Informatics Bulletin, Faculty of Computers and Artificial Intelligence, Helwan University
Published Online ***Vol 4 Issue 1, January 2022***
(https://fcihib.journals.ekb.eg)

A Systematic Literature Review on Software Vulnerability Detection Using Machine Learning Approaches

Ahmed Bahaa[1,2] , Aya El-Rahman Kamal[1] , Amr S. Ghoneim[3]
[1] Department of Information Systems, Faculty of Computers and Artificial Intelligence, Helwan University, Helwan 11795, Egypt; ahmed.bahaa@fci.helwan.edu.eg or ahmed.bahaa.farid@gmail.com (A.B.); aya.rahman@fci.helwan.edu.eg or aya.elrahman93@gmail.com (A.K.)
[2] Department of Information Systems, Faculty of Computers and Artificial Intelligence, Beni-Suef University, Beni-Suef 62521, Egypt; ahmed.bahaa@fcis.bsu.edu.eg
[3] Department of Computer Science, Faculty of Computers and Artificial Intelligence, Helwan University, Helwan 11795, Egypt; amr.ghoneim@fci.helwan.edu.eg or amr.ghoneim@gmail.com (A.G.);

***Abstract*— Software vulnerabilities are security flaws, defects, or weaknesses in software architecture, design, or implementation. With the explosion of open source code available for analysis, there is a chance to learn about bug patterns that can lead to security vulnerabilities to assist in the discovery of vulnerabilities. Recent advances in deep learning in natural language processing, speech recognition, and image processing have demonstrated the great potential of neural models to understand natural language. This has encouraged researchers in the cybersecurity sector and software engineering to utilize deep learning to learn and understand vulnerable code patterns and semantics that indicate vulnerable code properties. In this paper, we review and analyze the recent state-of-the-art research adopting machine learning and deep learning techniques to detect software vulnerabilities, aiming to investigate how to leverage neural techniques for learning and understanding code semantics to facilitate vulnerability detection. From this paper's results, 12 primary studies were found from the search processes. 7 out of them were published in IEEE, 2 were published in ACM, 2 were published in Springer and the rest of them were published in different conferences and journals. Most primary studies worked on NVD and SARD datasets, and others used open-source projects. Results show that machine learning and deep learning techniques give promising results in the automatic detection of vulnerabilities, but there are still some gaps in existing models that need to be addressed in future research.**

***Index Terms*—Software vulnerabilities, vulnerability detection , machine learning, deep learning, program analysis.**

I. Introduction

With the rapid advancement of information technology, software is becoming increasingly vital in a wide range of fields around the world. Simultaneously, possible software security vulnerabilities are rising as a global challenge. One of the main reasons for security issues is software vulnerabilities. Recent history is full of examples, e.g., the infamous Heartbleed vulnerability was triggered by two missing lines of code [1]. Because of the widespread usage of open-source software and code reuse, these flaws are frequently caused by programming errors and can quickly propagate.

Vulnerabilities remain a big issue in spite of academic and industry attempts to improve software quality. This is demonstrated by the fact that numerous vulnerabilities are reported each year in the Common Vulnerabilities and Exposures (CVE) [2].

According to the Common Vulnerabilities and Exposures (CVE) organization's statistics and the National Vulnerability Database (NVD) Report [3], The number of vulnerabilities recorded in a single year has never been higher than in 2020 (18,103). It can be seen from Figure 1 that the number of vulnerabilities has reached its peak in the past three years.

Fig. 1. CVE number by year: 1988-2020

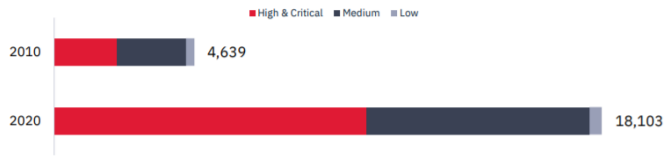

Fig. 2. CVEs reported by year: 2010 vs. 2020

Figure 2 depicts the rate of change, as demonstrated by the fact that in 2020 (10,342) there were more critical and high

severity vulnerabilities than in 2010. (4,639, including low, medium, high, and critical).

Resolving vulnerabilities discovered after software release is much more expensive than fixing them during development. Knowing that there are three types of code analysis approaches for detecting vulnerabilities: static, dynamic, and hybrid. Static techniques that depend on the source code without executing it, such as rule-based analysis, symbolic execution [4], and code similarity detection. Hence false-positives vulnerabilities may be reported. However, this method often struggles to reveal vulnerabilities that can occur during runtime. Dynamic methods focused on vulnerability detection that occurs during program execution. Dynamic analysis cannot analyze the behavior of a program as a whole because there is often an infinite amount of input and execution state, as taint analysis and fuzzing. They have low-code coverage since it's feasible to miss some vulnerabilities in invisible program states. Hybrid techniques combine static and dynamic analysis techniques to overcome the aforementioned shortcomings.

The structure of this article is as follows: Second section presents the used research methodology. The third section provides some background knowledge of different definitions of software vulnerability summarized from previous studies. Then, the related work of prior studies for vulnerability detection is discussed. Section four discusses machine learning-based vulnerability detection techniques. Section five discusses vulnerability detection techniques based on deep learning. Section six reviews and analyzes the state-of-the-art studies according to used techniques, datasets, and limitations. The results of the study are discussed in section seven. Finally, the last section concludes this paper and provides future work recommendations.

II. Research Methodology

To study software vulnerability detection models, the systematic literature review (SLR) methodology is followed. To answer the review questions below, the SLR includes reviewing, investigating, and evaluating the existing research work.

A. Review Questions (RQs)

Research questions are presented here to assess and review the primary studies.

RQ1: What data sets were utilized to detect software vulnerabilities?
RQ2: Which machine learning techniques have been used for software vulnerabilities detection?
RQ3: What level of granularity is used to extract features?
RQ4: What feature extraction methods are used?
RQ5: What are the evaluation metrics used for vulnerabilities detection models?

B. Review Protocol

The procedure of the study search consisted of selecting digital sources, constructing a search string, doing an initial search, and then retrieving a collection of primary studies from digital sources.

Digital Sources Selection

Many searches were conducted to locate and select the most relevant research papers relative to the research questions. The following are the used online sources:

- IEEE Xplore
- Springer Link
- ACM
- Other journals & conferences

Search Strings Construction

After selecting the sources, the search string needs to be constructed to perform a comprehensive search to select the most relevant search studies to the topic as follows:

- (Vulnerability OR Software Vulnerability) AND (Detection OR Automatic Detection OR Discovery) AND (Machine Learning OR Deep Learning OR Deep Neural Networks)
- (Automatic AND Vulnerability AND Detection OR Discovery) OR (Automatically AND Detect AND Vulnerabilities) OR (Automated AND Vulnerability AND Detection) AND (Machine Learning OR Deep Learning)

Inclusion and Exclusion Criteria

Search strings are used to retrieve all available research papers in the digital sources mentioned above. In order to select the primary studies from the initial search result, inclusion and exclusion criteria were designed.

Inclusion criteria:

- the text is written in the English language.
- relevant to the detection of software vulnerabilities.
- a paper that has been published in a journal or at a conference.
- publications that have been peer-reviewed.

Exclusion criteria:

- research studies that are irrelevant to the search string.
- without any empirical research or results.
- outdated search papers.

- the structure of the study is disorganized.

Using the search strings, we acquired a huge number of papers as preliminary studies from digital sources. Then, using the above-mentioned exclusion and inclusion criteria, we chose the primary studies based on the title, abstract, and keywords.

III. BACKGROUND

A. Software Vulnerability: Different definitions

Software security vulnerabilities (short for vulnerabilities), also known as security defects [5], security bugs [6], and software weaknesses [7], are described by the Mitre organization in charge of the Common Vulnerability and Exposures (CVE) dictionary as a:

"A flaw in the computational logic, such as code, is identified in software and hardware components that, when exploited, has a detrimental influence on confidentiality, integrity, or availability." [2]. Among various vulnerabilities, code-based vulnerabilities are the reason for most exploits. [8] (Software, data, or commands that may cause a violation of security policy [9]).

"An instance of an error in the specification, development, or configuration of software that allows its execution to violate a security policy" is what a software vulnerability is [9].

National Institute of Standards and Technology (NIST):

"A threat source could exploit or cause a weakness in an information system, system security procedures, internal controls, or implementation.".

B. Prior studies related to vulnerability detection.

Vulnerability detection is a technique for detecting flaws in the software. Traditional vulnerability detection methods rely on human involvement and are prone to false positives and negatives [10] [11]. It requires a significant amount of time and effort from domain experts to generate handcrafted features to identify vulnerable code. These handcrafted features frequently fail to capture program semantic and structural information.

There are two methods for source code-based static vulnerability detection: code similarity-based and pattern-based. Code similarity-based detectors can only detect vulnerabilities incurred by code cloning as VUDDY [12] and VulPecker [13].

VUDDY and VulPecker both have high false-negative rates of 18.2 % and 38 %, respectively.

There are three steps to the code similarity-based technique. Breaking down a program into code portions is the first step. Tokens, trees, and graphs are instances of code snippets that are abstractly represented in the second stage. The third step is to determine the similarity between code snippets using the abstract representations established in the previous phase.

Pattern-based methods, which can be further subdivided into rule-based and machine learning-based methods.

In rule-based methods, vulnerability detection is dependent on rules that are usually generated manually by human experts. (E.g., Flawfinder [14], RATS [15], Checkmarx [16]). These tools often have high false-positive rates and/or high false-negative rates [17].

Thus, it's important to combine program analysis technologies with machine learning to help software security research enhance automated vulnerability detection is being developed.

IV. MACHINE LEARNING-BASED VULNERABILITY DETECTION

One of the key research directions is to develop intelligent vulnerability detection techniques that act on source code. The following three sub-categories can be found: vulnerability detection methods based on software metrics [18] [19] [20] [21], anomaly detection technique for detecting vulnerabilities by looking for abnormal patterns [22], and vulnerable pattern learning [23].

The intelligent vulnerability detection approaches make use of software syntax and semantic information to enhance detection performance.

V. DEEP LEARNING-BASED VULNERABILITY DETECTION

In the past few years, deep learning has received a lot of attention. Deep learning is a subset of machine learning in artificial intelligence (AI) which mimics the human brain's neuronal network. Deep learning models build a network that resembles the nervous system in the human body. It imitates the thought mechanism of humans. Deep learning is a network of nodes, each of which functions as a neuron.

Deep learning has recently sparked a surge of interest in the field of software engineering. In recent studies, deep learning technology was used to investigate automated intelligent software security analysis. To extract features, the researchers used techniques inspired by previous DL applications, such as automated language processing.

A Deep Neural Network (DNN) consists of an input layer, an output layer, and some hidden layers between them. This network is capable of handling not only unstructured and unlabeled data, but also non-linearity. One of the most significant benefits of using neural networks is their ability to learn features on their own. [24].

It's crucial to highlight that deep learning techniques have been widely used to discover software weaknesses. It is applied to the fields of the prediction of defects [25] [26] [27] [28] [29], finding erroneous source code [30], program analysis [31] [32] [33], code clone detection [34], ,

recognizing functions in binaries [32] and recently it has been used in software security vulnerability detection [24] [35] [36] [37] successfully.

VI. State-of-the-art Studies

In this section, we review and analyze the state-of-the-art studies in vulnerability detection using machine learning and deep learning where research questions are tackled:

Guanjun et al. [37] Proposed POSTER, an approach for function-level vulnerability detection on cross-project domains. They used Abstract Syntax Tree (AST) for function representation. To capture the features in a function, they used Bidirectional Long Short-Term Memory (BLSTM).
Runhao et al. [38] Presented an approach for detecting vulnerabilities using Bidirectional Long Short-Term Memory (BLSTM) based on the extraction of semantics features of function names from the intermediate representation of source code to distinguish between vulnerable and secure functions. They worked on CVE entries reported from 2008 to 2018 and selected eligible function names from vulnerabilities information. The approach is able to detect multiple kinds of vulnerabilities and reduced the false-positive rate.

Rebecca et al. [39] Proposed a system for vulnerability detection based on machine learning using the function-granularity level. They developed a special lexer representation intended to capture the relevant meaning of critical tokens from the raw source code of each function. They used a labeled dataset by static analysis tools which are collected from several open-source projects and Assurance Reference Dataset (SARD) [40] and they used Ensemble learning on neural representations based on Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to extract valuable features.

Zhen et al. [35] Presented VulDeePecker, which is classified as a pattern-based method for detecting vulnerabilities. VulDeePecker is the first system showing the feasibility of utilizing deep learning to discover vulnerabilities at the granularity level of API function calls. To represent the program, it uses code gadgets. At the beginning, the program is converted to intermediate representation to sustain the dependency between program elements and finally, this intermediate representation is converted to vectors which are the deep neural network's required input. The results showed that VulDeePecker achieved significantly fewer false-negative rates than other methods. VulDeePecker dataset is derived from both the National Vulnerability Database (NVD) [41] and the Software Assurance Reference Dataset (SARD) [40] and it is available at [42]. Although the code gadget paradigm allows for data and control dependencies, it's worth mentioning that VulDeePecker only provides data flow analysis but can't support control flow analysis.

Deqing et al. [36] Presented µVulDeePecker, the first system based on deep learning to find vulnerabilities in multiclass which extends VulDeePecker [35] work. µVulDeePecker accommodates not only data dependence but also control dependence which leads to higher effectiveness in multiclass vulnerability detection. µVulDeePecker dataset is derived from both SARD [40] and NVD [41].

Zhen et al. [43] Proposed SySeVR, the first system that used deep learning for vulnerability detection in the source code of the program in the slice-granularity level which is finer than presented in VulDeePecker. The focus of the research is to obtain a representation of the program that can contain both the syntax and semantic information related to the vulnerability. SySeVR dataset is derived from both NVD [41] and SARD [40] and it is available at [44]. SySeVR overcomes the weaknesses of VulDeePecker mentioned above by working on semantic information caused by data and control dependence.

Zeki et al. [45] Proposed a machine learning-based model for predicting vulnerabilities from the program code by using Multi-Layer Perceptron (MLP) algorithm and based on Abstract Syntax Tree (AST) for source code representation to extract the features to be able to perform intelligent analysis and distinguish between vulnerable and invulnerable code fragments. The code fragment is function-granularity level. Resulted AST of the source code fragment is converted into a numerical array representation to keep the syntactic and semantic relations of the source code fragments. The used dataset in this study is collected from several open-source projects and NVD [41].

Some of the prior studies in the field of vulnerability detection tried to evaluate theories that are a correlation between software characteristics: complexity, coupling, etc. [19] [46] [47] [18]. Mohammed et al. [20] Aim to use code metrics as features to detect software vulnerabilities based on deep learning with a fine granularity level. The findings showed that code metrics are good, but not ideal to be used as features in DL-based vulnerability detection. This approach is applied only for one type of vulnerabilities

Zhen et al. [48] Propose VulDeeLocator, the first deep learning-based vulnerability detector from source code that can achieve a high vulnerability locating precision and high detection capability as they introduced the notion of granularity refinement to locate the vulnerable lines of code using Bidirectional Recurrent Neural Network (BRNN). In comparison to the state-of-the-art detector [43], VulDeeLocator improved the vulnerability detection F1-measure, false-positive rate, and false-negative rate by 9.8 %, 7.9 %, and 8.2 %, respectively, as well as the vulnerability locating precision by 4.2X. VulDeeLocator dataset is combined from both (NVD) and (SARD) datasets and its dataset is available at [49].

Ning et al. [50] Presented VulHunter, a system to detect vulnerabilities based on bytecode using deep learning. VulHunter is the first system to detect vulnerabilities in source code using bytecode features. They worked only on two types of vulnerabilities: SQL injection and Cross-Site Scripting.

VulHunter outperformed prior approaches in terms of false-positive rate and false-negative rate. VulHunter calculates the resemblance between the target software and the vulnerability template, unlike earlier studies that directly decide if the target software contains vulnerabilities. However, they are incapable of detecting complex vulnerabilities.

Guanjun et al. [51] Suggest a vulnerability detection benchmarking framework. This system provides six common neural network models as well as two distinct code embedding approaches, allowing for one-click vulnerability detection model creation and testing. The used dataset is collected from SARD and real-world vulnerability ground truth dataset containing manually labeled vulnerable functions as they are working on the function granularity level.

VII. RESULTS

In this study, we reviewed 12 primary research papers on vulnerability detection using both machine learning and deep learning techniques that were published from 2017 to 2021. Table 1 provides a summary of these studies, which covers the below points:

A. *Datasets*

A dataset is a collection of data that is usually related to a specific field of study. Seven of the state-of-the-art studies we discussed worked on the National Vulnerability Database (NVD) dataset [41] in combination with the Software Assurance Reference Dataset (SARD) [40], whereas the rest relied on either one of them only or open-source projects. NVD is a repository of all Common Vulnerabilities and Exposures (CVE) [2] records which reached 156468 records. SARD is a group of known security weaknesses identified as to test cases and test suites. Both datasets have an issue that their data may not be representative of real-world software products which affect the performance results of many studies. Dataset imbalance is a prevalent issue in the majority of the reviewed research that results in certain biases in the results.

B. Machine learning techniques

In the literature, various machine learning and deep learning algorithms for vulnerability detection have been introduced. The Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) were used in the majority of the primary studies. Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BLSTM) are examples of applied RNN algorithms.

C. Evaluation Metrics

Vulnerability detection models should be assessed to see how effective and efficient they are. The primary studies that reviewed were using a variety of methodologies to assess the efficacy of their proposed technique. Precision, recall, false-positive rate, false-negative rate, and others are all examples of measurements. In the primary studies, both false-positive rates and false-negative rates were commonly applied.

The first evaluation metric is precision that measures how many results are relevant? Using the below formula:
Precision = true positives / (true positives + false positives).
The second widely utilized evaluation metric for detecting vulnerabilities is recall, which is defined as the percentage of all discovered positives (total relevant results) that were accurately predicted using the below formula:
Recall = true positives / (true positives + false negatives).
When the detection model predicts a security vulnerability that doesn't exist, it's called a false positive which is the third most commonly used evaluation metric. A false negative is the polar opposite of the FP, implying that there is no vulnerability when, in reality, there is,
which is the fourth most commonly used evaluation metric.

D. Granularity level

Because it's important to determine not only whether the software is vulnerable or not, but also to pinpoint where the vulnerabilities are located. Several reviewed works used function granularity for vulnerability detection [37] [38] [45] [59] and they have identified the reliance on it as a drawback as it is unsatisfactory and inaccurate. Other studies to work on finer granularity as code slice [20] [35] [43] [50]. As a result, pinpointing the exact position of vulnerabilities inside the functions becomes more challenging.

E. Feature-Extraction Approach

By applying Machine learning-based techniques to detect vulnerabilities, features can be developed automatically [35] [36] [37], eliminating the need for human experts to manually define the features, which frequently overlooks many vulnerabilities. In addition to one of the primary studies used code metrics for defining code characteristics [20].

F. Challenges and Issues

The most common challenges and problems of reviewed papers are explained as follows:

1) Type of vulnerabilities

Studies experiments focus on a few types of vulnerability syntax characteristics based on the used datasets which affect the overall coverage as the datasets might not have been reflective of real-world software products. For instance, Mohammed et al. [20] worked on only one kind of vulnerability, and Zhen et al. [43] focused on four kinds. Zhen et al. [35] Worked on function calls only. More comprehensive vulnerability syntactic characteristics will need to be identified in future research.

2) Dataset labelling

Some studies used a static analyzer for labeling the dataset [59] and others added it manually, which result in a mislabeled dataset and affected the result accuracy. These studies highlighted that as a limitation, and it is necessary to improve the labeling strategy for reliable results.

3) Single model

Experiments in state-of-the-art studies focus on employing a single model to discover several types of vulnerabilities.

4) Vulnerability localization

For detecting vulnerabilities, the reviewed studies used function and slice level detection, which might need to be enhanced to pin down the location more precisely where a vulnerability resides.

5) Model Selection

For the same data, different deep learning models have varying learning capacities, therefore determining which model is best for learning feature data and dealing with software security defects require more research.

6) Programming language

Almost all the research studies use the same programming language which is C/C++ to detect vulnerability in data sets. Many papers have mentioned that as a shortcoming that should be investigated more in the future to use other programming languages.

Figure 3 presents the classification of studies over various digital sources.

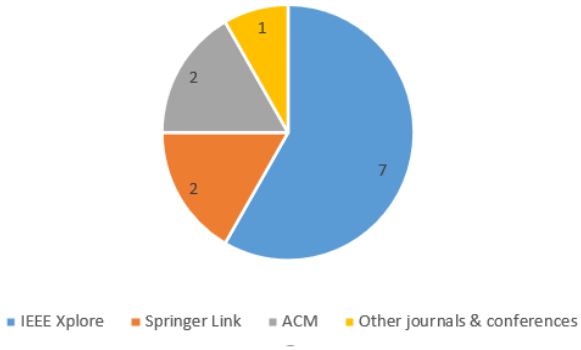

Fig. 3. Primary studies classification

Following the answers to research questions listed earlier in Section 2.1. RQ1 is investigating the datasets that are used by researchers to train and test the vulnerability detection models in their studies. Most primary studies used Software Assurance Reference Dataset (SARD) [52] and National Vulnerability Database (NVD) [41] which are public and free access datasets. RQ2 concerns machine learning techniques, which are frequently used to build vulnerability detection models and the result shows that three techniques are significantly used in the automated detection of vulnerabilities. The first one is a machine learning technique Multi-Layer Perceptron (MLP) and the other two techniques are deep learning neural networks techniques Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Other kinds of neural networks that could be used for vulnerability detection need to be applied. RQ3 is related to the granularity level of source code to represent the program that is used to extract the features of the vulnerabilities. Some primary studies worked on function level and the others tried to work with finer granularity level to pin down the precise location of the vulnerabilities using slice level. The result for RQ4 which analyses the feature extraction method used, and by using the automatic analysis of deep learning and machine learning methods for the detection of vulnerabilities the features are extracted automatically. RQ5 is about the applied evaluation metrics to evaluate vulnerability detection models, which are false-positive rate, false-negative rate, precision, recall, and F1 Score.

VIII. Conclusion and Future Work

Machine Learning and deep learning technologies have paved the way for researchers to investigate possible software security flaws, and several automated methods have been proposed as a result. Recently, automatic detection of security vulnerabilities has become an important research area. In this study, we reviewed and analyzed the recent existing studies of software vulnerability detection based on deep learning and machine learning techniques. We worked on 12 primary studies between 2017 and 2021. Additionally, the SLR summarizes these primary studies based on the used datasets, ML or DL techniques, granularity level, feature extraction approach, and evaluation metrics. The key findings are summarized as follows:

- Most primary studies worked on NVD and SARD datasets, and others used open-source programs.
- CNN, RNN, and MLP were the most commonly used ML and DL techniques for vulnerability detection in primary studies
- Most primary studies were working on the slice level of granularity, and others worked on the function level.
- Vulnerable code features could be extracted automatically without the need for human experts when applying deep learning and machine learning techniques.
- Precision, recall, false-positive rate, and false-negative were the most commonly used evaluation metrics in the primary studies.

All previously discussed deep learning and machine learning models for detecting software vulnerabilities have some gaps that require further improvements in future studies as follows:

- Most of the above-mentioned studies' experiments focus only on one or a few kinds of vulnerability syntax characteristics
- Datasets in some studies are not real-world software and are not sufficient in some other studies which affect the result.
- Algorithms used for generating syntactic and semantic information for vulnerability detection needs more improvement to provide more information.
- Applying a single model to detect multiple kinds of vulnerabilities.

TABLE I
A SUMMARY OF STATE-OF-THE-ART STUDIES

REFERENCE	USED DATASETS	SOURCE NAME	ML/DL TECHNIQUES	EVALUATION METRICS	GRANULARITY LEVEL	FEATURE-EXTRACTION APPROACH	CHALLENGES AND ISSUES
(Rebecca et al., 2018) [59]	Debian, The Juliet Test Suite programs, and open source projects from GitHub	IEEE Xplore	CNN, RNN, RF	Precision, recall , and false-positive rate	Function-level	Automatically learned features	Dataset labels and Vulnerability detection localization
(Zeki et al., 2020) [45]	Open-source projects and NVD	IEEE Xplore	MLP	Precision, recall, and F1 Score	Function-level	Automatically learned features	Vulnerability detection localization
(Zhen et al., 2021) [43]	NVD, SARD	IEEE Xplore	RNN, CNN, DBN	False-positive rate, false-negative rate, and precision	Slice-level	Automatically learned features	Vulnerability syntax characteristics, dataset, programing language, and single model
(Mohammed et al., 2020) [20]	SeVC and SyVC Dataset	IEEE Xplore	MLP, RNN	False-positive rate, false-negative rate, and precision	Slice-level	Code Metrics	Types of vulnerabilities, code metrics and model architecture
(Zhen et al., 2018) [35]	NVD, SARD	Other conference	RNN	False-positive rate, false-negative rate, and precision	Slice-level	Automatically learned features	Types of vulnerabilities, control flow analysis, and Programing language
(Deqing et al., 2019) [36]	NVD, SARD	IEEE Xplore	RNN	False-positive rate, false-negative rate, and F1 Score	Slice-level	Automatically learned features	Types of vulnerabilities, Vulnerability detection localization, and Programing language
(Guanjun et al., 2017) [37]	Open-source projects	ACM	RNN	recall	Function-level	Automatically learned features	Dataset size, vulnerability detection localization, and dataset labels
(Guanjun et al., 2020) [51]	SARD	Springer Link	CNN, RNN, DNN	Precision, recall	Function-level	Automatically learned features	
(Zhen et al., 2021) [48]	NVD, SARD	IEEE Xplore	BRNN	false positive rate, false negative rate,Precision,F1 Score	Slice-level	Automatically learned features	Dataset and Programing language
(Ning et al., 2020) [50]	NVD, SARD	Springer Link	RNN	false positive rate, false negative rate,Precision,recall,F1 Score	Slice-level	Automatically learned features	DL Model and complex vulnerabilities
(Runhao et al., 2019) [38]	NVD, Open source programs	IEEE Xplore	RNN	F2-score, false-positive rate, recall, and precision	Function-level	Automatically learned features	Inaccurate and false results
(Xiao et al., 2021) [56]	NVD, SARD	ACM	GNN	False-positive rate, false-negative rate, and F1 Score	Slice-level	Automatically learned features	Dataset and programing language

REFERENCES

[1] C. Williams, "Anatomy of OpenSSL's Heartbleed: Just four bytes trigger horror bug," April 2014. [Online]. Available: https://www.theregister.com/2014/04/09/heartbleed_explained/.

[2] "CVE website," [Online]. Available: https://cve.mitre.org.

[3] "A Redscan report , 'NIST security vulnerability trends in 2020: an analysis'," [Online]. Available: https://www.redscan.com/media/Redscan_NIST-Vulnerability-Analysis-2020_v1.0.pdf.

[4] D. A. Ramos and D. Engler, "Under-constrained symbolic execution: Correctness checking for real code," *in Proc. USENIX Secur. Symp.*, 2015.

[5] C. D. Sestili, W. Snavely and N. M. VanHoudnos, "Towards security defect prediction with AI," 2018.

[6] D. Votipka, R. Stevens, E. Redmiles, J. Hu and M. Mazurek, "Hackers vs. Testers: A comparison of software vulnerability discovery processes," *IEEE Symp. Secur. Privacy (SP)*, p. 374–391, 2018.

[7] Y. J. Lee, S.-H. Choi, C. Kim and K.-W. Park, "Learning binary code with deep learning to detect software weakness," *in Proc. KSII 9th Int. Conf. Internet Symp. (ICONI)*, 2017.

[8] G. McGraw, "Automated code review tools for security," *IEEE Computer*, 2008 .

[9] I. V. Krsul, "Software vulnerability analysis," no. PhD dissertation, Purdue University, West Lafayette., 1998.

[10] B. Johnson, Y. Song, E. Murphy-Hill and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *in Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[11] B. Liu, L. Shi, Z. Cai and M. Li, "Software Vulnerability Discovery Techniques: A Survey," in *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2013.

[12] S. Kim, S. Woo, H. Lee and H. Oh, "VUDDY: A scalable approach for vulnerable code clone discovery," *in Proceedings of the 38th IEEE Symposium on Security and Privacy*, 2017.

[13] Z. Li, D. Zou, S. Xu and H. Jin, "VulPecker: An automated vulnerability detection system based on code similarity analysis," *in Proceedings of the 32nd Annual Conference on Computer Security Applications. ACM*, p. 201–213, 2016.

[14] "FlawFinder," 2018. [Online]. Available: http://www.dwheeler.com/flawfinder.

[15] "Rough Audit Tool for Security," 2014. [Online]. Available: https://code.google.com/.

[16] "Checkmarx," 2018. [Online]. Available: https://www.checkmarx.com/.

[17] F. Yamaguchi, "Pattern-based vulnerability discovery," *Ph.D. dissertation, University of G¨ottingen*, 2015.

[18] M. Zagane and M. K. Abdi, "Evaluating and comparing size, complexity and coupling metrics as Web applications vulnerabilities predictors," *Int. J. Inf. Technol. Comput. Sci*, vol. 11, pp. 35-42, 2019.

[19] I. Chowdhury and M. Zulkernine, "Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?," *Proc. ACM Symp. Appl. Comput. (SAC)*, p. 1963, 2010.

[20] M. Zagane, M. K. Abdi and M. Alenezi, "Deep Learning for Software Vulnerabilities Detection Using Code Metrics," *IEEE Access , vol. 8,* pp. 74562 - 74570, 17 April 2020.

[21] H. Alves, B. Fonseca and N. Antunes, "Experimenting machine learning techniques to predict vulnerabilities," *In Dependable Computing (LADC)*, pp. 151-156, 2016.

[22] F. Yamaguchi, A. Maier, H. Gascon and K. Rieck, "Automatic Inference of Search Patterns for Taint-Style Vulnerabilities," *IEEE Symposium on Security and Privacy*, 2015.

[23] S. M. Ghaffarian and H. R. Shahriari, "Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey," *ACM Computing Surveys*, 2017.

[24] C. Catal, A. Akbulut, S. Karakatič, M. Pavlinek and V. Podgorelec, "Can we predict software vulnerability with deep neural network?," 2016.

[25] X. Yang, D. Lo, X. Xia, Y. Zhang and J. Sun, "Deep learning for just-in-time defect prediction," *in Proceedings of IEEE International Conference on Software Quality, Reliability and Security,Vancouver, BC, Canada*, p. 17–26, 2015.

[26] T. L. a. L. T. S. Wang, "Automatically Learning Semantic Features for Defect Prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2017.

[27] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect," *Cluster Comput 22*, p. 9847–9863, 2018.

[28] J. Li, P. He, J. Zhu and M. R. Lyu, "Software Defect Prediction via Convolutional Neural Network," in *International Conference on Software Quality, Reliability and Security (QRS)*, Prague, Czech Republic, 2017.

[29] S. Wang, T. Liu and L. Tan, "Automatically Learning Semantic Features for Defect Prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 2017.

[30] X. Huo, M. Li and Z.-H. Zhou, "Learning unified features from natural and programming languages for locating buggy source code," *in Proceedings of the 25th International Joint Conference on Artificial Intelligence, New York, USA*, p. 1606–1612, 2016.

[31] M. White, C. Vendome, M. Linares-Vasquez and D. Poshyvanyk, "Toward deep learning software repositories," *Mining Software Repositories*, p. pp. 334–345, 2015.

[32] C. R. Shin, D. Song and R. Moazzezi, "Recognizing functions in binaries with neural networks," in *2015*, Washington , D.C., USA, Proceedings of the 24th USENIX Security Symposium.

[33] C. L. Q. F. H. Y. L. S. a. D. S. X. Xu, "Neural network-based graph embedding for cross-platform binary code similarity detection," *ACM SIGSAC Conference on Computer and Communications Security*, p. pp. 363–376, 2017.

[34] M. White, M. Tufano, C. Vendome and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, Singapore, 2016.

[35] Z. Li, D. Zou, S. Xu and X. Ou, "VulDeePecker: A deep learning-based system for vulnerability detection," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium (NDSS'18)*, 2018.

[36] D. Zou, S. Wang, S. Xu, Z. Li and H. Jin, "μVulDeePecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Trans. Dependable Sec. Comput.*, p. 1–1, 2019.

[37] G. Lin, J. Zhang, W. Luo and L. Pan, "POSTER: Vulnerability discovery with function representation learning from unlabeled projects," *in Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA*, p. 2539–2541, 2017.

[38] R. Li, C. Feng, X. Zhang and C. Tang, "A Lightweight Assisted Vulnerability Discovery Method Using Deep Neural Networks," *IEEE Access*, vol. 7, pp. 80079 - 80092, June 2019.

[39] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood and M. McConley, "`Automated vulnerability detection in source code using deep representation learning,"," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018.

[40] "Software Assurance Reference Dataset," 2018. [Online]. Available: https://samate.nist.gov/SRD/index.php.

[41] "NVD," [Online]. Available: https://nvd.nist.gov/.

[42] "VulDeePecker Dataset," [Online]. Available: https://github.com/CGCL-codes/VulDeePecker.

[43] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities," *IEEE Transactions on Dependable and Secure Computing (Early Access)*, pp. 1 - 1, 13 January 2021.

[44] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, "SeVC and SyVC Dataset. [Online].," 2018. [Online]. Available: Available: https://github.com/SySeVR/SySeVR/.

[45] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Çomak and L. Karaçay, "Vulnerability Prediction From Source Code Using Machine Learning," *IEEE Access*, vol. 8, pp. 150672 - 150684, 2020.

[46] Y. Shin, A. Meneely, L. Williams and J. A. Osborne, "Evaluating complexity,code churn, and developer activity metrics as indicators of software vulnerabilities,," *IEEE Trans. Softw. Eng.*, vol. 37, pp. 772-787, 2011.

[47] L. Williams and Y. Shin, "An empirical model to predict security vulnerabilities using code complexity metrics," *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, p. 315, 2008.

[48] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu and H. Jin, "VulDeeLocator: A Deep Learning-based Fine-grained Vulnerability Detector," *IEEE Transactions on Dependable and Secure Computing (Early Access)*, pp. 1 - 1, 2021.

[49] "VulDeeLocator Dataset," [Online]. Available: https://github.com/VulDeeLocator/VulDeeLocator.

[50] N. Guo, X. Li, H. Yin, Y. Gao, J. Zhou, X. Luo, Q. Shen and Z. Xu, "VulHunter: An Automated Vulnerability Detection System Based on Deep Learning and Bytecode," *Information and Communications Security*, pp. 199-218, 2020.

[51] G. Lin, W. Xiao, J. Zhang, Y. Xiang, J. Zhou, X. Luo, Q. Shen and Z. Xu, "Deep Learning-Based Vulnerable Function Detection: A Benchmark," *Information and Communications Security*, pp. 219-232, 2020.

[52] "Software Assurance Reference Dataset," [Online]. Available: https://samate.nist.gov/SRD/index.php.

[53] "Wikipedia. [n.d.]. SandWorm," 15 August 2019 . [Online]. Available: https://www.cvedetails.com/cve/CVE-2014-4114.

[54] "Wikipedia. [n.d.]. DirtyCow," 15 August 2019 . [Online]. Available: https://en.wikipedia.org/wiki/Dirty_COW.

[55] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *in Advances in Neural Information Processing Systems,*, p. 1097–1105, 2012.

[56] X. Cheng, H. Wang, J. Hua, G. Xu and Y. Sui, "DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network," *ACM Transactions on Software Engineering and Methodology*, vol. 30, p. 1–33, 2021.

[57] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, p. 82–97, 2012.

[58] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?," *Empirical Software Engineering*, vol. 18, p. 25–59, 2013.

[59] R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood and M. McConley, "`Automated vulnerability detection in source code using deep representation learning,"," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec. 2018.