnerabilities	Dataset
Fang et al. (2019)	Existing models rely on complex graphs generated from source code	• Source code tokens with custom modification (unifying tokens)	• OS command injection • Cross-site scripting • SQL injection	SARD, SQLI-LABS
Liang et al. (2019)	Existing solutions unable to capture syntactic and semantics features of the source code	• Abstract syntax trees • Control flow graphs	Vulnerability information is unavailable.	Private dataset hosted in Github
Lin et al. (2018)	Rely on overly generic hand-crafted features	• Abstract syntax trees	Vulnerability information is unavailable.	LibTiff, LibPNG, FFmpeg, Pidgin, VLC media player, Asterisk
Dam et al. (2018)	Code features unable to capture semantic and syntactic representations of the source code	• Bag-of-Words (BoW) • Source codes embeddings and tokens	• OS command injection • Cross-site scripting	Android applications, Firefox
Sultana (2017)	Existing security metrics at code level have high false-negative rates	• Class-level • Method-level • Micro patterns • Nano patterns	Vulnerability information is unavailable.	Apache Tomcat, Apache XF, Stanford Securibench dataset
Sultana & Williams (2017)	Traditional software metrics are indirectly related to code and unable to specify any particular granularity level	• Class-level • Micro patterns	Vulnerability information is unavailable.	Apache Tomcat 7, Apache Camel, Stanford Securibech dataset
Li et al. (2016)	Instances of the same vulnerability may exist in multiple software copies that are difficult to track in real life	• Static • Component • Expression • Statement • Function-level	• Buffer overflow	NVD, Linux kernel, Mozilla Firefox 36.0, Thunderbird 24.8, Seamonkey, Apache HTTP server, etc.
L. K. Shar et al. (2013)	Static attributes contain less information regarding specific source code characteristics	• Static • Dynamic based on test inputs • Numerical	• SQL injection • Cross-site scripting	SchoolMate 1.5.4, FaqForge 1.3.2, Utopia News Pro 1.1.4, Phorum 5.2.18, CuteSITE 1.2.3, PhpMyAdmin 3.4.4
Hovsepyan et al. (2012)	Existing works focus on derived static features of the source code which loses the semantic information	• Source codes (text mining)	Vulnerability information is unavailable.	K9 mail clients

problem of over-fitting is very popular in machine learning approaches as the model is trained using a dataset supplied into the machine learning algorithm. Firdaus et al. (2017) also investigated the problem of overfitting through several methods of feature selection. Existing works addressed the problem by reducing the dimensionality of the dataset and, thus, reducing the complexity of the vulnerability detection models. Fig. 6 shows three different types of fitness in a machine learning model.

The leftmost illustration in Fig. 6 shows the visualisation of an underfitted machine learning model which is unable to distinguish the underlying pattern of the data successfully. This leads to a low performing model. The rightmost illustration is an overfitted model that occurs when there are noises in the dataset and affects the performance of the vulnerability detection models. Overfitting happens due to some of the features or metrics that are being used as the dataset being unreliable or meaningless. The middle illustration shows the appropriate fitted model, which is able to balance between underfitting and overfitting successfully. It discovers the underlying trends inside the data, which is very beneficial to the model. As such, in any machine learning tasks, the aim is always to achieve an appropriate fitted model which works well across training and testing sets. The problem of overfitting is based on the following:

i. Features space dimensionality

The dimensionality of the features space is the number of features selected to be included inside a detection model. Stuckman et al. (2017) highlighted the problem of overfitting by reducing the number of features used to train the detection model. It shows that, by lowering the number of features, it increases the detection performance and effectiveness of the vulnerability detection models. Similarly, Y. Pang et al. (2015) also perform dimensionality reduction techniques to increase the accuracy and effectiveness of the detection models.

Therefore, techniques like dimensionality reduction and feature selection must be adopted into the vulnerability detection models to reduce the effect of overfitting. Table 10 shows the list of existing works that focus on the problem of overfitting in software vulnerability detection.

3.4. Code

The code research interest groups existing works on software vulnerability detection that addresses the problems related to code activities and code complexity. Code activities are the activity that is related to the source codes, including updating and deleting by software developers. On the other hand, code complexity is about how complex the code is written inside the software. During the software development phase, there are certain functions and processes where developers are able to structure their codes to become simple or complex. This greatly depends on the objective of the software development team. With this, the following are some of the code related issues:

i. Code complexity

Code complexity is known to be greatly related to software security as complex code structures often lead to the presence of vulnerabilities and bugs (Moshtari et al., 2013; Shin et al., 2011). A conventional vulnerability detection tools are less capable in analysing the complex structure of codes efficiently and reliable. Similarly, Shin and Williams (2011) also highlight that there is a lower chance of detecting software vulnerabilities if the code structures are too complex.

ii. Code activity

During software development phases, software developers work in groups and each developer has their own individual way of coding structure. Without proper documentation, the changes made by several developers directly affect the code structure, which becomes inconsistent. This inconsistency often leads to the presence of security vulnerabilities (Meneely et al., 2013).

Therefore, software developers must determine which code structures are the most suitable for specific software development project and allow security researchers to harden the software through suitable hardening techniques. Table 11 shows the list of existing works that focus on code research problems in software vulnerability detection.

3.5. Dataset

The dataset research interest groups existing works on software vulnerability detection that addresses the problems related to the aspect of the dataset used. Dataset is very important in building reliable and effective vulnerability detection models and tools. A properly created dataset is able to help in creating a powerful vulnerability detection model. However, it depends on the type of vulnerability detection work, whether it is cross-project detection, multi-vulnerabilities detection, or binary classification of vulnerabilities. As such, datasets are divided into two types (i.e. public and private). Table 13 shows the list of public and private datasets used in all the research reported in this study. The datasets used in previous studies are both public and private datasets. Each dataset contains different types of information such as binaries, source codes and test suites. However, the problem of dataset is in the following:

i. Gold standard dataset

Dataset is a known problem in software vulnerability detection as there are fewer known gold standard datasets for comparison and evaluation of different detection approaches (Walden et al., 2014). This restricts the work on evaluating the performance and reliability of new approaches as compared to previous approaches. A gold standard dataset is needed to standardise the evaluation of different approaches in detecting software vulnerabilities.

ii. Size of dataset

The performance of a detection model always relies on the size of the dataset. The bigger the size of the dataset, the higher the number of

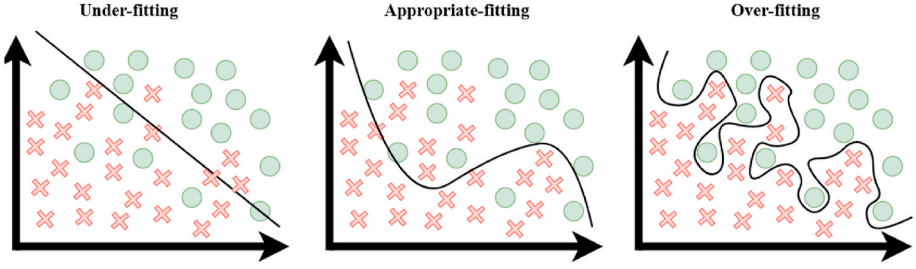

Fig. 6. Model fitness comparison.

Table 10
Overfitting research problems in software vulnerability detection.

Reference	Problem	Overfitting Reduction Techniques	Common Vulnerabilities	Dataset
Huang et al. (2019)	Due to the large amount of vulnerability data and short description, the generated word vector space presents the characteristics of high dimension and sparse	Features space dimensionality reduction using: • Information Gain (IG)	Vulnerability information is unavailable.	NVD
Stuckman et al. (2017)	The number of features used affects the predictive model	Features space dimensionality reduction using: • Feature Subset Selection (FSS) • Entropy Reduction • Principal Component Analysis (PCA) • Sparse Principal Component Analysis (SPCA) • Confirmatory Factor Analysis (CFA)	• OS command injection • Cross-site scripting • Missing authentication for critical function	PROMISE repository, Drupal 6.0, PHPMyAdmin 3.3, Moodle 2.0
Y. Pang et al. (2015)	The high dimensionality of the prediction models affects the accuracy and effectiveness of the prediction technique	Features space dimensionality reduction using: • Statistical test (Wilcoxon)	Vulnerability information is unavailable.	BoardGameGeek, Connectbot, CoolReader, AnkiDroid

Table 11
Code research problems in software vulnerability detection.

Reference	Problem	Code Interests	Analysis approach	Common Vulnerabilities	Dataset
Meneely et al. (2013)	Changes made by the developer on the source codes affect the work of other developers	Activity	Analyses VCCs in terms of: • Size • Interactive churn • Community dissemination	• Cross-site scripting	NVD, NIST, Apache HTTP server
Moshtari et al. (2013)	The complexity of code does not benefit to software security and leads to vulnerabilities	Complexity	Using complexity software metrics to predict vulnerability locations.	• OS command injection • Cross-site scripting	Eclipse 3.0, Apache Tomcat 6.0.35, Mozilla Firefox 2.0.0.5, Linux kernel 2.6.3, OpenSCADA
Shin et al. (2011)	Complex code has less chance to detect its vulnerability	Complexity	Examines execution complexity metrics to determine vulnerable code locations.	Vulnerability information is unavailable.	Mozilla Firefox 3.0, Wireshark 1.2.0, Mozilla Foundation Security Advisories (MFSA)
Shin et al. (2011)	Complex and large code is difficult to test for security and introduces vulnerabilities	Complexity	Examine software metrics: • Complexity • Code churn • Developer activity	Vulnerability information is unavailable.	MFSA, Red Hat Enterprise Linux 4 kernel, Red Hat Bugzilla DB, Red Hat package management system

Table 12
Different types of information that are available in a dataset.

Information	Description
Binaries	The executable file that is produced after a successful compilation of a program's source codes
Bug information	The information and description of bugs that are present in a program
Code gadgets	Fragments of the raw source codes that semantically and syntactically has to do with a vulnerability
CVSS score	Common vulnerability scores which indicate the severity of a specific vulnerability
Peer code review	Results of a peer code review process, which can generate higher quality codes
Software metrics	Software characteristics that are countable, which is used to measure the performance and quality of a software
Source code tokens	Source code that has been parsed by a parsing tool or a compiler
Source codes	The raw source codes of a program taken from the source files in the project repository
Test cases	A synthetic sample of vulnerable and non-vulnerable source codes for testing purposes
Vulnerability information	Description and information about a specific type of vulnerability

samples that the model can learn. In Alves et al. (2016) and Jimenez et al. (2016), they indicate that the dataset used for past evaluations are limited in size and representativeness. Therefore, it is unreliable and unable to be used for the evaluation of newer approaches. In addition, Meng et al. (2016) highlight the problem of having a small amount of labelled data available to train a vulnerability detection model.

This shows that a large, standardised, gold-standard dataset for software vulnerability detection is required so that it is available for evaluation purposes as well as for building a reliable vulnerability detection model. A reliable dataset must consist of real-world vulnerabilities as opposed to synthetic samples and test cases, such as in SARD dataset. However, constructing a reliable dataset requires thorough preparation as the source code must be carefully analysed before labelling it. The current available real-world vulnerability dataset is the NVD. However, the NVD dataset involves the extraction of the source codes from the repository manually. Therefore, the process must be done carefully to avoid any mislabelling when dealing with different repository version and git commits. In the datasets used for software vulnerability detection, there are different types of data, which carry different types of information. Table 12 shows different types of information that are present in a dataset. The presence of different types of information allows this study to distinguish the difference between each dataset. Table 14 highlights existing research works that focus on the

Table 13
List of public and private datasets.

Dataset	Type	Type of Information	Sources
Open source software datasets	• Public • Private	• Binaries • Source codes	• https://sourceforge.net • http://ftp.gnu.org • https://github.com
National Vulnerability Database (NVD)	• Public • Private	• Vulnerability information • Common Platform Enumeration (CPE) • CVSS base scores • Vulnerable product configuration • Weakness categorisation	https://nvd.nist.gov/vuln/data-feeds
Software Assurance Reference Dataset (SARD)	• Public • Private	• Test cases • Source codes • Binaries	• https://samate.nist.gov/SARD • https://samate.nist.gov/SRD/testsuite.php
Android applications	• Public • Private	• Binaries • Source codes	• https://sites.google.com/site/textminingandroid/ • https://f-droid.org/
Draper VDISC	Public	• Source codes	https://osf.io/d45bw/
VulDeePecker	Public	• Source codes • Code gadgets	https://github.com/CGCL-codes/VulDeePecker
μVulDeePecker	Public	• Source codes • Code gadgets	https://github.com/muVulDeePecker/muVulDeePecker
SySeVR	Public	• Source codes	https://github.com/SySeVR/SySeVR
Devign	Public	• Source codes	https://sites.google.com/view/devign
PHP Security vulnerability dataset	Public	• Vulnerability information • Software metrics	https://seam.cs.umd.edu/webvuldata/
VDiscovery dataset	Public	• Test cases	https://github.com/CIFASIS/VDiscover/releases
VulinOSS dataset	Public	• Metrics • CWE • CVE	https://drive.google.com/drive/folders/1rwEd3t6Cbh9Vj2Ayd2CNifQnVKCiVjHB
Transferable Representation Learning dataset	Public	• Source code • Tokens	https://github.com/DanielLin1986/TransferRepresentationLearning
Common Weakness Enumeration (CWE) database	Public	• Vulnerability information	https://cwe.mitre.org/
Common Vulnerability Exposure (CVE)	Public	• Vulnerability information	https://cve.mitre.org/
Static Analysis Tool Exposition (SATE) IV	Public	• Test cases	https://samate.nist.gov/SATE4.html
ManyBugs	Public	• Test cases • Source codes	https://repairbenchmarks.cs.umass.edu/
Stanford Securibench dataset	Public	• Binaries	https://suif.stanford.edu/~livshits/securibench/
PROMISE repository dataset	Public	• Software metrics	http://promise.site.uottawa.ca/SERepository/datasets-page.html
Source codes metrics and vulnerabilities dataset	Public	• Software metrics	https://eden.dei.uc.pt/~nmsa/metrics-dataset/
Gerrit code review	Public	• Peer code review	• http://amiangshu.com/VCC • http://code.google.com/p/gerrit
Mozilla Firefox vulnerability datasets	Private	• Vulnerability information • Bug information • Source codes	• https://www.mozilla.org/en-US/security/advisories/ • https://www.bugzilla.org/ • https://developer.mozilla.org • https://www.st.cs.unisaarland.de/softevo/vulnerabilities.php
DARPA Cyber Grand Challenge (CGC) binaries	Private	• Binaries	https://github.com/trailofbits/ceo/blob/master/install.sh
LAVDNN dataset	Public	• Functions name	https://github.com/StablelJay/LAVDNN

problem of the dataset in software vulnerability detection.

3.6. Miscellaneous

The miscellaneous research interest groups existing works on software vulnerability detection that addresses various aspects regarding vulnerability detection. This includes the problem regarding the exploited vulnerability, resource constraints and software security in general. The problems are as below:

- **Exploitable/Non-exploitable vulnerabilities**: Younis et al. (2016) highlighted the challenge in discriminating between a vulnerability that has exploit with the one that has no exploit. It is known to be challenging due to the similarities that both of these vulnerabilities have. A vulnerability detection model is only effective and reliable when it is able to detect vulnerabilities that are exploitable instead of raising a false alarm. Aiming to overcome this challenge, they proposed an approach to characterise the vulnerable functions using eight software metrics. The approach successfully distinguished between vulnerable and non-vulnerable functions.
- **Limited IT budgets**: Last (2015) expressed the challenge of limited IT budgets and an increasing number of critical software vulnerabilities across systems. Security experts must allocate their limited defence resources carefully by focusing only on the weak areas of their network. Therefore, they proposed an approach to forecast the vulnerability discovery trends using the data from NVD across different software categories (i.e. operating system, video player and web browser). The forecasts allow the security expert to allocate their resources efficiently where required.
- **Low expert resources**: Bosu et al. (2014) discussed the problem of having a low amount of appropriate resources for an expert to successfully identify and fix software security vulnerabilities during the software development phase. They approached this problem by developing a method to analyse the peer code review request for the presence of any Vulnerable Code Changes (VCC). This is very important to allow for the delivery of a secure and error-free product into the production environment. Thus, detecting and fixing software vulnerabilities during the early stage of software development greatly increases the efficiency of securing the software (Mohammed et al., 2017).

Table 14
Dataset research problems in software vulnerability detection.

Reference	Problem	Common Vulnerabilities	Dataset
Lin et al. (2020b)	Previous studies evaluation were done on self-constructed/ collected datasets	Vulnerability information is unavailable.	NVD, SARD
Alves et al. (2016)	Dataset for every past evaluation is limited in size and representativeness	Vulnerability information is unavailable	Mozilla, Httpd, Glibc, Linux kernel, Xen HV
Meng et al. (2016)	Low amount of labelled data available	• Buffer overflow	Nginx 0.6.32, Inspircd 1.1.22, libexif 0.6.2, Popler 0.15.0, Snort 2.40, FFmpeg 0.5.7
Jimenez et al. (2016)	A less reliable and comprehensive comparison of existing studies due to the lack of standardised dataset	Vulnerability information is unavailable	NVD, Linux kernel, Github
L. K. Shar et al. (2015)	In the real world, past vulnerability data is often unavailable or at least incomplete	• OS command injection • Cross-site scripting	SchoolMate 1.5.5, FaqForge 1.3.2, Utopia News Pro 1.1.4, Phorum 5.2.18, CuteSITE 1.2.3, PhpMyAdmin 3.4.4, PhpMyAdmin 3.5.0
Walden et al. (2014)	Many prediction models were evaluated with different methodologies and datasets, thus making it difficult to determine the relative strengths and weakness of different modelling techniques	• OS command injection • Cross-site scripting • Missing authentication for critical function	NVD, Drupal, Moodle, PHPMyAdmin

In this section, this study presents and discusses the taxonomy of research interests in software vulnerability detection. The taxonomy is divided into methods, detection, features, code, dataset and miscellaneous. Each category represents the main research interest highlighted by existing researchers in the area of software vulnerability detection. As such, this study also highlights the current research interests that are considered important, valid and addressable in future work. This taxonomy creates a new perspective in understanding the interests in software vulnerability detection research.

4. Taxonomy of machine learning approaches in software vulnerability detection

Machine learning is a technique that enables computers to learn and work out how to solve problems and perform tasks (Domingos, 2012). It automatically learns from the data provided and it tries to find the relationship between the data. The relationship then permits the machine to execute the assigned tasks very well. In other words, machine learning helps humans to perform specific tasks after learning how to do the task over time. This technique is very efficient and cost-effective when compared to manual programming or rule-based programming. It increases the productivity of a certain activity through the fast learning and execution process. Machine learning is also known as one of the branches in artificial intelligence. Many applications of machine learning are available in various fields including computer security (Feizollah et al., 2019; Firdaus et al., 2019; Hanif et al., 2018).

In this section, this study discusses the machine learning approaches used in software vulnerability detection. The flexibility of the machine learning technique in solving various classification and regression problems permits researchers to incorporate the technique into the domain of vulnerability detection. It is an improvement to the existing conventional vulnerability detection approaches like static analysis, dynamic analysis and fuzz testing. This study proposes a taxonomy of machine learning approaches and divides them into four categories: a) supervised learning, b) semi-supervised learning, c) ensemble learning and d) deep learning. Fig. 7 shows the taxonomy of the machine learning approaches in software vulnerability detection. Existing works are categorised into different categories of machine learning approaches based on their methodology of detection.

4.1. Supervised learning

Supervised learning is the most common type of machine learning approaches in software vulnerability detection. This is due to its ability to perform really well with the help of reliable datasets. Supervised learning concerns with the usage of datasets which contain labels such as vulnerable and non-vulnerable. Each observation in the dataset is already labelled prior to the training of the vulnerability detection model. Some of these labelled datasets were made available in public by past researchers who worked on vulnerability detection research. Some researchers created their private labelled dataset to assist them in their research. Fig. 8 illustrates the process of supervised learning in software vulnerability detection.

The process is described below:

i. Collects the labelled datasets from various sources, such as Github, SARD and NVD.
ii. Pre-processes and cleans the dataset from unwanted noise that could interrupt the model learning process.
iii. Generates features and performs features selection using a technique such as InfoGain and ExtraTree
iv. Splits the dataset into two different sets: a) training set and b) testing set.
v. Trains a predictive model using the training set with a supervised learning algorithm such as Decision Tree.
vi. Evaluates the performance of the predictive model using the testing set.
vii. The predictive model outputs its prediction based on the classes of the labelled dataset, such as vulnerable and non-vulnerable.

This explains the generic process of a supervised learning approach in software vulnerability detection. There are many existing works that implement supervised learning in their methodology. Table 15 shows the list of past works in software vulnerability detection that implement a supervised learning approach.

The supervised learning approach is famously known for its great performance in executing a given task. In this case, it is also the same in software vulnerability detection. In Kronjee et al. (2018) and Yu et al. (2018), they achieved detection performance up to 95%, which is very high as compared to other works that use a conventional method such as in Li et al. (2017b). In addition, supervised learning is also applicable for multi-class detection rather than binary detection, as shown by Gawron et al. (2017) and Grieco and Dinaburg (2018). They performed multi-class detection by detecting more than two classes of vulnerability

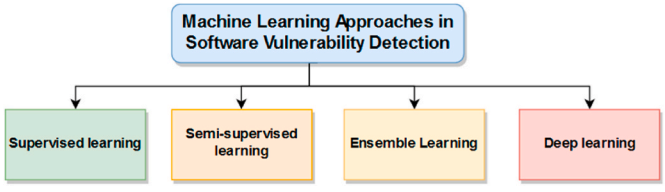

Fig. 7. Taxonomy of machine learning approaches in software vulnerability detection.

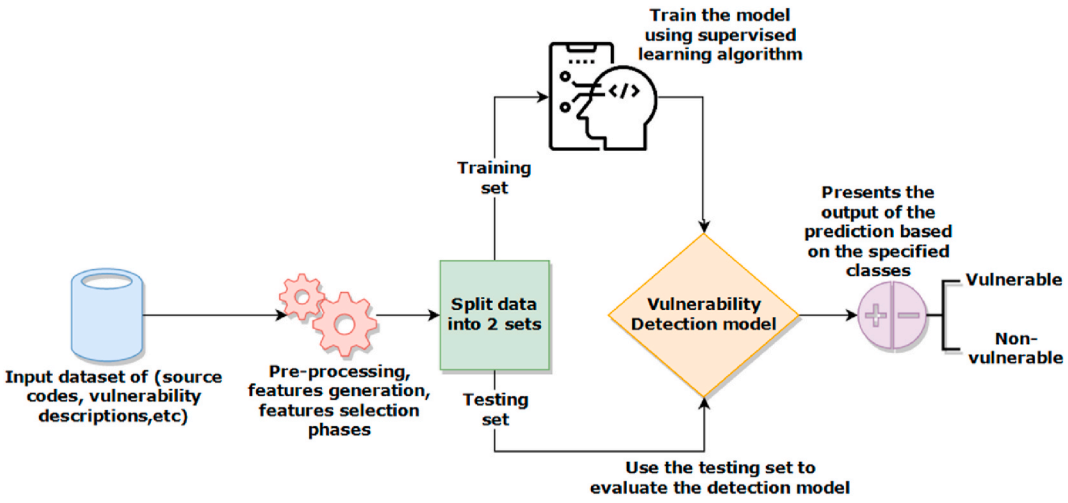

Fig. 8. The generic process of supervised learning in software vulnerability detection.

as compared to other works that only detect whether the observation is vulnerable or non-vulnerable. This highlights the reliability and ability of a supervised learning approach as a more granular task able to be executed. Even though supervised learning is widely used in software vulnerability detection, there are also several existing works that implement another type of machine learning approach called semi-supervised learning.

4.2. Semi-supervised learning

Semi-supervised learning is an extension of supervised learning where the training consists of both labelled and unlabelled observations or instances (Xiaojin and Andrew, 2009). The idea of semi-supervised learning is to take advantage of the classifier trained from the labelled data and perform prediction on the unlabelled data. Consequently, the predicted data are then used to retrain the model to achieve a better model performance as compared to training only on the initially labelled dataset. Semi-supervised learning is desirable as it is able to utilise both labelled and unlabelled data. This is very reliable in a situation where it is hard to obtain a huge amount of labelled data as compared to unlabelled data. This approach is beneficial in various research fields, such as in sentiment analysis, medical and face recognition (Gan, 2018; Ito et al., 2019; Xu and Tan, 2019). Similarly, semi-supervised learning is also being applied to detect software vulnerabilities. Fig. 9 illustrates the generic process of semi-supervised learning in software vulnerability detection.

The process is described below:

i. Divides the dataset into two sets: a) labelled dataset b) unlabelled dataset.
ii. Trains the predictive model using the labelled dataset (incomplete model).
iii. Predicts the label of the unlabelled dataset using the incomplete model.
iv. Maps the predicted labels back into their unlabelled observations.
v. This completely changes the unlabelled dataset to become a pseudo-labelled dataset.
vi. Combines the pseudo-labelled dataset and the labelled dataset into one complete dataset.
vii. Splits the complete dataset into training and testing sets.
viii. Trains the detection model using the newly-created training set.
ix. Evaluates the model using the newly-created testing set.
x. Outputs the prediction of the predictive model based on the classes of the labelled dataset such as vulnerable and non-vulnerable.

This explains the generic process of a semi-supervised learning approach in software vulnerability detection. There are several past research works that implement semi-supervised learning in their methodology. Table 16 lists the recent works that use semi-supervised learning approach for software vulnerability detection.

Meng et al. (2016) implemented a semi-supervised learning approach in their work to detect Buffer overflow vulnerabilities. They use a Label Propagation graphs algorithm to train a vulnerability detection model and achieved great detection performance with an accuracy of 88.4% on Nginx and Libexif binaries. Similarly, L. K. Shar et al. (2015) used a particular variant of the Random Forests algorithm that is designed for semi-supervised learning called Co-Forest (Settouti et al., 2013). They managed to achieve a precision score of up to 81% for the detection of cross-site scripting, SQL injection and remote code execution vulnerabilities across several datasets. This shows that semi-supervised learning is gaining potential as it helps with the problem of having few labelled datasets. The applicability of this approach in vulnerability detection has raised some attention in another robust approach called ensemble learning.

4.3. Ensemble learning

Ensemble learning is another branch of machine learning approaches where it combines multiple machine learning classifiers into one classifier to obtain a composite classifier with lower variance and, thus, fewer errors (Polikar, 2012). In other words, the idea of using ensemble learning is to combine the predictive power of each classifier to achieve the same objective. Ensemble learning is divided into two categories: a) bagging and b) boosting. Bagging or Bootstrap Aggregation is a combination of multiple independent learners, while boosting utilises weighted averages to make multiple weak learners into stronger learners (Quinlan, 1996). However, this study only discusses on bagging approaches as there are few known works that implement boosting for software vulnerability detection. As such, Fig. 10 illustrates the generic process of ensemble learning in software vulnerability detection.

The process is described as below:

i. Cleans and pre-processes the dataset
ii. Generates the features needed for the predictive models.
iii. Splits the dataset into two sets: a) training set and b) testing set.
iv. Passes the training set to the ensemble phase where a multiple machine learning model is trained using different or the same algorithms.
v. This results in multiple numbers of models waiting to be used.

Table 15
Supervised learning approaches in software vulnerability detection.

Reference	Techniques	Type of detection	Research interest	Performance
Ren et al. (2019)	Classification and Regression Trees (CART)	Binary detection	Detection performance	Accuracy up to 82.55% with Gini index as features selection technique
Chernis & Verma (2018)	Naïve Bayes, K-nearest Neighbor, Neural Network, Support Vector Machine, Decision Tree, Random Forests	Binary detection	Detection performance	Accuracy up to 75% with InfoGain as feature selection technique
Kronjee et al. (2018)	Random Forests, Decision Tree, Logistic Regression, Naïve Bayes, Tree Augmented Naïve Bayes	Binary detection	Conventional techniques	Precision up to 95% across two datasets
Yu et al. (2018)	Support Vector Machines	Binary detection	Conventional techniques	Recall up to 95% when using text data
Grieco & Dinaburg (2018)	Support Vector Machines, K-nearest Neighbor, Random Forests	Multiclass detection	Conventional techniques	Accuracy up to 70% using sequences of system calls
Sultana (2017)	Support Vector Machines, Logistic Regression	Binary detection	Classic metrics and features	Precision up to 88.8% for detection
Sultana & Williams (2017)	J48 Decision Tree, Naïve Bayes	Binary detection	Classic metrics and features	Recall up to 92.3% for detection
Gawron et al. (2017)	Neural Network, Naïve Bayes	Multiclass detection	Manual analysis	Accuracy up to 90.8% based on attack vector
Catal et al. (2017)	Averaged perceptron, Boosted Decision Trees, Decision Forests, Locally deep support vector machine, Logistic regression, Support Vector Machines, Neural Network	Binary detection	Conventional techniques	Accuracy up to 76.5% for averaged across three datasets
Medeiros et al. (2016)	Hidden Markov Model	Binary detection	Manual analysis	Accuracy and precision scores up to 96%
Alves et al. (2016)	Logistic Regression, Decision Tree	Binary detection	Dataset	Accuracy up to 90% using release-fold validation
Grieco et al. (2016)	Logistic Regression, Multilayer Perceptron, Random Forests	Binary detection	Conventional techniques	Average test error of 31% using bag-of-words
Younis et al. (2016)	Logistic Regression, Naïve Bayes, Random Forest, Support Vector Machines	Binary detection	Miscellaneous	Accuracy up to 84% using wrapper subset
Li et al. (2016)	Support Vector Machines	Binary detection	Classic metrics and features	F-measure is 18% higher than the best existing code similarity algorithm
Morrison et al. (2015)	Logistic Regression, Naïve Bayes, Recursive Partitioning, Support Vector Machines, Tree Bagging, Random Forest	Binary detection	Detection performance	Precision up to 75% for binary-level prediction
Perl et al. (2015)	Linear Support Vector Machines	Binary detection	Detection performance	Precision up to 60% for detection
Y. Pang et al. (2015)	Support Vector Machines	Binary detection	Overfitting	Accuracy up to 92.25% for detection
Scandariato et al. (2014)	Naïve Bayes, Random Forest	Binary detection	Detection performance	Precision up to 99% for single Android application
Moshtari et al. (2013)	Naïve Bayes, Bayesian Network, Random Forests, J48 Decision Tree, Logistic Regression	Binary detection	Code complexity	Accuracy up to 92% for within-project evaluation
Lwin Khin Shar and Tan (2013)	Naïve Bayes, Multilayer Perceptron, C4.5 Decision Tree	Binary detection	Conventional techniques	Accuracy up to 93% for detection
L. K. Shar and Tan (2012)	Naïve Bayes, Multilayer Perceptron, C4.5 Decision Tree	Binary detection	Conventional techniques	Accuracy up to 84% for detection
Hovsepyan et al. (2012)	Support Vector Machines	Binary detection	Classic metrics and features	Accuracy up to 88% for detection
Shin et al. (2011)	Logistic Regression	Binary detection	Code complexity	Recall up to 81% using InfoGain
Shin et al. (2011)	Logistic Regression, J48 Decision Tree, Random Forest, Naïve Bayes	Binary detection	Code complexity	Accuracy up to 90% using multivariate detection
S. Zhang et al. (2011)	Linear Regression, Multilayer Perceptron, Support Vector Machines, Logistic Regression, RBF Network	Regression	Detection class	RMSE value of 11.46 for regression

vi. Uses the outputs of these models as an input for a final ensemble model.
vii. Trains the final predictive model using an ensemble algorithm such as Random Forests.
viii. Evaluates using the testing set to measure its performance in detecting the vulnerabilities.

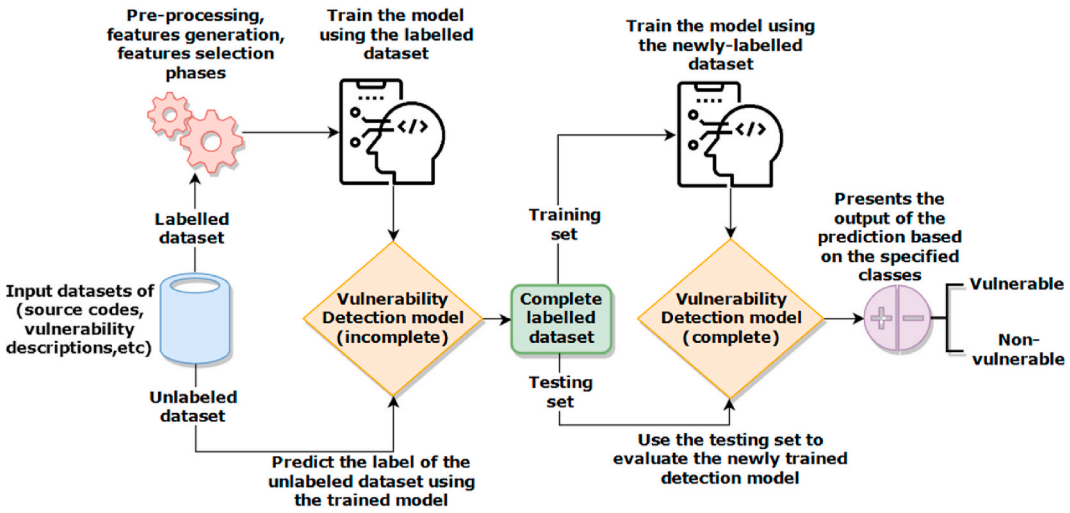

Fig. 9. The generic process of semi-supervised learning in software vulnerability detection.

Table 16
Semi-supervised learning approach in software vulnerability detection.

Reference	Techniques	Classes of Detection	Research interest	Performance
Meng et al. (2016)	Label Propagation Graphs	Binary detection	Dataset	Accuracy up to 88.4% for detection
L. K. Shar et al. (2015)	Co-trained Random Forest	Binary detection	Dataset	Precision up to 81% for detection of cross-site scripting vulnerability
L. K. Shar et al. (2013)	Logistic Regression, Multilayer perceptron, K-means clustering	Binary detection	Classic metrics and features	Recall up to 90% on labelled data and 76% across unlabelled data

ix. Outputs the prediction of the predictive models based on the classes of the labelled dataset, such as vulnerable and non-vulnerable.

This explains the generic process of an ensemble learning approach in software vulnerability detection. There are several past research works that implement ensemble learning in their framework. Table 17 lists all the recent works in software vulnerability detection that implements an ensemble learning approach.

Khalid et al. (2019) implemented a semi-supervised approach in detecting well-known software vulnerabilities like SQL injection, XSS and cross-site request forgery. They built six different classifiers using J48, Naïve Bayes, Random Forests algorithms and combined all of them using a meta-classifier (Random Forests). As expected, they managed to achieve precision scores up to 84% with the help of a synthetic minority oversampling technique (SMOTE) algorithm to handle the class balance problem. Similarly, Jimenez et al. (2016) and Stuckman et al. (2017) also implemented an ensemble technique using Random Forests and achieved recall and precision score up to 88%. This shows that the

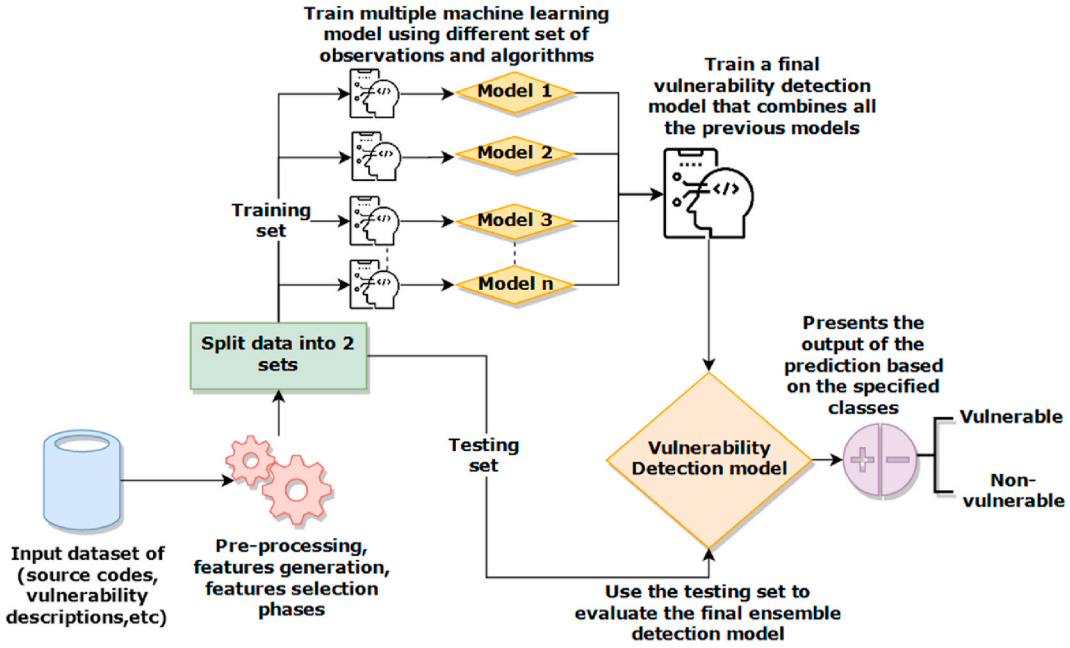

Fig. 10. The generic process of ensemble learning in software vulnerability detection.

Table 17
Ensemble learning approach in software vulnerability detection.

Reference	Techniques	Classes of Detection	Research interest	Performance
Khalid et al. (2019)	Meta classifier (Random Forests)	Binary detection	Detection class	Precision up to 84% with SMOTE filter applied
Stuckman et al. (2017)	Random Forests	Binary detection	Overfitting	Recall up to 88% for cross-project detection
Jimenez et al. (2016)	Random Forests	Binary detection	Dataset	Precision up to 76% using text mining
Y. Zhang et al. (2015)	Composer (Random Forests)	Binary detection	Detection performance	F1 scores up to 75% for detection
Walden et al. (2014)	Random Forests	Binary detection	Dataset	Recall up to 80.5% using text mining

ensemble learning approach is reliable in detecting software vulnerabilities by combining multiple different prediction models. The robustness of ensemble learning gives rise to another powerful type of machine learning approach called deep learning.

4.4. Deep learning

Deep learning is an extension of a very powerful and complex machine learning technique called Artificial Neural Network (ANN). It consists of multiple hidden layers of neurons connected which contains weights and performs real-value calculations based on its input (Schmidhuber, 2015). In other words, Deep Neural Network (DNN) is a feed-forward artificial neural network that contains more than one fully-connected hidden layers along with input and output layers (Hinton et al., 2012). Besides DNN, there are many other deep learning techniques such as Convolutional Neural Network, Long Short-Term Memory (LSTM) and Recurrent Neural Network (RNN) (LeCun et al., 2015). Each deep learning technique has its architectures, implementations and structures. Fig. 11 illustrates the generic process of a deep learning approach in software vulnerability detection.

The process is described as below:

i. Cleans and pre-processes the dataset
ii. Generates the features needed for the predictive models.
iii. Splits into training and testing sets.
iv. Feeds the training set into the feed-forward neural network.
v. As the inputs pass through each hidden layer, weights and bias are calculated and propagated throughout the network.
vi. The network propagates through the activated neurons to the last fully-connected layer, which is the output layer
vii. The network calculates the class probability using an activation function and assigns the class probability to the output neurons.
viii. The number of output neurons depends on the number of classes. If the problem is a binary detection problem, the output neurons will be only two.
ix. Feeds the testing set into the trained DNN to evaluate the performance of the neural network.
x. The DNN outputs its prediction based on the specified classes.

This explains the generic process of a DNN architecture in software vulnerability detection. Keep in mind that different deep learning techniques exhibit different architectures and processes. Therefore, their ability to perform vulnerability detection also differs. There are several past research works that implement deep learning in their framework. Table 18 shows the list of existing works that use deep learning approach in software vulnerability detection.

Dam et al. (2018) and Li et al. (2018b) implemented bi-directional LSTM and LSTM to detect software vulnerabilities across multiple datasets. They achieved precision scores up to 94.6% by using the proposed approaches. On top of that, Li et al. (2018b) also managed to detect four vulnerabilities that are not reported in the NVD and silently patched by the developers. This shows that, with proper training and reliable datasets, deep learning approaches able to uncover new unidentified vulnerabilities. This also very useful in the case of predicting zero-day vulnerabilities during post-software development phase. Additionally, Russell et al. (2018) implemented Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) to detect vulnerabilities using the SATE IV Juliet Suite dataset. They used the Area Under the Receiver Operating Characteristics (ROC) Curve as the metric to measure the performance of their detection model. As expected, they managed to achieve 95.4% of AUC score, which means that the model is very accurate and has low number of false positives. This shows that deep learning approaches is slowly dominating the research in software vulnerability detection as more and more high-performance detection models are produced over time.

In this section, this study presents and discusses the taxonomy of machine learning approaches in software vulnerability detection. The taxonomy of machine learning approaches is divided into supervised learning, semi-supervised learning, ensemble learning and deep learning. Each category represents different types of machine learning approaches in the area of software vulnerability detection. As such, this study also highlights the uprising machine learning approach, known as

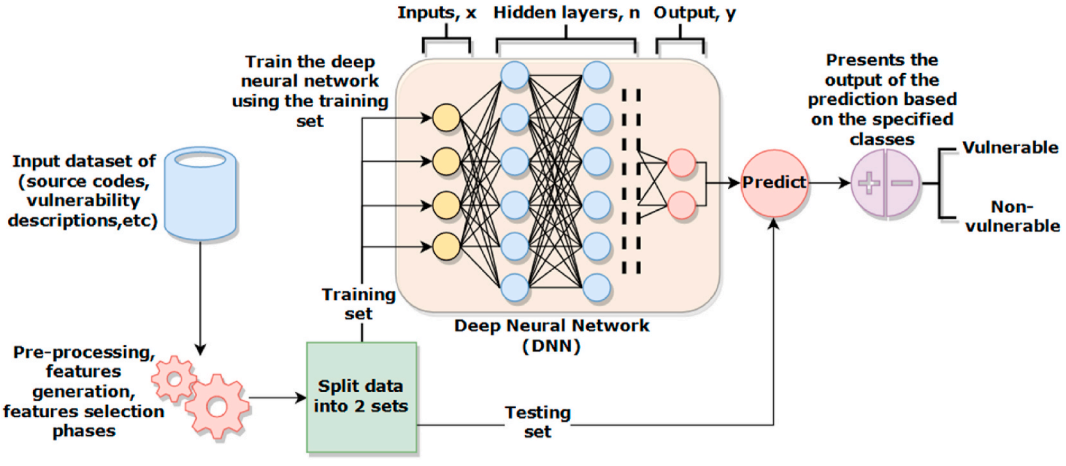

Fig. 11. The generic process of deep learning approach in software vulnerability detection.

Table 18
Deep learning approach in software vulnerability detection.

Reference	Techniques	Classes of Detection	Research interest	Performance
Li et al. (2020b)	Bi-directional Gated Recurrent Unit	Binary detection	Detection performance	F1 score up to 97% on test data
Fidalgo et al. (2020)	Long Short-term Memory	Binary detection	Conventional techniques	Accuracy score up to 95%
Guo et al. (2020)	Bi-directional Long Short-term Memory	Binary detection	Manual analysis	F1 score up to 91.44 for both vulnerabilities
Li et al. (2020a)	Concatenated Convolutional Neural Network	Binary detection	Manual analysis	F1 score up to 89.5% on test data
Niu et al. (2020)	CNN-Bi-directional Long Short-term Memory	Binary detection	Manual analysis	Accuracy up to 97.32% on test data
Tian et al. (2020)	Bi-directional Gated Recurrent Unit	Binary detection	Detection performance	F1 scores up to 89.9% with precision of up to 91%
Lin et al. (2020b)	Convolutional Neural Network	Binary detection	Dataset	Recall scores up to 97%
Zagane et al. (2020)	Deep Neural Network	Binary detection	Manual analysis	F1 scores up to 76.57% on test data
Duan et al. (2019)	Deep neural network with attention mechanism	Binary detection	Detection performance	F1 scores up to 80.6% on test data
Ban et al. (2019)	Bi-directional Long Short-term Memory	Binary detection	Manual analysis	Precision scores up to 90% on test data
Fang et al. (2019)	Long Short-term Memory	Binary and multiclass detection	Conventional techniques	F1 score up to 98.8%
Zhou et al. (2019a)	Gated Graph Neural Network	Binary detection	Manual analysis	F1 score up to 84.97% on test data
Saccente et al. (2019)	Long Short-term Memory	Binary detection	Detection performance	Accuracy up to 92.1 average across different vulnerabilities
Liu et al. (2019)	Bi-directional Long Short-Term Memory	Binary detection	Manual analysis	Precision up to 85% on C, C++ and Python functions.
Zou et al. (2019)	Bi-directional Long Short-Term Memory	Multiclass detection	Detection class	Weighted F1 score of 94.69% across all vulnerabilities
Liang et al. (2019)	Tree-based Convolutional Neural Network	Binary detection	Classic metrics and features	Accuracy up to 96.73% on test set
Huang et al. (2019)	Deep Neural Network	Binary detection	Overfitting	F1 score up to 81% on test data
Lin et al. (2018)	Bi-directional Long Short-Term Memory	Binary detection	Classic metrics and features	Precision up to 100% on LibTiff and FFmpeg.
Li et al. (2018b)	Bi-directional Long Short-Term Memory	Binary detection	Manual analysis	Precision up to 94.6% using RM dataset
Dam et al. (2018)	Long Short-Term Memory	Binary detection	Classic metrics and features	Precision up to 90% for joint features
Li et al. (2018a)	Convolutional Neural Network, Deep Belief Network, Recurrent Neural Network, Long Short-Term Memory, Gated Recurrent Unit, Bi-directional LSTM, Bi-directional GRU	Binary detection	Manual analysis	Accuracy up to 96% using data dependency and control dependency
Russell et al. (2018)	Convolutional Neural Network, Recurrent Neural Network	Binary detection	Conventional techniques	ROC Area Under the Curve up to 95.4% on SATE IV Juliet Suite dataset
Pechenkin & Demidov (2018)	Bi-directional Long Short-Term Memory	Binary detection	Detection performance	Accuracy up to 92% on private dataset
Wu et al. (2017)	Convolutional Neural Network, Long Short-Term Memory, Convolutional Neural Network – Long Short-Term Memory	Binary detection	Manual analysis	Accuracy up to 86% for detection
Phan et al. (2017)	Directed Graph Convolutional Neural Network	Multiclass detection	Detection performance	Accuracy gain up to 11% as compared to tree-based model
Yulei Pang et al. (2017)	Deep Neural Network	Binary detection	Manual analysis	Accuracy up to 92.87% on Android applications
Han et al. (2017)	Convolutional Neural Network	Multiclass detection	Manual analysis	Accuracy up to 81.6% using vulnerability description

deep learning. The approach is gaining its popularity in the domain of software vulnerability detection as the approach is proven to be reliable and effective in other domains (Rhode et al., 2018; Zhu et al., 2018). The proposed taxonomy creates a new standpoint in approaches to software vulnerability detection as opposed to conventional methods.

5. Open issues and future trends

Software vulnerability detection is currently in need of a better discussion regarding their issues and future trends in the research. Existing works address various research problems in detection software vulnerabilities including methods, detection, features, codes and dataset. However, there are several issues that need far more attention and discussion. In this section, this study discusses the open issues and future trends in software vulnerability detection. Although there are many issues that must be discussed, this study selects only the most common issues which are gaining traction and attention in software vulnerability detection. The issues are dataset, multi-vulnerabilities detection, transfer learning and real-world application.

5.1. Datasets

Datasets are very important in any research work, especially in software vulnerability detection. There are several issues concerning the current state of the datasets in this area. These are some of the open issues regarding datasets in software vulnerability detection:

a) Labelled dataset

Meng et al. (2016) highlight the issue regarding labelled datasets. Obtaining data such as software source codes and vulnerability descriptions is manageable. However, the problem arises as these huge amounts of data are unlabelled. This is a critical issue because, currently, most of the approaches proposed by existing researchers require a tremendous amount of labelled data. This is also the case with supervised learning and deep learning approaches. The low amount of labelled data leads to lower performance and reliability of the detection model. Therefore, to overcome this issue, a huge set of labelled data must be proposed collectively by researchers that are working in the area of software vulnerability detection. A community-maintained database is also one of the options as research communities are able to contribute to solving the labelled dataset issue.

b) Gold-standard dataset

The next issue is regarding the standardised datasets used to perform evaluations on multiple vulnerability detection approaches. This issue is highlighted by Walden et al. (2014) and Jimenez et al. (2016) due to the inconsistencies of datasets used in the evaluation of vulnerability detection models. It is seen that different researchers adopt different datasets and some of them also use their private dataset, which limits and questions the reliability of their approach to detect vulnerabilities effectively (Seokmo Kim et al., 2016; Li et al., 2017a). To overcome this issue, standardised datasets for software vulnerability detection are necessary so that future researchers are able to evaluate and compare the performance of their proposed approach with existing approaches reliably. The standardised datasets must be available publicly so that security researchers can contribute to the growth of the datasets. At the same, they are also able to maintain the integrity and reliability of the datasets through community-based verification approach.

c) Synthetic dataset

Synthetic datasets are datasets that consist of non-real-world vulnerability samples which are unable to reflect the true structure of real-world vulnerabilities. These datasets are widely used in many software vulnerability detection works, such as in Li et al. (2018b) and Fang et al. (2019). The most widely used synthetic dataset is the Software Assurance Reference Dataset (SARD) (Technology, 2018). The dataset contains synthetic programs and test cases which are less useful because of their limited functionalities compared to real-world programs (Li et al., 2020b). In Liu et al. (2019), they also generated synthetic data to overcome the problem of an imbalanced dataset. However, they mentioned that precautionary steps must be taken as the synthetic vulnerable data sample may be less representative of real-world data.

In the future, a gold standard dataset is needed to provide a consistent comparative evaluation between works in software vulnerability detection. The dataset must have an adequate number of labelled samples, both vulnerable and non-vulnerable. In order to promote multiclass vulnerability detection, the dataset must include different types of vulnerability. This inclusion will increase the versatility and flexibility of the dataset while maintaining the quality samples. It is also imperative to avoid the use of synthetic samples and test cases when creating this dataset as it decreases the robustness and effectiveness of vulnerability detection system when tested with real-world source codes.

5.2. Multi-vulnerabilities detection

Multi-vulnerabilities detection is an open issue that most researchers are facing as there are numerous numbers of software security vulnerabilities available and reported (Mouzarani and Sadeghiyan, 2016). This list of vulnerabilities is also increasing over time. Therefore, most researchers focus on binary detection, which only determines whether the observations or data points are vulnerable or non-vulnerable (Khalid et al., 2019; Ren et al., 2019; Russell et al., 2018). There needs a more robust vulnerability detection model that is able to perform granular detection of software vulnerabilities. Therefore, to overcome this issue, a more robust model is needed as it is able to help security researchers and developers when fixing the vulnerabilities that are detected by the detection model. For example, a model which is able to detect and classify multiple types of vulnerabilities is considered to be more robust and has a broader usage. This means that the model is able to perform multi-vulnerabilities detection and distinguish between buffer overflow, SQL injection and cross-site scripting vulnerabilities at the same.

In the future, we need more works that focus on multi-vulnerabilities detection and support multiple different types of vulnerabilities other than buffer overflow, SQL injection and cross-site scripting. Currently, some of the existing works combine different types of vulnerability to perform a binary prediction of each sample. However, this confuses the detection model as it will learn patterns of varying vulnerability and be unable to distinguish between vulnerable and non-vulnerable samples effectively. Therefore, future works must address this issue, and, indirectly, this will affect the software engineers as they will be informed of which specific type of vulnerability the detection system is reporting. This allows them to patch the vulnerability effectively and efficiently without hassle.

5.3. Transfer learning

Transfer learning is another discipline of machine learning approach that is applicable in software vulnerability detection. This approach initially learns to perform a specific task. Then, the model is reused to perform another machine learning task through the process of knowledge transfer (Pan and Yang, 2010). In real-world applications, it is costly and time-consuming to retrain a machine learning model. Therefore, transfer learning helps in reducing the time consumption of training a machine learning model for different tasks or problems. This is hugely beneficial in software vulnerability detection as security researchers are able to reuse vulnerability detection models across various software projects. A model that is trained on a project is then ready to be used on different projects and continue learning incrementally over time. This approach indirectly addresses various problems, such as manual analysis (Liu et al., 2018), conventional techniques (Kronjee et al., 2018) and overfitting (Stuckman et al., 2017). Therefore, a transfer learning approach is applicable for cross-project vulnerability detection.

Future works in software vulnerability detection must consider adopting transfer learning, provided if enough data are available to perform the task. Transfer learning works only if it is trained on huge datasets that are high in quality and reliable. However, this is not only limited to C/C++ programming language, but also for other programming languages such as PHP and Javascript. Since detection models are less interoperable across different programming languages, a separate detection model is needed for each language. In the long run, this increases the possibility of having a generalised vulnerability detection model that applies to multiple different software projects. The model is very robust and efficient in detecting vulnerabilities across different software.

5.4. Real-world application

Detecting software security vulnerabilities in real-world projects is very challenging as new projects contain no history of vulnerabilities or source codes. This situation creates a challenge for vulnerability detection models as they need to predict zero-day software vulnerabilities during the software development phase rather than in post-production settings. Li et al. (2018b) managed to discover four new software vulnerabilities upon performing the vulnerability detection process on software. This shows that their approach is very reliable as it managed to detect unknown hidden vulnerabilities inside the software. Therefore, there is in need of a robust and dependable approach that is functioning

in real-world environments due to the challenges posed by real-world software projects.

In the future, detecting real-world vulnerabilities should be the main priority of software vulnerability detection works instead of achieving the best detection performance across synthetic datasets and test cases. This indirectly affects the robustness and reliable of the detection model as it is exposed to more real-world samples, which allows the model to learn the syntactic and semantic structure of real-world vulnerabilities. By achieving this, such model will be able to help software engineers and security analysts in discovering potential vulnerabilities in software during the software development phase.

In this section, this study discusses open issues and trends related to the research in software vulnerability detection. Several important issues and future directions are highlighted to increase the visibility, reliability and effectiveness of the works in software vulnerability detection.

6. Conclusion

In conclusion, software vulnerability detection holds an essential role in software security research. It is instrumental during the software development phase as developers are still developing the software. It also promotes in-development software vulnerability detection and efficiently allows developers to reduce the number of vulnerabilities patch after the production phase. In addition, software vulnerability detection also grants big corporations the ease of security as they have to concern less regarding the security state of their software and are able to focus on a more complex decision-making task. As such, there are several methodologies and approaches proposed by the research and industry community to boost the development of a more sustainable, reliable, robust and effective vulnerability prediction framework. This allows researchers to produce numerous review papers that discuss the domain of software vulnerability detection. However, existing review papers focus on conventional approaches and methodologies while putting less attention on the primary research problems and the newer machine learning approaches in software vulnerability detection.

This study presents a comprehensive review of software vulnerability detection from the perspective of research interests and machine learning approaches. The review process assesses numerous research papers in the area of software vulnerability detection from 2011 to 2020. Furthermore, this study proposes a taxonomy of research interests by categorising the existing works into six main categories, which are methods, detection, features, code, dataset and miscellaneous. Some of these categories have their subcategories, which further narrows the problems raised in the existing works. Moreover, this study also proposes a taxonomy of machine learning approaches in software vulnerability detection. The taxonomy divides existing works into four categories: a) supervised learning, b) semi-supervised learning, c) ensemble learning and d) deep learning. This study offers a paradigm shift in software vulnerability detection review by highlighting the trends of research problems and the usage of a popular machine learning approach called deep learning.

This study also discusses several open issues and trends that are gaining attention in software vulnerability detection. As such, a standardised and labelled dataset is required to assist future researchers in evaluating their proposed vulnerability detection approaches. On the other hand, multi-vulnerabilities detection is also important as a more robust detection model is needed to detect multiple types of vulnerabilities. This allows the developers to effectively and instantaneously patch the detected vulnerabilities as they already know which vulnerability it belongs to. Another trend that is gaining popularity in various domains is the transfer learning approach. It is crucial to note that transfer learning is very reliable as it allows vulnerability detection models to perform cross-project detection and increases the generalisation of the model across different projects. Therefore, security researchers are less required to train different models for various software projects. Finally, this study addresses the real-world application of software vulnerability detection solutions. As such, real-world application is very challenging due to the unavailability of past source codes and vulnerabilities histories. The situation is more or less like detecting zero-day vulnerabilities. Therefore, it is imperative that further developments of a robust and effective software vulnerability detection approach are needed to address the problem of the real-world application.

Compliance with ethical standards

This study acknowledges and complies with the ethical standards of writing a research article.

Availability of data and materials

This study performs a survey of past research works without performing any experiment. Therefore, there is no experimentation data involved in the process.

Authors' contributions

Mohamad Hazim: Conceptualisation, Data curation, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualisation, Writing – original draft, Writing – review & editing. Mohd Hairul Nizam Md Nasir: Conceptualisation, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualisation, Writing – original draft, Writing – review & editing. Mohd Faizal Ab Razak: Conceptualisation, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Validation, Writing – review & editing. Ahmad Firdaus: Conceptualisation, Formal analysis, Investigation, Resources, Software. Nor Badrul Anuar: Conceptualisation, Data curation, Formal analysis, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualisation, Writing – original draft, Writing – review & editing.

Ethical approval

This article does not contain any study involving human participants or animals performed by any of the authors.

Declaration of competing interest

The author(s) declare(s) that there is no conflict of interest.

Acknowledgement

The work of the authors was supported by the University of Malaya Research Grant Programme (UMRG) under grant RF008D-2018 and partly by the Ministry of Higher Education, Malaysia under grant RACER/1/2019/ICT03/UMP/2.

References

Afifi, F., Anuar, N.B., Shamshirband, S., Choo, K.-K.R., 2016. DyHAP: dynamic hybrid ANFIS-PSO approach for predicting mobile malware. PloS One 11 (9), e0162627. https://doi.org/10.1371/journal.pone.0162627.

Alves, H., Fonseca, B., Antunes, N., 2016. Experimenting machine learning techniques to predict vulnerabilities. In: Paper Presented at the 2016 Seventh Latin-American Symposium on Dependable Computing (LADC).

Ban, X., Liu, S., Chen, C., Chua, C., 2019. A performance evaluation of deep-learnt features for software vulnerability detection. Concurrency Comput. Pract. Ex. 31 (19), e5103 https://doi.org/10.1002/cpe.5103.

Bissell, K., Cin, R.M.L.P.D., 2019. Ninth Annual Cost of Cybercrime Study. Retrieved from. https://www.accenture.com/us-en/insights/security/cost-cybercrime-study.

Bosu, A., Carver, J.C., Hafiz, M., Hilley, P., Janni, D., 2014. Identifying the characteristics of vulnerable code changes: an empirical study. In: Paper Presented at the

Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China.
Catal, C., Akbulut, A., Ekenoglu, E., Alemdaroglu, M., 2017. Development of a software vulnerability prediction web service based on artificial neural networks. In: Paper Presented at the 2017 Pacific-Asia Conference on Knowledge Discovery and Data Mining. PAKDD, Cham.
Chernis, B., Verma, R., 2018. Machine learning methods for software vulnerability detection. In: Paper Presented at the Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics. Tempe, AZ, USA.
Cowan, C., Pu, C., Maier, D., Hintony, H., Walpole, J., Bakke, P., Zhang, Q., 1998. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In: Paper Presented at the Proceedings of the 7th Conference on USENIX Security Symposium, vol. 7. San Antonio, Texas.
Dam, H.K., Tran, T., Pham, T.T.M., Ng, S.W., Grundy, J., Ghose, A., 2018. Automatic feature learning for predicting vulnerable software components. IEEE Trans. Software Eng. 1. https://doi.org/10.1109/TSE.2018.2881961, 1.
Domingos, P., 2012. A few useful things to know about machine learning. Commun. ACM 55 (10), 78–87. https://doi.org/10.1145/2347736.2347755.
Duan, X., Wu, J., Ji, S., Rui, Z., Luo, T., Yang, M., Wu, Y., 2019. VulSniper: focus your attention to shoot fine-grained vulnerabilities. In: Paper Presented at the IJCAI.
Fang, Y., Han, S., Huang, C., Wu, R., 2019. TAP: a static analysis model for PHP vulnerabilities based on token and deep learning technology. PloS One 14 (11), e0225196. https://doi.org/10.1371/journal.pone.0225196.
Feizollah, A., Ainin, S., Anuar, N.B., Abdullah, N.A.B., Hazim, M., 2019. Halal products on twitter: data extraction and sentiment analysis using stack of deep learning algorithms. IEEE Access 7, 83354–83362. https://doi.org/10.1109/ACCESS.2019.2923275.
Fidalgo, A., Medeiros, I., Antunes, P., Neves, N., 2020. Towards a deep learning model for vulnerability detection on web application variants. In: Paper Presented at the 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).
Firdaus, A., Anuar, N.B., Razak, M.F.A., Sangaiah, A.K., 2017. Bio-inspired computational paradigm for feature investigation and malware detection: interactive analytics. Multimed. Tool. Appl. https://doi.org/10.1007/s11042-017-4586-0.
Firdaus, A., Razak, M.F.A., Feizollah, A., Hashem, I.A.T., Hazim, M., Anuar, N.B., 2019. The rise of "blockchain": bibliometric analysis of blockchain study. Scientometrics. https://doi.org/10.1007/s11192-019-03170-4.
Gan, H., 2018. A noise-robust semi-supervised dimensionality reduction method for face recognition. Optik 157, 858–865. https://doi.org/10.1016/j.ijleo.2017.11.140.
Gawron, M., Cheng, F., Meinel, C., 2017. Automatic vulnerability classification using machine learning. In: Paper Presented at the 2017 12th International Conference on Risks and Security of Internet and Systems, Cham.
Ghaffarian, S.M., Shahriari, H.R., 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: a survey. ACM Comput. Surv. 50 (4), 1–36. https://doi.org/10.1145/3092566.
Ghosh, A.K., Connor, T.O., McGraw, G., 1998. An automated approach for identifying potential vulnerabilities in software. In: Paper Presented at the Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186).
Goseva-Popstojanova, K., Perhinschi, A., 2015. On the capability of static code analysis to detect security vulnerabilities. Inf. Software Technol. 68, 18–33. https://doi.org/10.1016/j.infsof.2015.08.002.
Grieco, G., Dinaburg, A., 2018. Toward smarter vulnerability discovery using machine learning. In: Paper Presented at the Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security, Toronto, Canada.
Grieco, G., Grinblat, G.L., Uzal, L., Rawat, S., Feist, J., Mounier, L., 2016. Toward large-scale vulnerability discovery using machine learning. In: Paper Presented at the Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, Louisiana, USA.
Guo, N., Li, X., Yin, H., Gao, Y., 2020. VulHunter: an Automated Vulnerability Detection System Based on Deep Learning and Bytecode, Cham.
17-22 Sept. 2017 Han, Z., Li, X., Xing, Z., Liu, H., Feng, Z., 2017. Learning to predict severity of software vulnerability using only vulnerability description. In: Paper Presented at the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME).
Hanif, M.H.M., Adewole, K.S., Anuar, N.B., Kamsin, A., 2018. Performance evaluation of machine learning algorithms for spam profile detection on twitter using WEKA and RapidMiner. Adv. Sci. Lett. 24 (2), 1043–1046. https://doi.org/10.1166/asl.2018.10683.
Hawkins, D.M., 2004. The problem of overfitting. J. Chem. Inf. Comput. Sci. 44 (1), 1–12. https://doi.org/10.1021/ci0342472.
Hazim, M., Anuar, N.B., Ab Razak, M.F., Abdullah, N.A., 2018. Detecting opinion spams through supervised boosting approach. PloS One 13 (6), e0198884. https://doi.org/10.1371/journal.pone.0198884.
Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A., Jaitly, N., Kingsbury, B., 2012. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. 29 (6), 82–97. https://doi.org/10.1109/MSP.2012.2205597.
Hovsepyan, A., Scandariato, R., Joosen, W., Walden, J., 2012. Software vulnerability prediction using text analysis techniques. In: Paper Presented at the Proceedings of the 4th International Workshop on Security Measurements and Metrics, Lund, Sweden.
Huang, G., Li, Y., Wang, Q., Ren, J., Cheng, Y., Zhao, X., 2019. Automatic classification method for software vulnerability based on deep neural network. IEEE Access 7, 28291–28298. https://doi.org/10.1109/ACCESS.2019.2900462.
Ito, R., Nakae, K., Hata, J., Okano, H., Ishii, S., 2019. Semi-supervised deep learning of brain tissue segmentation. Neural Network. 116, 25–34. https://doi.org/10.1016/j.neunet.2019.03.014.
Jimenez, M., Papadakis, M., Traon, Y.L., 2016. Vulnerability prediction models: a case study on the linux kernel. In: Paper Presented at the 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM).
Jingling, Z., Rulin, G., 2015. A new framework of security vulnerabilities detection in PHP web application. In: Paper Presented at the 2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing.
Jurn, J., Kim, T., Kim, H., 2018. An automated vulnerability detection and remediation method for software security. Sustainability 10 (5), 1652.
Khalid, M.N., Farooq, H., Iqbal, M., Alam, M.T., Rasheed, K., 2019. Predicting web vulnerabilities in web applications based on machine learning. In: Paper Presented at the 2018 International Conference on Intelligent Technologies and Applications, Singapore.
Kim, S., Kim, R.Y.C., Park, Y.B., 2016. Software vulnerability detection methodology combined with static and dynamic analysis. Wireless Pers. Commun. 89 (3), 777–793. https://doi.org/10.1007/s11277-015-3152-1.
Kim, S., Woo, S., Lee, H., Oh, H., 2017. VUDDY: a scalable approach for vulnerable code clone discovery. In: Paper Presented at the 2017 IEEE Symposium on Security and Privacy (SP).
Kronjee, J., Hommersom, A., Vranken, H., 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In: Paper Presented at the Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany.
Kulenovic, M., Donko, D., 2014. A survey of static code analysis methods for security vulnerabilities detection. In: Paper Presented at the 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).
Last, D., 2015. Using historical software vulnerability data to forecast future vulnerabilities. In: Paper Presented at the 2015 Resilience Week (RWS).
LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436. https://doi.org/10.1038/nature14539.
Lekies, S., Stock, B., Johns, M., 2013. 25 million flows later: large-scale detection of DOM-based XSS. In: Paper Presented at the Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany.
Li, J., Chen, J., Huang, M., Zhou, M., Zhang, L., Xie, W., 2017a. An integration testing platform for software vulnerability detection method. In: Paper Presented at the 2017 IEEE Trustcom/BigDataSE/ICESS.
Li, R., Feng, C., Zhang, X., Tang, C., 2019a. A lightweight Assisted vulnerability discovery method using deep neural networks. IEEE Access 7, 80079–80092. https://doi.org/10.1109/ACCESS.2019.2923227.
Li, X., Chen, J., Lin, Z., Zhang, L., Wang, Z., Zhou, M., Xie, W., 2017b. A mining approach to obtain the software vulnerability characteristics. In: Paper Presented at the 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD).
Li, X., Wang, L., Xin, Y., Yang, Y., Chen, Y., 2020a. Automated vulnerability detection in source code using minimum intermediate representation learning. Appl. Sci. 10 (5), 1692.
Li, Z., Zou, D., Tang, J., Zhang, Z., Sun, M., Jin, H., 2019b. A comparative study of deep learning-based vulnerability detection system. IEEE Access 7, 103184–103197. https://doi.org/10.1109/ACCESS.2019.2930578.
Li, Z., Zou, D., Xu, S., Chen, Z., Zhu, Y., Jin, H.-t. J.A., 2020b. VulDeeLocator: A Deep Learning-Based Fine-grained Vulnerability Detector abs/2001.02350.
Li, Z., Zou, D., Xu, S., Jin, H., Qi, H., Hu, J., 2016. VulPecker: an automated vulnerability detection system based on code similarity analysis. In: Paper Presented at the Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, California, USA.
Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., Chen, Z., 2018a. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities arXiv e-prints. https://ui.adsabs.harvard.edu/abs/2018arXiv180706756L.
Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Zhong, Y., 2018b. VulDeePecker: a deep learning-based system for vulnerability detection. In: Paper Presented at the Network and Distributed System Security Symposium, San Diego, California, USA.
14-19 July 2019 Liang, H., Yang, Y., Sun, L., Jiang, L., 2019. JSAC: a novel framework to detect malicious JavaScript via CNNs over AST and CFG. In: Paper Presented at the 2019 International Joint Conference on Neural Networks (IJCNN).
Lin, G., Wen, S., Han, Q.L., Zhang, J., Xiang, Y., 2020a. Software vulnerability detection using deep neural networks: a survey. Proc. IEEE 108 (10), 1825–1848. https://doi.org/10.1109/JPROC.2020.2993293.
Lin, G., Xiao, W., Zhang, J., Xiang, Y., 2020b. Deep Learning-Based Vulnerable Function Detection. A Benchmark, Cham.
Lin, G., Zhang, J., Luo, W., Pan, L., Xiang, Y., Vel, O.D., Montague, P., 2018. Cross-project transfer representation learning for vulnerable function discovery. IEEE Trans. Ind. Inf. 14 (7), 3289–3297. https://doi.org/10.1109/TII.2018.2821768.
Liu, D., Wang, J., Rong, Z., Mi, X., Gai, F., Tang, Y., Wang, B., 2018. Pangr: a behavior-based automatic vulnerability detection and exploitation framework. In: Paper Presented at the 2018 17th IEEE International Conference on Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference on Big Data Science And Engineering. TrustCom/BigDataSE.
Liu, S., Lin, G., Han, Q., Wen, S., Zhang, J., Xiang, Y., 2019. DeepBalance: deep-learning and fuzzy oversampling for vulnerability detection. IEEE Trans. Fuzzy Syst. 1, 1.
Ma, S., Thung, F., Lo, D., Sun, C., Deng, R.H., 2017. VuRLE: Automatic Vulnerability Detection and Repair by Learning from Examples. Cham.
Matteson, S., 2018. Software Failure Caused $1.7 Trillion in Financial Losses in 2017. Retrieved from. https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/.

Medeiros, I., Neves, N., Correia, M., 2016. DEKANT: a static analysis tool that learns to detect web application vulnerabilities. In: Paper Presented at the Proceedings of the 25th International Symposium on Software Testing and Analysis.
Meneely, A., Srinivasan, H., Musa, A., Tejeda, A.R., Mokary, M., Spates, B., 2013. When a patch goes bad: exploring the properties of vulnerability-contributing commits. In: Paper Presented at the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.
Meng, Q., Wen, S., Feng, C., Tang, C., 2016. Predicting buffer overflow using semi-supervised learning. In: Paper Presented at the 2016 9th International Congress on Image and Signal Processing. BioMedical Engineering and Informatics (CISP-BMEI).
Mohammed, N.M., Niazi, M., Alshayeb, M., Mahmood, S., 2017. Exploring software security approaches in software development lifecycle: a systematic mapping study. Comput. Stand. Interfac. 50, 107–115. https://doi.org/10.1016/j.csi.2016.10.001.
Morrison, P., Herzig, K., Murphy, B., Williams, L., 2015. Challenges with applying vulnerability prediction models. In: Paper Presented at the Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, Urbana, Illinois.
Moshtari, S., Sami, A., Azimi, M., 2013. Using complexity metrics to improve software security. Comput. Fraud Secur. 2013 (5), 8–17. https://doi.org/10.1016/S1361-3723(13)70045-9.
Mouzarani, M., Sadeghiyan, B., 2016. Towards designing an extendable vulnerability detection method for executable codes. Inf. Software Technol. 80, 231–244. https://doi.org/10.1016/j.infsof.2016.09.004.
Niu, W., Zhang, X., Du, X., Zhao, L., Cao, R., Guizani, M., 2020. A deep learning based static taint analysis approach for IoT software vulnerability location. Measurement 152, 107139. https://doi.org/10.1016/j.measurement.2019.107139.
Pan, S.J., Yang, Q., 2010. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191.
Pang, Y., Xue, X., Namin, A.S., 2015. Predicting vulnerable software components through N-gram analysis and statistical feature selection. In: Paper Presented at the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA).
Pang, Y., Xue, X., Wang, H., 2017. Predicting vulnerable software components through deep neural network. In: Paper Presented at the Proceedings of the 2017 International Conference on Deep Learning Technologies, Chengdu, China.
Pechenkin, A., Demidov, R., 2018. Applying deep learning and vector representation for software vulnerabilities detection. In: Paper Presented at the Proceedings of the 11th International Conference on Security of Information and Networks, Cardiff, United Kingdom.
Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Acar, Y., 2015. VCCFinder: finding potential vulnerabilities in open-source projects to assist code audits. In: Paper Presented at the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA.
Phan, A.V., Nguyen, M.L., Bui, L.T., 2017. Convolutional neural networks over control flow graphs for software defect prediction. In: Paper Presented at the 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI).
Polikar, R., 2012. Ensemble learning. In: Zhang, C., Ma, Y. (Eds.), Ensemble Machine Learning: Methods and Applications. Springer US, Boston, MA, pp. 1–34.
Quinlan, J.R., 1996. Bagging, boosting, and C4.S. In: Paper Presented at the Proceedings of the Thirteenth National Conference on Artificial Intelligence, vol. 1. Portland, Oregon.
Razak, M.F.A., Anuar, N.B., Othman, F., Firdaus, A., Afifi, F., Salleh, R., 2018. Bio-inspired for features optimization and malware detection. Arabian J. Sci. Eng. 43 (12), 6963–6979. https://doi.org/10.1007/s13369-017-2951-y.
Razak, M.F.A., Anuar, N.B., Salleh, R., Firdaus, A., 2016. The rise of "malware": bibliometric analysis of malware study. J. Netw. Comput. Appl. 75, 58–76. https://doi.org/10.1016/j.jnca.2016.08.022.
Ren, J., Zheng, Z., Liu, Q., Wei, Z., Yan, H., 2019. A buffer overflow prediction approach based on software metrics and machine learning. Secur. Commun. Network. 2019, 13. https://doi.org/10.1155/2019/8391425.
Rhode, M., Burnap, P., Jones, K., 2018. Early-stage malware prediction using recurrent neural networks. Comput. Secur. 77, 578–594. https://doi.org/10.1016/j.cose.2018.05.010.
Russell, R., Kim, L., Hamilton, L., Lazovich, T., Harer, J., Ozdemir, O., McConley, M., 2018. Automated vulnerability detection in source code using deep representation learning. In: Paper Presented at the 17th IEEE International Conference on Machine Learning and Applications. ICMLA, Orlando, Florida, USA.
Saccente, N., Dehlinger, J., Deng, L., Chakraborty, S., Xiong, Y., 2019. Project achilles: a prototype tool for static method-level vulnerability detection of java source code using a recurrent neural network. In: Paper Presented at the 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW).
Saleh, A.Z.M., Rozali, N.A., Buja, A.G., Jalil, K.A., Ali, F.H.M., Rahman, T.F.A., 2015. A method for web application vulnerabilities detection by using boyer-moore string matching algorithm. Procedia Comput. Sci. 72, 112–121. https://doi.org/10.1016/j.procs.2015.12.111.
Sarmah, U., Bhattacharyya, D.K., Kalita, J.K., 2018. A survey of detection methods for XSS attacks. J. Netw. Comput. Appl. 118, 113–143. https://doi.org/10.1016/j.jnca.2018.06.004.
Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W., 2014. Predicting vulnerable software components via text mining. IEEE Trans. Software Eng. 40 (10), 993–1006. https://doi.org/10.1109/TSE.2014.2340398.
Schmidhuber, J., 2015. Deep learning in neural networks: an overview. Neural Network. 61, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.
Settouti, N., Daho, M.E.H., Lazouni, M.E.A., Chikh, M.A., 2013. Random forest in semi-supervised learning (Co-Forest). In: Paper Presented at the 2013 8th International Workshop on Systems, Signal Processing and their Applications (WoSSPA).
Shahmehri, N., Mammar, A., Montes de Oca, E., Byers, D., Cavalli, A., Ardi, S., Jimenez, W., 2012. An advanced approach for modeling and detecting software vulnerabilities. Inf. Software Technol. 54 (9), 997–1013. https://doi.org/10.1016/j.infsof.2012.03.004.
Shahriar, H., Zulkernine, M., 2012. Mitigating program security vulnerabilities: approaches and challenges. ACM Comput. Surv. 44 (3), 1–46. https://doi.org/10.1145/2187671.2187673.
Shar, L.K., Briand, L.C., Tan, H.B.K., 2015. Web application vulnerability prediction using hybrid program analysis and machine learning. IEEE Trans. Dependable Secure Comput. 12 (6), 688–707. https://doi.org/10.1109/TDSC.2014.2373377.
Shar, L.K., Tan, H.B.K., 2012. Predicting common web application vulnerabilities from input validation and sanitization code patterns. In: Paper Presented at the 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering.
Shar, L.K., Tan, H.B.K., 2013. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. Inf. Software Technol. 55 (10), 1767–1780. https://doi.org/10.1016/j.infsof.2013.04.002.
Shar, L.K., Tan, H.B.K., Briand, L.C., 2013. Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In: Paper Presented at the 2013 35th International Conference on Software Engineering (ICSE).
Shin, Y., Meneely, A., Williams, L., Osborne, J.A., 2011. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans. Software Eng. 37 (6), 772–787. https://doi.org/10.1109/tse.2010.81.
Shin, Y., Williams, L., 2011. An initial study on the use of execution complexity metrics as indicators of software vulnerabilities. In: Paper Presented at the Proceedings of the 7th International Workshop on Software Engineering for Secure Systems. Waikiki, Honolulu, HI, USA.
Shin, Y., Williams, L., 2013. Can traditional fault prediction models be used for vulnerability prediction? Empir. Software Eng. 18 (1), 25–59. https://doi.org/10.1007/s10664-011-9190-8.
Shuai, B., Li, H., Zhang, L., Zhang, Q., Tang, C., 2015. Software vulnerability detection based on code coverage and test cost. In: Paper Presented at the 2015 11th International Conference on Computational Intelligence and Security (CIS).
Singh, U.K., Joshi, C., Kanellopoulos, D., 2019. A framework for zero-day vulnerabilities detection and prioritization. J. Inf. Secur. Appl. 46, 164–172. https://doi.org/10.1016/j.jisa.2019.03.011.
Stuckman, J., Walden, J., Scandariato, R., 2017. The effect of dimensionality reduction on software vulnerability prediction models. IEEE Trans. Reliab. 66 (1), 17–37. https://doi.org/10.1109/TR.2016.2630503.
Sultana, K.Z., 2017. Towards a software vulnerability prediction model using traceable code patterns and software metrics. In: Paper Presented at the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE).
Sultana, K.Z., Williams, B.J., 2017. Evaluating Micro Patterns and Software Metrics in Vulnerability Prediction. In: Paper Presented at the 2017 6th International Workshop on Software Mining (SoftwareMining).
Technology, N.I.o.S.a., 2018. Software Assurance Reference Dataset. Retrieved from. https://samate.nist.gov/SRD/.
Technology, N.I.o.S.a., 2020. National Vulnerability Database - Statistics. Retrieved from. https://nvd.nist.gov/vuln/search/statistics.
20-28 May 2017 Thomé, J., Shar, L.K., Bianculli, D., Briand, L., 2017. Search-driven string constraint solving for vulnerability detection. In: Paper Presented at the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE).
Tian, J., Xing, W., Li, Z., 2020. BVDetector: a program slice-based binary code vulnerability intelligent detection system. Inf. Software Technol. 123, 106289. https://doi.org/10.1016/j.infsof.2020.106289.
Vaidya, S., Ambad, P., Bhosle, S., 2018. Industry 4.0 – a glimpse. Procedia Manuf. 20, 233–238. https://doi.org/10.1016/j.promfg.2018.02.034.
Walden, J., Stuckman, J., Scandariato, R., 2014. Predicting vulnerable components: software metrics vs text mining. In: Paper Presented at the 2014 IEEE 25th International Symposium on Software Reliability Engineering..
Wang, X., Wu, R., Ma, J., Long, G., Han, J., 2018. Research on vulnerability detection technology for WEB mail system. Procedia Comput. Sci. 131, 124–130. https://doi.org/10.1016/j.procs.2018.04.194.
Wu, F., Wang, J., Liu, J., Wang, W., 2017. Vulnerability detection with deep learning. In: Paper Presented at the 2017 3rd IEEE International Conference on Computer and Communications (ICCC).
Xiaojin, Z., Andrew, G., 2009. Introduction to Semi-supervised Learning. Morgan & Claypool.
Xu, W., Tan, Y., 2019. Semi-supervised target-oriented sentiment classification. Neurocomputing 337, 120–128. https://doi.org/10.1016/j.neucom.2019.01.059.
Yamaguchi, F., Golde, N., Arp, D., Rieck, K., 2014. Modeling and discovering vulnerabilities with code property graphs. In: Paper Presented at the Proceedings of the 2014 IEEE Symposium on Security and Privacy.
Yamaguchi, F., Lottmann, M., Rieck, K., 2012. Generalized vulnerability extrapolation using abstract syntax trees. In: Paper Presented at the Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, Florida, USA.
Yamaguchi, F., Maier, A., Gascon, H., Rieck, K., 2015. Automatic inference of search patterns for taint-style vulnerabilities. In: Paper Presented at the Proceedings of the 2015 IEEE Symposium on Security and Privacy.
Yamaguchi, F., Wressnegger, C., Gascon, H., Rieck, K., 2013. Chucky: exposing missing checks in source code for vulnerability discovery. In: Paper Presented at the Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany.
Younis, A., Malaiya, Y., Anderson, C., Ray, I., 2016. To fear or not to fear that is the question: code characteristics of a vulnerable functionwith an existing exploit. In: Paper Presented at the Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, Louisiana, USA.

Yu, Z., Theisen, C., Williams, L., Menzies, T., 2018. Improving Vulnerability Inspection Efficiency Using Active Learning arXiv e-prints. https://ui.adsabs.harvard.edu/abs/2018arXiv180306545Y.

Zagane, M., Abdi, M.K., Alenezi, M., 2020. Deep learning for software vulnerabilities detection using code metrics. IEEE Access 8, 74562–74570. https://doi.org/10.1109/ACCESS.2020.2988557.

Zhang, S., Caragea, D., Ou, X., 2011. An empirical study on using the national vulnerability database to predict software vulnerabilities. In: Paper Presented at the Proceedings of the 22nd International Conference on Database and Expert Systems Applications, Toulouse, France.

Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., Li, S., 2015. Combining software metrics and text features for vulnerable file prediction. In: Paper Presented at the 2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS).

Zheng, W., Gao, J., Wu, X., Xun, Y., Liu, G., Chen, X., 2020. An empirical study of high-impact factors for machine learning-based vulnerability detection. In: Paper Presented at the 2020 IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF).

Zhou, M., Chen, J., Liu, Y., Ackah-Arthur, H., Chen, S., Zhang, Q., Zeng, Z., 2019a. A method for software vulnerability detection based on improved control flow graph. Wuhan Univ. J. Nat. Sci. 24 (2), 149–160. https://doi.org/10.1007/s11859-019-1380-z.

Zhou, Y., Liu, S., Siow, J., Du, X., Liu, Y., 2019b. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Paper Presented at the NeurIPS.

Zhu, X., Zhu, M., Ren, H., 2018. Method of plant leaf recognition based on improved deep convolutional neural network. Cognit. Syst. Res. 52, 223–233. https://doi.org/10.1016/j.cogsys.2018.06.008.

Zou, D., Wang, S., Xu, S., Li, Z., Jin, H., 2019. μVulDeePecker: a deep learning-based system for multiclass vulnerability detection. IEEE Trans. Dependable Secure Comput. 1. https://doi.org/10.1109/TDSC.2019.2942930, 1.

Hazim Hanif received his masters's degree in computer science by research from the University of Malaya, Malaysia in the topic of opinion spam detection. He is currently pursuing his PhD at Imperial College London, United Kingdom. His research interests include computer security (vulnerability detection), data sciences and machine learning.

Mohd Hairul Nizam Md Nasir is a Senior Lecturer in the Department of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya. His area of specialization includes software process, software process improvement, project management, software validation and verification, mobiles apps, soft-computing, e-learning and data analytics. He has published more than 100 scientific publications and won several awards at national and international exhibition

Mohd Faizal Ab Razak distinctively received his PhD from University of Malaya, Malaysia. He also obtained his Masters of Computer Science (Networking) at Universiti Malaysia Pahang, Malaysia. He is currently a senior lecturer at the Faculty of Computer Systems and Software Engineering at Universiti Malaysia Pahang, Malaysia. His area of research includes Mobile Security, Bibliometrics and Intrusion Detection System

Ahmad Firdaus distinctively received his PhD from University of Malaya, Malaysia. He also obtained his Masters of Computer Science (Networking) from University Teknologi Mara, Malaysia. He is currently a senior lecturer at the Faculty of Computer Systems and Software Engineering at Universiti Malaysia Pahang, Malaysia. His area of research includes Mobile Security, Blockchain and Intrusion Detection System

Nor Badrul Anuar received the master's degree in computer science from the University of Malaya, Kuala Lumpur, Malaysia, in 2003, and the Ph.D. degree in information security from the Centre for Security, Communications and Network Research, Plymouth University, U.K., in 2012. He is currently an Associate Professor with the Faculty of Computer Science and Information Technology, University of Malaya. He has published a number of conference and journal papers locally and internationally. His research interests include information security (intrusion detection systems), data sciences, artificial intelligence, and library information systems.