Static Analysis of Source Code Vulnerability Using Machine Learning Techniques: A Survey

Jingjing Wang
National Key Laboratory of Science and Technology on Information System Security
Beijing, China
wang_1094412598@163.com

Minhuan Huang*
National Key Laboratory of Science and Technology on Information System Security
Beijing, China
darbean@126.com

Yuanping Nie
National Key Laboratory of Science and Technology on Information System Security
Beijing, China
yuanpingnie@nudt.edu.cn

Jin Li
National Key Laboratory of Science and Technology on Information System Security
Beijing, China
tianyi198012@163.com

2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD) | 978-1-6654-1515-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICAIBD51990.2021.9459075

***Abstract*—With the rapid increase of practical problem complexity and code scale, the threat of software security is increasingly serious. Consequently, it is crucial to pay attention to the analysis of software source code vulnerability in the development stage and take efficient measures to detect the vulnerability as soon as possible. Machine learning techniques have made remarkable achievements in various fields. However, the application of machine learning in the domain of vulnerability static analysis is still in its infancy and the characteristics and performance of diverse methods are quite different. In this survey, we focus on a source code-oriented static vulnerability analysis method using machine learning techniques. We review the studies on source code vulnerability analysis based on machine learning in the past decade. We systematically summarize the development trends and different technical characteristics in this field from the perspectives of the intermediate representation of source code and vulnerability prediction model and put forward several feasible research directions in the future according to the limitations of the current approaches.**

Keywords—machine learning, software security, source code vulnerability, static analysis

I. INTRODUCTION

A vast variety of software applications have penetrated all aspects of social life which provide ubiquitous and intelligent services for people in the information era. While we appreciate the great convenience brought by them, the harm caused by exploitable vulnerabilities in software to computer security and user privacy is becoming more and more serious. The report [2] shows that Microsoft's 47000 developers generate nearly 30000 errors a month. According to Coralogix [3], the average developer produces 70 errors for every 1000 lines of code, and it takes 30 times longer to fix one than to write a single line of code. In the United States alone, $113 billion is spent each year on identifying and repairing product defects.

To further alleviate the adverse impact on companies and users [5-7] caused by accidental data leakage, expose or alter sensitive information, system disrupts or crash and so on, developers strive to explore potential vulnerabilities and repair them as soon as possible by correctly identifying security vulnerabilities from a variety of automated testing tools and a large number of errors in defect reports.

Compared with compiled binary code [8-10], source code possesses more transparent and abundant semantic information. The research on the source code-oriented vulnerability analysis method helps to facilitate the understanding of the vulnerability problem, leading to a more comprehensive analysis of the shortcomings of conventional approaches in the meanwhile.

According to whether the program needs to be executed during the analysis procedure, the vulnerability analysis approach for the source code can be divided into static analysis and dynamic analysis. The dynamic analysis draws lessons from the idea of software testing and explores the security errors of the source code by analyzing the input and output test cases or monitoring the running status of the code at different levels [11]. This method has high accuracy and low false alarm rate, and the construction of test cases is still the focus of the research at present.

Static analysis refers to the process of finding the code with security defects by extracting the source code model and vulnerability rules to analyze the instructions and structure of the source code and mastering the logic and function of program execution without running. In this survey, we concentrate on the research of source code vulnerability analysis using machine learning techniques, which is a kind of static analysis.

A series of vulnerability analysis studies for source code has been implemented to assist early security checks [12-18]. And machine learning technology has made remarkable success in multiple research and application scenarios [19, 20]. The fruitful research of machine learning technology has brought tremendous changes to further enhance the automation of source code vulnerability analysis and the efficiency of vulnerability mining.

We review the relevant research on source code vulnerability analysis based on machine learning in recent years and systematically summarize the studies from the perspectives of the intermediate representation of source code and machine learning algorithm model. Furthermore, we analyze the characteristics and limitations of different approaches. We try to explore the internal mechanism of diverse algorithm models in source code vulnerability analysis in the meanwhile. Finally, we briefly describe the feasible future directions in this field at the present stage.

* Corresponding author: Minhuan Huang.

A. Related Reviews

A series of studies have systematically summarized the methods of source code vulnerability analysis utilizing machine learning technology or state-of-the-art research achievements of related technical fields. Ghaffarian and Shahriari [5] reviewed different types of work using machine learning and data mining techniques for software vulnerability analysis and discovery. They concluded that research in this field is still in an immature state. Zeng et al. [21] summarized in detail the four game-changers that have had a significant impact on the field of deep learning-based vulnerability detection and reviewed the methods and solutions that have been established or extended based on them. Lin et al. [22] reviewed the current software vulnerability detection based on deep learning/neural networks to understand how state-of-the-art research uses neural technology to learn and understand code semantics to facilitate vulnerability detection. Lin et al. [23] proposed a benchmark framework for building and testing deep learning-based vulnerability detection, to compare and evaluate the performance of mainstream neural network models and provide a reference for function-level vulnerability detection. Allamanis et al. [24] discussed how to apply it to the design of probabilistic models based on the similarities and differences between programming languages and natural languages, and proposed the source code model of probabilistic machine learning and the classification of its applications. Pan et al. [25] discussed the feasibility and advantages of deep learning technology in software vulnerability analysis and detection. Malhotra [26] summarized 64 preliminary studies and different types of machine learning techniques, then concluded that machine learning technology can predict software fault proneness, but its application is still limited.

II. Overview

An overview of a common machine learning-based vulnerability analysis framework is described in Figure 1. According to the different goals of the sub-process, we divide the processing procedure into two phases: feature extraction and vulnerability prediction.

The feature extraction phase establishes the correlation between source code and vulnerability, and aims to obtain the most discriminative feature of the vulnerability in the input, and then convert it into numerical vectors as the input of the next stage of the learning model.

This phase contains two inputs: vulnerability training data with labels and target source code for analysis. Vulnerability data is obtained from a large vulnerability repository. Some of the studies also combined input in the form of natural language, vulnerability reports for example. Afterwards data preprocessing operations such as labeling and slicing are performed to obtain the researcher's ideal vulnerability sample format, which is usually a time-consuming and cumbersome task.

The majority of the researches focus on designing exquisite vulnerability feature extraction strategies, promising to enhance the correlation between features and vulnerabilities while capturing abundant source code information to further promote the efficiency of model representation learning. The intermediate representation of the source code, as a bridge to transform the text programming language into numerical vectors, is also essential for the algorithm to effectively learn the source code features and establish a vulnerability prediction model.

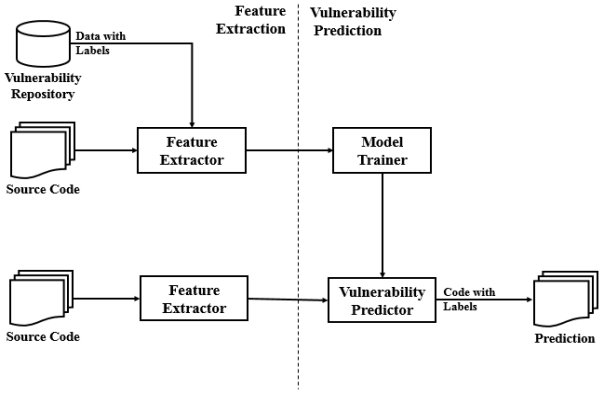

Fig. 1. Vulnerability analysis framework based on machine learning.

The core of the vulnerability prediction phase refers to the selection and construction of the prediction model. In this paper, according to whether the manual predefinition of source code features by domain experts is required in the feature extraction phase, we divide machine learning algorithms into two categories: conventional machine learning algorithms and deep learning algorithms.

Conventional machine learning algorithms require domain experts to predefine the source code features manually, and the effect of classification largely depends on the definition and selection of features. Kronjee et al. [27] argue that utilizing the right features is more important than using a machine learning algorithm. In contrast, for deep learning algorithms, the robust learning capacity of the model largely depends on sophisticated deep neural networks. At present, the majority of the experience applied to vulnerability prediction originates from the success of deep learning technology in other fields, typical examples are natural language processing and image recognition.

Ultimately, according to the vulnerability prediction results, the vast majority of studies usually evaluate the performance of the model from the following four aspects: precision, recall, accuracy, and F-measure. Besides, we figured out that F-measure, as the harmonic mean of accuracy and recall, is the most frequently used evaluation index.

In summary, the motivation of this paper comes from the two phases of source code vulnerability analysis based on machine learning mentioned above, while concentrating on the intermediate representation of source code and the machine learning model. Based on these two key points, the research progress in this field in the past decade has been systematically surveyed, and some conclusive insights have been put forward.

III. Intermediate Representation of Source Code

In this section, we review the different methods and evolution of source code representation in vulnerability analysis using machine learning techniques in recent years and

divide these methods into three categories from the point of view of feature extraction.

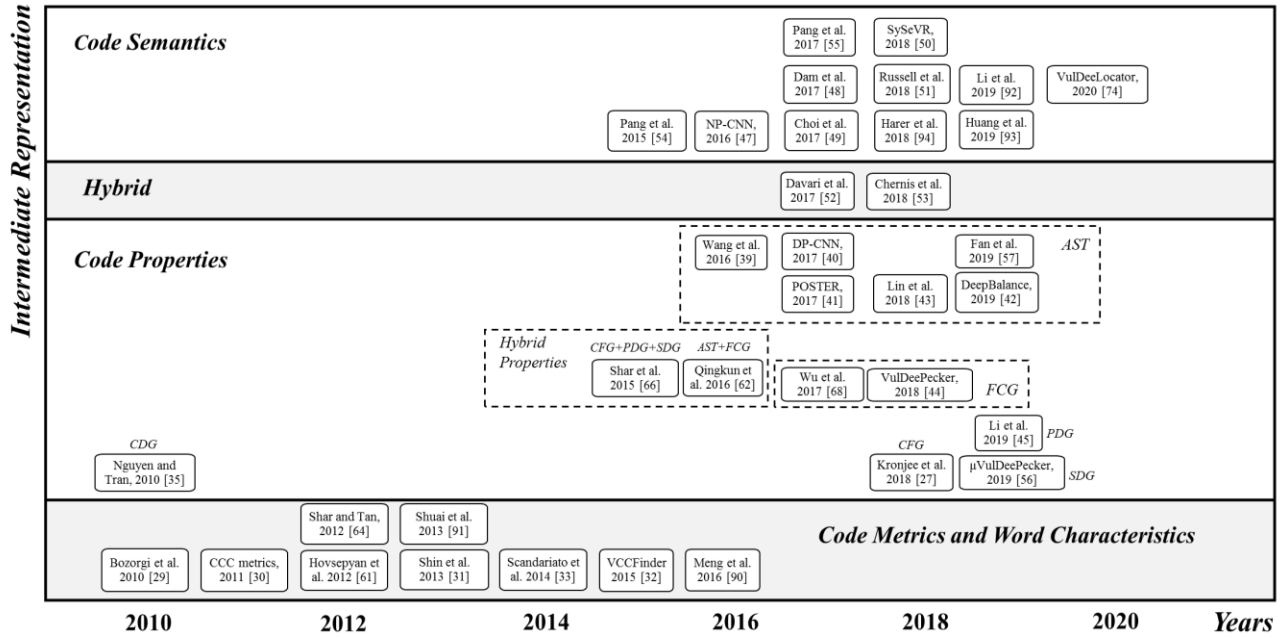

Fig. 2. The taxonomy of intermediate representation of source code.

It can be seen from figure 2, earlier vulnerability analysis concentrated on simple metrics and character characteristics of source code, with time going on, machine learning techniques explored further in this area, the notion of code properties was applied to intermediate representations of source code during the feature extraction phase. Nowadays, most approaches rely more on the automatic extraction of higher-level code semantics and syntax by sophisticated deep neural networks. It is worth noting that the changes in the intermediate representation of source code are complementary to the application and development of machine learning models.

To sum up, the general representation form with more domain adaptability and stronger correlation with vulnerability information is what we expect. More importantly, the strong representation learning capacity of deep learning algorithms makes this progress possible. We do not elaborate on the impact of machine learning algorithms on vulnerability detection in the modeling phase, for the reason that it is what we will discuss in the following sections.

A. Code Metrics and Word Characteristics

Some researches based on conventional machine learning consider the quantifiable measurement criteria of the whole source code in the feature definition stage, including complexity, coupling degree, the number of historical faults, and so on. Or they only pay attention to a specific word characteristic contained in them.

Bozorgi et al. [29] extract feature vectors from text fields, timestamps, cross-references, and other entries in existing vulnerability disclosure reports as input to SVMs to predict whether vulnerabilities will be exploited within a specific time range while implementing vulnerability classification, which aims to replace traditional heuristic algorithms that rely on expert knowledge on a small scale through large-scale statistical data.

Similarly, Chowdhury and Zulkernine [30] adopt statistical and machine learning techniques to construct a vulnerability prediction framework based on CCC (Complexity, Coupling, and Cohesion) metrics. Combined with the historical information of vulnerabilities, a model is built and trained to predict whether the entity is vulnerable according to the generated probability values.

In [31], the fault prediction model based on classical metrics is empirically studied, and it is proved that the fault prediction model constructed by using three traditional metrics: 18 complexity metrics, 5 code churn metrics, and one past fault history metric is universal for specialized vulnerability prediction models.

Another typical method is VCCfinder [32], which analyzes the statistical characteristics of each feature based on extracting a series of hypothetical submitted features to measure the similarities and differences of statistical distribution between this feature and the submitted classes with vulnerabilities.

Unlike the representation based on code metrics, some studies focus on the number or frequency of specific words in the source code as a feature description of vulnerability information.

In [33], defect prediction is carried out for the first time based on source code text mining, and a series of terms and related frequencies contained in the source code are represented as the feature of Java source files. It is worth noting that it also complements existing techniques for vulnerability analysis using code metrics.

B. Code Properties

The key insight underlying the intermediate representation of code properties is to consider the properties such as code structure, control flow dependency, and so on from a more global point of view. Notably, code properties are usually shown in a graph structure, so to a certain extent, it is also an intermediate representation based on graph/tree.

The code properties adopted in vulnerability analysis reviewed in this survey including Abstract Syntax Tree (AST), Control Flow Graph (CFG), Program Dependency Graph (PDG), Function Call Graph (FCG), Component Dependency Graph (CDG), System Dependency Graph (SDG).

Yamaguchi et al. [36] propose the concept of code property graph as a new method of source code representation firstly, which represents the properties of source code through a joint data structure that combines AST, CFG, and PDG. On this basis, an efficient detection template is produced, which enables us to model common vulnerabilities elegantly by traversing property graphs. After that, some studies [37, 38] have made further improvements to the code property graph.

Compared with the above-mentioned joint data structures that contain three code properties, many studies tend to focus on only one or two of them. As one of the first intermediate representations generated by code parsing, AST can better describe the program structure and retain more source code information. It is the basis for many other forms of code representation as well.

For the first time, Wang, Liu, and Tan [39] adopt the deep learning algorithm to automatically learn the semantic features of the source code from the extracted token vector. Rather than directly from AST, Kronjee, Hommersom and Vranken [27] make some changes by extracting CFG from the AST converted by the PHP source code file to extract vulnerability feature, since CFG can indicate the path that may be traversed during code execution.

Furthermore, there are other researches concerned with AST-based source code representation in the feature extraction stage [40-43], whose essence is to obtain the high-level and generalizable function representation of the source code from AST to reveal the code semantics. This is because the vulnerable programming patterns in functions may usually be associated with multiple lines of code.

The concept of code gadget proposed by Vuldeepecker [44] is similar to this idea. Code gadget is composed of multiple program statements that are semantically related to each other in terms of data dependencies or control dependencies. It is the first time that fine-grained source code representation is applied to neural network learning high-level representation.

The approach proposed in [45] is inspired by Vuldeepecker [44]. It generates program slices for each FCG according to the PDG, to obtain two kinds of code slices with both data dependency, data dependency and control dependency correspond to different code gadget as the intermediate representation of the source code and reveals the influence of different semantic information from code properties on the results of vulnerability analysis.

C. Code Semantics

Referring to the remarkable effect of deep learning technology in the application field of Natural Language Processing (NLP) [24], many studies apply similar methods to code vulnerability analysis by extracting semantic features of source code in the form of text. And combined with the structural characteristics of the neural network to capture different aspects of semantic information, which can obtain the high-level abstract representation.

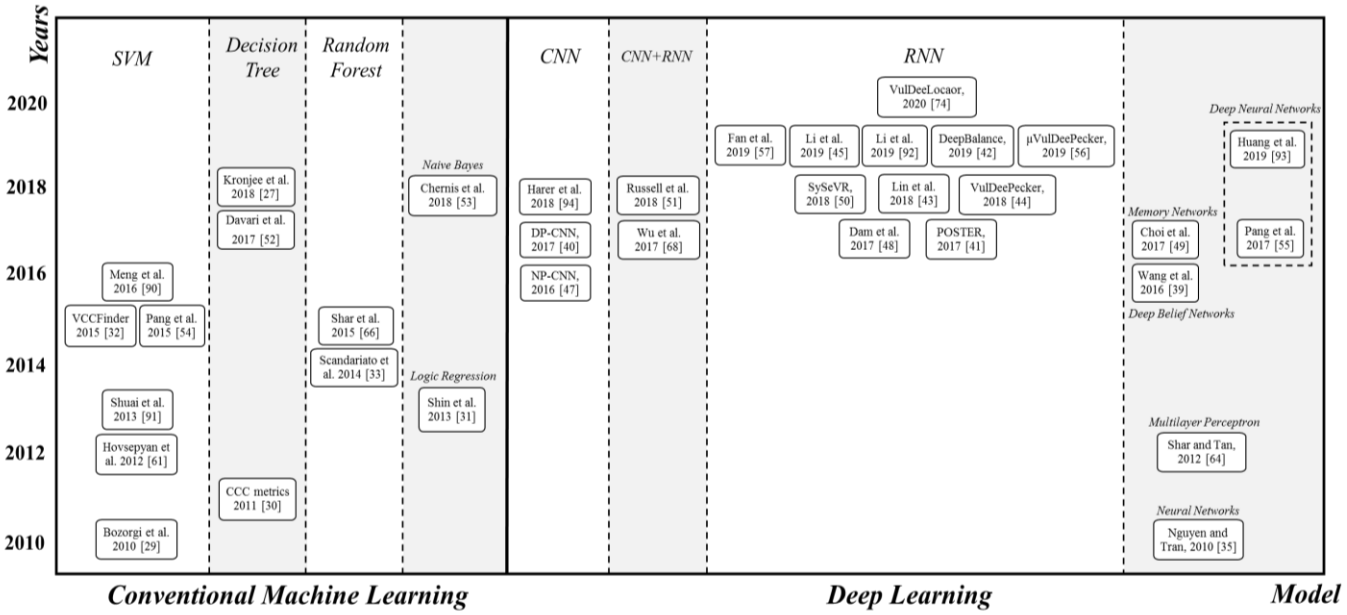

Fig. 3. The taxonomy of the model for vulnerability analysis.

According to the different characteristics of natural language and programming language, Huo, Li, and Zhou [47] design specific network structures for software defect reports and source code files. They extract the internal semantic features of the two languages respectively and then fuse the middle-level features of the language based on the cross-language feature fusion layer to generate a unified feature representation.

Dam et al. [48] take into account the impact of differences in semantic features between projects and within projects on vulnerability analysis, through the combination of intra-project syntactic features and cross-project semantic features to achieve a comprehensive representation of the semantic and sequential structure of source code tags.

These earlier studies analyze the language information such as project source code or defect report from more than one point of view. While other approaches only focus on the relationship between semantic and grammatical features and vulnerability information in the source code. They had achieved some groundbreaking work in this field while giving full play to the superior performance of deep learning neural network structure.

For the first time, Choi et al. [49] apply deep learning technology to problems that require syntactic and semantic knowledge in program analysis. After that, the first systematic framework SySeVR (Syntax-based, Semantics-based, and Vector Representations) [50] is proposed to detect vulnerabilities using deep learning. Furthermore, an approach proposed in [51] further improves the feature representation of sentence sentiment classification in NLP [19]. It captures longer dependencies while covering the entire embedded space of the token, thus aggregating and generating neural representations of fixed size.

D. Feature Selection and Combination

Reviewing the research progress of each method of the intermediate representation of code, we find that the comprehensive utilization and feature selection of different features is a research trend of code representation. Some studies consider extracting source code features from multiple angles, such as [47], [48], [52], [53].

The majority of them rely on deep learning neural networks to learn unified representation from different features. As we can see, the purpose of comprehensive utilization of features is to integrate features based on a specific representation, to capture more comprehensive program information, rather than just focusing on a single perspective.

Another strategy is feature selection. The approaches proposed in [54] and [55] generate source code features based on n-grams analysis and utilize similar sorting schemes for feature selection. These features are sorted by the scores obtained from the feature quality measurement, to identify the feature subset that shares specific attributes. This feature selection scheme excludes a large number of redundant features with small correlation or low importance and effectively alleviates the impact of higher feature dimensions and relatively few sample points on the accuracy of the model in practical application.

Besides, some improved algorithms apply code attention mechanism to enhance features. μVulDeePecker [56] improves code gadget based on VulDeePecker [44] and introduces code attention mechanism to capture more localization information related to vulnerability types in program statements. Similarly, an approach proposed in [57] based on the vectors extracted from AST embeds the attention layer after the BiLSTM, which aggregates the key nodes of strong importance to the feature sequence and enhances the effect.

It can be seen that these studies have begun to turn their attention to the enhancement and screening of existing feature information based on obtaining more extensive and more comprehensive source code information, to alleviate the impact of data defects at the present stage, in the meantime, improve the correlation between intermediate representation and vulnerability features. This transformation of interest is of great significance to improve the feature effectiveness and model accuracy of vulnerability detection in the modeling stage.

IV. Model for Vulnerability Analysis

In this section, we review the model algorithms deployed in the vulnerability prediction stage of source code vulnerability analysis in recent years. According to whether domain experts are required to predefine features in the feature extraction stage, we divide machine learning into two categories: conventional machine learning and deep learning.

We summarize all the machine learning algorithms implemented in the studies involved in this paper and show them in figure 3. In the field of early source code vulnerability analysis, conventional machine learning algorithms are the main tools for researchers to analyze vulnerabilities, despite the need for laborious feature definitions, their prominent classification performance and high model scalability contribute a large number of samples to the further exploration in this field and guide the future research direction.

It can be seen from figure 3 that deep learning technology has been gradually applied to source code vulnerability analysis around 2016, and achieved vigorous development in recent years. Compared with conventional machine learning technology, the application of deep neural networks liberates people from complex feature engineering.

Although the evidence indicates the vulnerability prediction result of the deep neural networks is better than conventional machine learning classifiers, we can not assert that the deep neural networks can completely replace the conventional machine learning algorithm. In sum, the exploration of deep learning technology in the field of security is not comprehensive and immature, due to numbers of defects such as datasets, model applicability, time efficiency, and so on limit the development of it.

Apart from this, the conventional machine learning algorithm is fast and flexible which can be reasonably implemented in the combination with other vulnerability analysis techniques, to make up for the inherent shortcomings of multiple algorithms. Therefore, the research of conventional machine learning also remains advancing.

A. Conventional Machine Learning

We reviewed some studies on source code vulnerability analysis adopting conventional machine learning algorithms such as Support Vector Machine (SVM), Decision Tree, Random Forest, Logical Regression, Naive Bayes, and so on. In these studies, we focus on several cases of vulnerability classification based on SVM. Moreover, we summarize the experimental comparison of other conventional machine learning algorithms.

1) SVM-based model: In our investigation, the accurate and efficient classification ability of SVM makes it the most frequently adopted model among these algorithms. In a nutshell, SVM aims to maximize the distance between the edge samples of a given class on both sides of the plane and the decision boundary by fitting the optimal separation hyperplane (OSH). The related research of statistical learning theory [59] shows that SVM has better generalization ability and a lower expected error rate when facing unknown samples.

At present, the vast majority of vulnerability analysis studies are faced with the problem that the scale of training samples is much smaller than that of input data. Based on the above considerations, the approach proposed in [29] integrates various vulnerability-related local data sources in a data-driven way, then trains and predicts the high-dimensional feature vectors extracted from vulnerability disclosure reports based on SVM, which achieves higher accuracy than similar standard heuristics.

From another point of view, as the decision reason of OSH, support vectors can explain the classification algorithm, which is not available in other machine learning algorithms. Consequently, taking advantages of the strong expandability of SVM in dealing with large-scale datasets and the interpretability of the algorithm, the vulnerability classification tool VCCFinder [32] is proposed to implement the large-scale mapping from CVE to GitHub submission for the first time, to create a Vulnerability-Contributing Commits (VCCs) database to train the SVM detection model. The experiments indicate that it reduces false positives by more than 99% and is much better than Flawfinder [60].

Apart from this, there are many other application scenarios of SVM-based vulnerability analysis of source code. Some studies construct SVM classifiers based on different feature definition manners such as CDG [35], word frequency [61], AST substructure [62], or according to the same feature definition to compare the vulnerability classification effects of SVM with other classical machine learning algorithms [53, 54]. To be expected, SVM always embodies prominent class representation performance.

2) Other conventional models: In the modeling stage, most studies prefer to adopt a variety of classic machine learning algorithms for comparison, and evaluate the performance of different vulnerability models by measuring various indicators of the result. Interestingly, we noticed that Decision Trees usually performed well in most comparative experiments.

Chowdhury and Zulkernine [30] define features based on code metrics and compare the performance of Logistic Regression, C4.5 Decision Tree, Random Forests, and Naive Bayes. Among them, C4.5 Decision Trees, Random Forests, and Naive Bayes are adopted for the first time for vulnerability prediction. Furthermore, Shar and Tan [64] use Naive Bayes, C4.5, MLP to predict SQL injection (SQLI) and cross-site scripting (XSS) vulnerabilities based on static code properties, of which C4.5 and MLP are the most prominent. Similarly, in the comparative experiment of Kronjee, Hommersom, and Vranken [27] for these two vulnerabilities, the performance of the Decision Tree classifier is the best.

There are other vulnerability detection methods based on conventional machine learning algorithms that are not discussed in detail [33, 52, 66]. It is worth noting that these classical algorithms embody distinctive performance on different evaluation indicators, however, the results show that there is no extremely significant difference between them [67].

It is undeniable that the limitation of manual predefinition of features in conventional machine learning techniques makes feature engineering the key of these studies. Different from the study of vulnerability detection based on deep learning, we hardly see any great improvement in the architecture of the classifier, nor do mention the reasons for the selection of several algorithms compared in the experiment. When diverse algorithms show differences in performance, we yearn for a reasonable explanation of the internal mechanism of the model.

B. Deep Learning

We summarize the studies on source code vulnerability analysis based on deep learning algorithms in recent years. It can be concluded that CNN and RNN are the most commonly used neural network models in this field.

1) CNN-based model: Some studies establish vulnerability prediction models based on CNN, and capture the structural information of the source code through the connectivity between neurons, while the maximum pooling layer can skillfully reduce the dimensions of the intermediate representation of source code, providing additional robustness for the prediction model.

A novel convolution neural network NP-CNN [47] is proposed, which reflects the program structure through a well-designed convolution operation based on the original CNN network structure, and learns the unified feature representation and location of vulnerabilities according to defect report and source code.

Li et al. [40] propose a defect prediction framework DP-CNN, which can not only capture the semantic and structural features of the program but also adopt the method of word embedding, combined with the conventional manually defined features to realize accurate software defect prediction. Moreover, Harer et al. [51] improve the TextCNN model based on Kim et al. [19] and achieve better results in the experiment.

Apart from this, a series of studies consider the combination of CNN and RNN to build prediction models and compare their performance objectively.

Wu et al. [68] point out the advantages of CNN in time efficiency and feature representation. They extract features from source code function call sequences and build models based on CNN, LSTM respectively firstly, and the CNN-LSTM model is constructed which combined CNN spatial learning characteristics with LSTM sequence learning. The experimental results show that the CNN-LSTM model outperforms the other models in terms of F1-measure.

Russell et al. [51] comprehensively utilize CNN and RNN to automatically generate source code features. Compared with using the model for feature extraction independently, this manner achieves better results on the same datasets.

The essential characteristics of CNN lead to the loss of most of the context information in the feature extraction stage [20]. In contrast, RNN solves the problem that CNN cannot deal with time-related series.

2) RNN-based model: The output of RNN depends on the accumulated information within a given time step. It can capture the useful information from the past for current prediction [69], to extract complex sequential interactions to contribute to prediction decisions. As a variant of the cyclic network, LSTM solves the problems of gradient disappearance, gradient explosion, and large memory consumption of original RNN in the training process to some extent [71]. It is currently one of the most popular and effective models for source code vulnerability analysis.

POSTER [41] acquires the independent characteristics of the project by learning the serialized AST representation. It designs the Bi-directional Long Short-Term Memory (Bi-LSTM) neural network learning function representation and compares it with the Random Forest classifier to prove its effectiveness in vulnerability detection. The underlying reasons for motivating POSTER to use Bi-LSTM modeling are twofold.

First of all, the AST sequence elements based on depth-first traversal retain the structure information of the original tree to some degree, and the element order of vectors reflects the context of source code. This high similarity with sentences in natural language prompts POSTER to take advantage of the Bi-LSTM architecture in tackling this type of data to learn the high-level representation of source code vulnerability information.

Secondly, considering that there are various forms of vulnerabilities related to consecutive or intermittent elements in vulnerable programming patterns, POSTER builds a network based on LSTM and captures long-term dependencies between elements through a stacked LSTM layer to learn more advanced vulnerability abstraction instead of modeling directly with RNN.

Similarly, many studies represent code semantics based on AST and then establish vulnerability detection models combined with Bi-LSTM [42, 43, 57], which proved the effectiveness of Bi-LSTM network structure as one of the most advanced research results in vulnerability discovery.

As the pioneer of vulnerability detection systems based on deep learning, VulDeePecker [44] and SyseVR [50] adopt Bi-LSTM to apply fine-grained program representation to high-level representation of neural network learning and locate vulnerabilities for the first time. Besides, inspired by the multi-feature fusion of image and video in [72], μVulDeePecker [56] learns the global and local features captured by the code gadget through deep Bi-LSTM layers of different sizes, and ultimately connects and adjusts the features with the Bi-LSTM layer through the fusion layer. It is worth noting that μVulDeePecker is the first to apply this fusion notion to the context of vulnerability detection. After that, the follow-up studies of μVulDeePecker such as VulDeeLocator [74] concerned about the limitation and advance the state-of-the-art by realizing high representation capability and high precision. The following table compares the results of the series of studies based on the three indicators of FNR (False Negative Rate), FPR (False Positive Rate) and F1.

TABLE I. COMPARISION OF THE RESULTS OF DIFFERENT VULNERABILITY DETECTION SYSTEMS

Method	FNR(%)	FPR(%)	F1(%)
VulDeePecker	7.9	49.4	65.2
SyseVR	10.1	12.2	86.0
μVulDeePecker	7.49	0.13	94.69
VulDeeLocator	4.0	0.5	97.0

V. FUTURE WORK

In this section, we concentrate on the common problems mentioned in the majority of the studies reviewed in this survey and summarize the feasible research directions in the future.

A. Dealing with the Shortage of Datasets with Ground Truth

Whether it is a conventional machine learning algorithm that relies on manual feature extraction, or a deep neural network with a high degree of automation, the appropriate and complete datasets are of great significance to the effectiveness of the algorithm in the modeling phase, which is urgently needed for vulnerability analysis of source code today.

Compared with application situations such as NLP, digital image processing, and other research fields that have accumulated rich experience and related technologies are becoming increasingly mature, the research foundation of machine learning in source code vulnerability analysis is comparatively weak. It is worth noting that the shortage of datasets with ground truth is one of the main problems mentioned in most studies.

Although there are widely used vulnerability data sources such as NVD [95], SARD [96], facing the amount of training data required for deep learning, it is undoubtedly a laborious and time-consuming task to deal with the appropriate source code format and label it in the data preprocessing phase.

Vuldeepecker [44] adopts a heuristic method to label the source code, automatically labeling the code gadget according to the patch, and then manually correcting error labels. The result indicates that this automatic labeling method is very effective. Another strategy is based on the semi-supervised model for learning. In the case of a low sampling rate, the

approach proposed in [66] compares the vulnerability detection effects of the supervised model and the semi-supervised model. Facts have proved that in the face of limited vulnerability samples, the semi-supervised model is a better choice than the supervised one. Besides, many semi-supervised learning models have not been mentioned, such as GCN [75], DBM [76], and so on. Their strong data inference potential can be also a promising research direction in the context of vulnerability analysis in the future.

Ultimately, with the prosperity and development of research in this field, as well as the construction and accumulation of vulnerability datasets with ground truth in existing studies, we believe that this problem will be alleviated to a certain extent over time.

B. Mitigation of Class Imbalance

Class imbalance is another issue that we should take into consideration. The amount of vulnerability code is far less than that of benign code, which will inevitably lead to the uneven distribution of data categories. This phenomenon will induce the model to treat minority instances as noise and ignore them, consequently degrading the performance of the model, which has been demonstrated in [77].

One solution is that we can alleviate this problem by reducing the amount of benign code in the training data, but this can lead to the loss of plenty of useful information. DP-CNN [40] balances the two by repeatedly copying the vulnerability file several times. Furthermore, DeepBalance, the first deep learning vulnerability detection system for class imbalance, is proposed in [42]. The system adopts an innovative sample synthesis scheme FOS (Fuzzy-Based Oversampling) to create new samples based on the existing data column feature vectors to redistribute the unbalanced data.

There have been types of research on data imbalance [77]. There are cost-sensitive learning and ensemble learning [78] that have been leveraged to tackle the imbalance problem in other situations. In the future, we look forward to further solving the problem of data imbalance in this field by establishing the relationship between these methods and vulnerability analysis.

C. Improvement of Model Interpretability

When the source code is represented in a more abstract and advanced intermediate form, the basis for discriminating the vulnerable/benign code by the machine learning model cannot be intuitively understood by humans. The core question behind this is: What is the reason for the vulnerability prediction results made by the model? Can I trust the prediction model?

The interpretability of the model enables us to carry a further comprehending of the internal mechanism acting on the vulnerability prediction results and code information, therefore guiding the procedure of source code vulnerability analysis and continuously improving the domain adaptability and overall performance of the neural network architecture.

In some of the studies aforementioned [7, 56, 57], the attention mechanism is adopted to capture critical information of source code and emphasize features that are significant to a particular vulnerability. It embodies the function of interpretation to some extent. Nevertheless, the reason for implementing attention mechanism in most studies is to optimize the extracted features for improving the performance of the vulnerability prediction model. Accordingly, it is not an intuitive and specialized design for model interpretation.

Based on the interpretable model, LIME [80] can explain the prediction of any classifier or regressor. The representative instances with explanations are selected by SP-LIME to reflect the global perspective of the model and provide evidence for the trust. On this basis, Anchors [81] adopts a more understandable and less laborious method to predict the behavior of the model on unknown instances, which provides a rather clear coverage for interpretation to achieve higher precision.

These model interpretation tools have demonstrated their practicality and flexibility in a variety of situations such as text classification, structured prediction, visual question answering, and so on. However, they are expected to be further expanded in the domain of vulnerability analysis.

D. Some Novel Research Directions

By summarizing the limitations and shortcomings of machine learning technology in the context of security, and combined with some preliminary innovative researches emerging at present, we put forward some novel research directions in this field in the future.

The transfer learning from sequence source to sequence target is realized by CDAN [82] for the first time. This new architecture not only carries out software vulnerability detection but also provides a basis for a wide range of applications, including financial technology, computational biology, and other fields. SCDAN (Semi-supervised Code Domain Adaptation Network) further realizes the development and utilization of untagged target data based on CDAN and its performance outperforms other benchmarks. This concept of transfer learning provides a more extensive research and data basis for the application of experienced deep learning technology in the field of vulnerability analysis. For enhancing the adaptability of diverse models in different fields and the generalization ability between various types of vulnerabilities, transfer learning can be a promising research direction.

The emerging Graph Neural Network (GNN) [75, 83] has achieved better results than CNN, RNN in non-Euclidean structured data such as topological graphs [84, 85]. Considering the data form of the graph (or tree) structure of most code properties, and combined with some research achievements in the vulnerability discovery based on graph traversal [36-38], attempting to utilize GNN to graph structure data to explore vulnerability code specific patterns will be an innovation in the context.

Furthermore, Coulter et al. [86, 87] present a novel methodology of data-driven cybersecurity (DDCS), which reflects the strong link among data, models, and methodology in the detection of practical application network code. Under the background of big data, this purely data-driven approach provides a new idea for the application of machine learning in the field of vulnerability analysis.

There remain tentative explorations to be elaborated. These novel research directions originate from the analysis of numbers of unsolved problems of source code vulnerability analysis based on machine learning currently, which provides new ideas for future work in this field. Moreover, it also brings more opportunities for the evolution of security technology.

VI. Conclusion

Generally speaking, with the popularity of software applications and the increasing complexity of practical applications in the information era, the game between human beings and software security issues will never end. Nevertheless, machine learning technology remains great room for improvement in the field of source code vulnerability analysis, which is worth looking forward to. In this survey, we review the researches on source code vulnerability analysis utilizing machine learning technology in recent years.

First of all, through an overview of the general vulnerability prediction model based on machine learning, we point out the focus of research in this field and the classification basis of this paper from two aspects of feature extraction and prediction model. Secondly, we divide the research into three categories according to the intermediate representation of the source code in the feature extraction stage. We summarize and reveal the development trend of intermediate representation and the characteristics of different methods. Thirdly, according to whether it is required to define features manually in the feature predefinition stage, we divide machine learning algorithms into two categories: conventional machine learning algorithms and deep learning algorithms. Combined with the essence and principles of different models adopted in these studies, we discuss and compare their performance when applied to vulnerability prediction. Finally, based on the above studies, we draw a summary of the existing problems in this field and the feasible research direction in the future.

References

[1] I. Arghire. (2018). Record-breaking number of vulnerabilities disclosed in 2017: Report. Available: https://www.securityweek.com/record-breaking-number-vulnerabilities-disclosed-2017-report

[2] M. Pereira and S. Christiansen. (2020). Secure the software development lifecycle with machine learning. Available: https://www.microsoft.com/security/blog/2020/04/16/secure-software-development-lifecycle-machine-learning/

[3] A. ASSARAF. This is what your developers are doing 75% of the time, and this is the cost you pay. Available: https://coralogix.com/log-analytics-blog/this-is-what-your-developers-are-doing-75-of-the-time-and-this-is-the-cost-you-pay/

[4] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," IEEE Communications Surveys & Tutorials, vol. 20, no. 2, pp. 1397-1417, 2018.

[5] S. M. Ghaffarian and H. R. Shahriari, "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey," vol. 50, no. 4 %J ACM Comput. Surv., p. Article 56, 2017.

[6] G. Ollmann. (2020). Tackling the SDLC with machine learning. Available: https://www.securityweek.com/tackling-sdlc-machine-learning

[7] S. Liu, M. Dibaei, Y. Tai, C. Chen, J. Zhang, and Y. Xiang, "Cyber vulnerability intelligence for internet of things binary," IEEE Transactions on Industrial Informatics, vol. 16, no. 3, pp. 2154-2163, 2020.

[8] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," presented at the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017.

[9] M. A. J. I. A. Albahar, "A Modified Maximal Divergence Sequential Auto-Encoder And Time Delay Neural Network Models For Vulnerable Binary Codes Detection," vol. PP, no. 99, pp. 1-1, 2020.

[10] G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, and L. Mounier, "Toward large-scale vulnerability discovery using machine learning," presented at the Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, 2016.

[11] E. J. Schwartz, T. Avgerinos, and D. Brumley, "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)," in 2010 IEEE Symposium on Security and Privacy, 2010, pp. 317-331.

[12] S. Kim, S. Woo, H. Lee, and H. Oh, "VUDDY: A scalable approach for vulnerable code clone discovery," presented at the 2017 IEEE Symposium on Security and Privacy (SP), 2017.

[13] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," IEEE Transactions on Software Engineering, vol. 28, no. 7, pp. 654-670, 2002.

[14] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: scaling code clone detection to big-code," presented at the Proceedings of the 38th International Conference on Software Engineering, Austin, Texas, 2016. Available: https://doi.org/10.1145/2884781.2884877

[15] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: finding copy-paste and related bugs in large-scale software code," IEEE Transactions on Software Engineering, vol. 32, no. 3, pp. 176-192, 2006.

[16] J. Walden and M. Doyle, "SAVI: Static-analysis vulnerability indicator," IEEE Security & Privacy, vol. 10, no. 3, pp. 32-39, 2012.

[17] B. Baloglu, "How to find and fix software vulnerabilities with coverity static analysis," in 2016 IEEE Cybersecurity Development (SecDev), 2016, pp. 153-153.

[18] A. Z. Baset and T. Denning, "IDE Plugins for Detecting Input-Validation Vulnerabilities," in 2017 IEEE Security and Privacy Workshops (SPW), 2017, pp. 143-146.

[19] Y. J. a. e.-p. Kim, "Convolutional neural networks for sentence classification," p. arXiv:1408.5882Accessed on: August 01, 2014Available: https://ui.adsabs.harvard.edu/abs/2014arXiv1408.5882K

[20] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," presented at the Proceedings of the AAAI Conference on Artificial Intelligence, 2015. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9513

[21] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang, "Software vulnerability analysis and discovery using deep learning techniques: A survey," IEEE Access, vol. 8, pp. 197158-197172, 2020.

[22] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: A survey," Proceedings of the IEEE, vol. 108, no. 10, pp. 1825-1848, 2020.

[23] G. Lin, W. Xiao, J. Zhang, and Y. Xiang, "Deep Learning-based vulnerable function detection: A benchmark," Cham, 2020, pp. 219-232: Springer International Publishing.

[24] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," ACM Computing Surveys, vol. 51, no. 4, pp. 1-37, 2018.

[25] S. K. Singh and A. Chaturvedi, "Applying deep learning for discovery and analysis of software vulnerabilities: A brief survey," in Soft Computing: Theories and Applications(Advances in Intelligent Systems and Computing, 2020, pp. 649-658.

[26] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Applied Soft Computing, vol. 27, pp. 504-518, 2015.

[27] J. Kronjee, A. Hommersom, and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," presented at the Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018.

[28] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications, vol. 36, no. 4, pp. 7346-7354, 2009/05/01/ 2009.

[29] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: Learning to classify vulnerabilities and predict exploits," presented at the Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, DC, USA, 2010. Available: https://doi.org/10.1145/1835804.1835821

[30] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," Journal of Systems Architecture, vol. 57, no. 3, pp. 294-313, 2011.

[31] Y. Shin and L. Williams, "Can traditional fault prediction models be used for vulnerability prediction?," Empirical Software Engineering, vol. 18, no. 1, pp. 25-59, 2013/02/01 2013.

[32] H. Perl et al., "VCCFinder: Finding potential vulnerabilities in open-source projects to assist code audits," presented at the Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015.

[33] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," IEEE Transactions on Software Engineering, vol. 40, no. 10, pp. 993-1006, 2014.

[34] D. Wijayasekara, M. Manic, and M. McQueen, "Vulnerability identification and classification via text mining bug databases," in IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, 2014, pp. 3612-3618.

[35] V. H. Nguyen and L. M. S. Tran, "Predicting vulnerable software components with dependency graphs," presented at the Proceedings of the 6th International Workshop on Security Measurements and Metrics, Bolzano, Italy, 2010. Available: https://doi.org/10.1145/1853919.1853923

[36] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," presented at the 2014 IEEE Symposium on Security and Privacy, 2014.

[37] F. Yamaguchi, "Pattern-Based Vulnerability Discovery," 2015.

[38] F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, "Automatic inference of search patterns for taint-style vulnerabilities," presented at the 2015 IEEE Symposium on Security and Privacy, 2015.

[39] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," presented at the Proceedings of the 38th International Conference on Software Engineering, 2016.

[40] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," presented at the 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), 2017.

[41] G. Lin, J. Zhang, W. Luo, L. Pan, and Y. Xiang, "POSTER: Vulnerability discovery with function representation learning from unlabeled projects," presented at the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017.

[42] S. Liu, G. Lin, Q.-L. Han, S. Wen, J. Zhang, and Y. Xiang, "DeepBalance: Deep-learning and fuzzy oversampling for vulnerability detection," IEEE Transactions on Fuzzy Systems, pp. 1-1, 2019.

[43] G. Lin et al., "Cross-project transfer representation learning for vulnerable function discovery," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3289-3297, 2018.

[44] Z. Li et al., "VulDeePecker: A deep learning-based system for vulnerability detection," presented at the Proceedings 2018 Network and Distributed System Security Symposium, 2018.

[45] Z. Li, D. Zou, J. Tang, Z. Zhang, M. Sun, and H. Jin, "A comparative study of deep learning-based vulnerability detection system," IEEE Access, vol. 7, pp. 103184-103197, 2019.

[46] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," presented at the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017.

[47] X. Huo, M. Li, and Z.-H. Zhou, "Learning unified features from natural and programming languages for locating buggy source code," presented at the Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, New York, USA, 2016.

[48] H. Khanh Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. J. a. e.-p. Ghose, "Automatic feature learning for vulnerability prediction," p. arXiv:1708.02368Accessed on: August 01, 2017Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170802368K

[49] M.-j. Choi, S. Jeong, H. Oh, and J. J. a. e.-p. Choo, "End-to-end prediction of buffer overruns from raw source code via neural memory networks," p. arXiv:1703.02458Accessed on: March 01, 2017Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170302458C

[50] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. J. a. e.-p. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," p. arXiv:1807.06756Accessed on: July 01, 2018Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180706756L

[51] R. Russell et al., "Automated vulnerability detection in source code using deep representation learning," in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, pp. 757-762.

[52] M. Davari, M. Zulkernine, and F. Jaafar, "An automatic software vulnerability classification framework," presented at the 2017 International Conference on Software Security and Assurance (ICSSA), 2017.

[53] B. Chernis and R. Verma, Machine learning methods for software vulnerability detection. 2018, pp. 31-39.

[54] Y. Pang, X. Xue, and A. S. Namin, "Predicting vulnerable software components through n-gram analysis and statistical feature selection," presented at the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015.

[55] Y. Pang, X. Xue, and H. Wang, "Predicting vulnerable software components through deep neural network," presented at the Proceedings of the 2017 International Conference on Deep Learning Technologies - ICDLT '17, 2017.

[56] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "μVulDeePecker: A deep learning-based system for multiclass vulnerability detection," IEEE Transactions on Dependable and Secure Computing, pp. 1-1, 2019.

[57] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," Scientific Programming, vol. 2019, pp. 1-14, 2019.

[58] R. Malhotra, A. Kaur, and Y. Singh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines," International Journal of System Assurance Engineering and Management, vol. 1, no. 3, pp. 269-281, 2011.

[59] V. Vapnik, The nature of statistical learning theory. Springer science & business media, 2013.

[60] O. Ferschke, Iryna Gurevych, and M. Rittberger, "FlawFinder: A modular system for predicting quality flaws in wikipedia," presented at the CLEF (Online Working Notes/Labs/Workshop), 2012.

[61] A. Hovsepyan, R. Scandariato, W. Joosen, and J. Walden, "Software vulnerability prediction using text analysis techniques," presented at the Proceedings of the 4th international workshop on Security measurements and metrics, Lund, Sweden, 2012. Available: https://doi.org/10.1145/2372225.2372230

[62] M. Qingkun, W. Shameng, Z. Bin, and T. Chaojing, "Automatically discover vulnerability through similar functions," in 2016 Progress in Electromagnetic Research Symposium (PIERS), 2016, pp. 3657-3661.

[63] E. Penttilä, "Improving C++ software quality with static code analysis," 2014.

[64] L. K. Shar and H. B. K. Tan, "Predicting common web application vulnerabilities from input validation and sanitization code patterns," in 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 310-313.

[65] K. Zhang, "A machine learning based approach to identify SQL injection vulnerabilities," presented at the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019.

[66] L. K. Shar, L. C. Briand, and H. B. K. Tan, "Web application vulnerability prediction using hybrid program analysis and machine learning," IEEE Transactions on Dependable and Secure Computing, vol. 12, no. 6, pp. 688-707, 2015.

[67] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings," IEEE Transactions on Software Engineering, vol. 34, no. 4, pp. 485-496, 2008.

[68] F. Wu, J. Wang, J. Liu, and W. Wang, "Vulnerability detection with deep learning," in 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 1298-1302.

[69] J. L. Elman, "Finding Structure in Time," Cognitive Science, https://doi.org/10.1207/s15516709cog1402_1 vol. 14, no. 2, pp. 179-211, 1990/03/01 1990.

[70] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166, 1994.

[71] R. Pascanu, T. Mikolov, and Y. J. a. e.-p. Bengio, "On the difficulty of training Recurrent Neural Networks," p. arXiv:1211.5063Accessed on: November 01, 2012Available: https://ui.adsabs.harvard.edu/abs/2012arXiv1211.5063P

[72] X. Ou, H. Ling, H. Yu, P. Li, F. Zou, and S. Liu, "Adult image and video recognition by a deep multicontext network and fine-to-coarse strategy," vol. 8, no. 5 %J ACM Trans. Intell. Syst. Technol., p. Article 68, 2017.

[73] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, "VulPecker: An automated vulnerability detection system based on code similarity analysis," presented at the Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016.

[74] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu, and H. J. a. e.-p. Jin, "VulDeeLocator: A deep learning-based fine-grained vulnerability detector," p. arXiv:2001.02350Accessed on: January 01, 2020.Available: https://ui.adsabs.harvard.edu/abs/2020arXiv200102350L

[75] T. N. Kipf and M. J. a. e.-p. Welling, "Semi-supervised classification with Graph Convolutional Networks," p. arXiv:1609.02907Accessed on: September 01, 2016.Available:

https://ui.adsabs.harvard.edu/abs/2016arXiv160902907K

[76] R. Salakhutdinov and G. Hinton, "A better way to pretrain Deep Boltzmann Machines," in International Conference on Neural Information Processing Systems, 2012.

[77] P. Morrison, K. Herzig, B. Murphy, and L. Williams, "Challenges with applying vulnerability prediction models," presented at the Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, Urbana, Illinois, 2015. Available:

https://doi.org/10.1145/2746194.2746198

[78] S. Liu, Y. Wang, J. Zhang, C. Chen, and Y. Xiang, "Addressing the class imbalance problem in Twitter spam detection using ensemble learning," Computers & Security, vol. 69, pp. 35-49, 2017/08/01/ 2017.

[79] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015/05/01 2015.

[80] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why should I trust you?"," presented at the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[81] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018, vol. 32, no. 1.

[82] V. Nguyen et al., "Deep domain adaptation for vulnerable code function identification," in 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-8.

[83] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on Graph Neural Networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4-24, 2021.

[84] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," presented at the Proceedings of The 33rd International Conference on Machine Learning, Proceedings of Machine Learning Research, 2016. Available: http://proceedings.mlr.press

[85] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," IEEE Signal Processing Magazine, vol. 34, no. 4, pp. 18-42, 2017.

[86] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Data-driven cyber security in perspective—Intelligent traffic analysis," IEEE Transactions on Cybernetics, vol. 50, no. 7, pp. 3081-3093, 2020.

[87] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Code analysis for intelligent cyber systems: A data-driven approach," Information Sciences, vol. 524, pp. 46-58, 2020.

[88] M. Alvares, T. Marwala, and F. B. d. L. Neto, "Applications of computational intelligence for static software checking against memory corruption vulnerabilities," in 2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), 2013, pp. 59-66.

[89] X. Li et al., "A mining approach to obtain the software vulnerability characteristics," presented at the 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), 2017.

[90] Q. Meng, B. Zhang, C. Feng, and C. Tang, "Detecting buffer boundary violations based on SVM," presented at the 2016 3rd International Conference on Information Science and Control Engineering (ICISCE), 2016.

[91] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Automatic classification for vulnerability based on machine learning," in 2013 IEEE International Conference on Information and Automation (ICIA), 2013, pp. 312-318.

[92] 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, 2019.

[93] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao, "Automatic classification method for software vulnerability based on deep neural network," IEEE Access, vol. 7, pp. 28291-28298, 2019.

[94] J. A. Harer et al., "Automated software vulnerability detection with machine learning," p. arXiv:1803.04497Accessed on: February 01, 2018.Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180304497

[95] NVD, https://nvd.nist.gov/

[96] Software Assurance Reference Dataset, https://samate.nist.gov/SRD/index.php.