Systematic Analysis of Deep Learning Model for Vulnerable Code Detection

Mohammad Taneem Bin Nazim*, Md Jobair Hossain Faruk†, Hossain Shahriar‡, Md Abdullah Khan*
Mohammad Masum§, Nazmus Sakib‡, Fan Wu¶

*Department Computer Science, Kennesaw State University, USA
†Department of Software Engineering and Game Development, Kennesaw State University, USA
‡Department of Information Technology, Kennesaw State University, USA
§School of Data Science, Kennesaw State University, USA
¶Department of Computer Science, Tuskegee University, USA

{*mnazim, †mhossa21}@students.kennesaw.edu, {‡hshahria, *mkhan74, §mmasum, ‡nsakib1}@kennesaw.edu
{¶fwu}@tuskegee.edu

***Abstract*—Software vulnerabilities have become a serious problem with the emergence of new applications that contain potentially vulnerable or malicious code that can compromise the system. The growing volume and complexity of software source codes have opened a need for vulnerability detection methods to successfully predict malicious codes before being the prey of cyberattacks. As leveraging humans to check sources codes requires extensive time and resources and preexisting static code analyzers are unable to properly detect vulnerable codes. Thus, artificial intelligence techniques, mainly deep learning models, have gained traction to detect source code vulnerability. A systematic review is carried out to explore and understand the various deep learning methods employed for the task and their efficacy as a prediction model. Additionally, a summary of each process and its characteristics are examined and its implementation on specific data sets and their evaluation will be discussed.**

***Index Terms*—Deep Learning, Software Security, Source code Vulnerability**

I. INTRODUCTION

The growing number of software applications in the modern world has also seen a growing number of cyber security attacks have plagued the information era. The Mitre organization in charge of the Common Vulnerability and Exposures (CVE) term software vulnerability 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" [37].

The number of software vulnerabilities have only increased over time where CVE reports 20149 vulnerabilities reported in 2021 while 4155 vulnerabilities detected in 2011 as shown extensively in Figure 1. Recently, Apple suffered an iMessage vulnerability, FORCEDENTRY exploit, by Pegasus spyware that bypassed the iOS BlastDoor security feature to deploy the spyware and create malicious webpages when iPhone or iPads access the internet [39]. This attack implemented integer overflow as tracked by CVE [37] and resulted in snooping of private information of political figures and extortion scams that blackmailed people to pay a ransom [39].

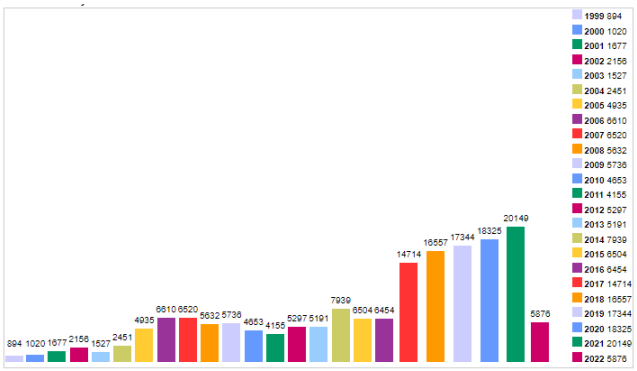

Fig. 1. CVE Vulnerabilities detected from 1999 to 2022 [42]

The prevalence of software vulnerabilities still exist in spite of academics and industries efforts to enhance software quality. The main two reasons for this phenomenon is widespread usage of open-software and code reuse that contain these flaws [40] and the increase in the number of internet user which also increased the number of attacks [38]. Thus, to prevent cyberattacks that cause data leakage of sensitive information, denial-of service condition, lose control of software or even suspend production of firms [41] deep learning approaches can be implemented to detect vulnerabilities existing in the source code of the software.

Deep learning is a subset of machine learning in Artificial Intelligence that aims to imitate the human brain and nervous system. Deep learning techniques tend to be superior than traditional rule-based methods generated manually by human experts, which are can be imprecise, resource-intensive, time-consuming, subjective to the software and entirely dependent on the knowledge and experience of the expert, or by static tools such as Checkmarx, Flawfinder etc. that have high false-positive rates and/or high false-negative rates [40]. Deep learning techniques can learn features automatically to demonstrate

978-1-6654-8810-5/22/$31.00 ©2022 IEEE
DOI 10.1109/COMPSAC54236.2022.00281

better generalization ability than manually determined features. So, success and efficacy of the vulnerability prediction model is mediated by the sophistication of the deep neural networks of the model in deep learning rather than the feature selection used in conventional machine learning.

A. Contributions of this Survey

Few surveys exist that explicitly conduct the investigation of deep learning models in software vulnerability detection. This survey aims to expand on this research field and provide a different perspective. An overview of the nature of datasets that are used to train the deep learning models such as: synthetic codes, semi-synthetic codes and real codes. An in-depth perspective on the popular source code representations to capture semantic information of unlabelled source code are explored. Furthermore, deep learning models in vulnerability detection are presented with its evaluation metrics.

II. Research Methodology

A systematic literature review has been conducted to find existing papers related to our topic of research. Thus, a "Search Process" was implemented to find such papers for our study [43], [44]. The specific search strings had the keywords, "Source Code Vulnerability Detection", "Software Vulnerability Detection using Deep Learning", "Automatic Detection OR Discovery of vulnerability", "Deep Learning detect Software Vulnerability". An exclusion process was utilized to exclude papers unrelated to our goal or duplicates of the selected paper. The scientific databases considered were: (i) IEEE Xplore (ii) arXiv e-Print Archive (iii) ACM Digital ibrary and (iv) Google Scholar.

TABLE I
Inclusion and exclusion criteria for the primary studies

Condition (Inclusion)	Condition (Exclusion)
The study must be related to Software Vulnerability that implement Deep Learning methods specifically	The studies not focusing on software vulnerability or implements other models such as machine learning trained models
Papers are not duplicated in different databases	Similar papers in different databases
Peer-reviewed papers published in a conference proceeding or journal	Non-peer-reviewed papers
Studies that are available in the full format and in English	Studies are not available fully and Non-English studies

The initial keyword search process had undergone a filtration procedure that only selected papers published in the last 5 years from 2017 to 2022. Additional restraints were placed in each of the scientific databases to find relevant research material. IEEE Xplore included only Conferences and Journals while ACM required filters specifying Journals and Research Articles. arXiv e-Print Archive did not require any predefined filters nor did Google Scholar. However, Google Scholar failed to provide any unique research papers related to our study. A total of 303 research papers were procured in-depth screening process that accounted for the publication title, abstract, experimental results and conclusions shortened the list to 10 papers for our study.

TABLE II
Generalized Table for Search Criteria

Scientific Databases	Initial Keyword Search	Total Inclusion
IEEE Xplore	161	5
arXiv e-Print Archive	42	3
ACM Digital ibrary	50	1
Springer	25	1
Google Scholar	50	0

A. Related Reviews

Multiple studies have been conducted to systematically summarize the machine learning technology, deep learning technology or state-of-the-art research achievements in source code vulnerability analysis. Ghaffarian and Shahriari [51] provided an extensive and detailed review of the traditional machine learning and data mining techniques for detection and analysis of software vulnerability. This research concluded the immature state of machine learning techniques but did not explore deep learning techniques. Malhotra [52] came to a similar conclusion after reviewing 64 preliminary studies of machine learning techniques used on software fault prediction.

Radjenovic *et al.* [45] conducted a systematic literature review on the software metrics for software fault solutions. However, software fault prediction has very limited application and lacks relevance to software vulnerability detection [46]. M. Masum *et al.* [47] introduced a novel Bayesian optimization-based framework for the automatic optimization of hyper-parameters to ensure the best deep neural network architecture. The authors also introduced a feature selection-based framework by adopting different machine learning algorithms including neural network-based architectures to classify the security level for ransomware detection and prevention by applying Decision Tree (DT), Random Forest (RF), Naive Bayes (NB), Logistic Regression (LR) as well as Neural Network (NN)-based classifiers [48]. M.J.H. Faruk *et al.* emphasized Artificial Intelligence (AI) based techniques for detecting and preventing malware activity. Both machine learning and deep learning methods, techniques, and approaches were presented to detect and prevent malware [49].

III. Source Code Representation

Source code representation is an essential step to decompose the input sample of source code to only contain important syntactic and semantic information by removing unnecessary lines, comments, spaces. While many representations exist, this study will aim to present the state-of-the-art methods utilized to capture the structural and semantic information from the source code for feature extraction.

A. Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) is the tree representation of a source code that can capture the abstract syntax structure and semantics of the code block that allows the source code to be

analyzed statically [17, 18]. This procedure allows the partition of the initial input source code into smaller parts and achieves greater granularity of the function-level source code [13]. An AST can be acquired by using a parser such as CodeSensor [18], [19], or Pycparser [7]. While these ASTs can be directly used, it lacks granularity due to codes being large or complex [16]. Thus, in [7] the AST tree, which can be considered an m-ary tree, enforces rules to convert it into a complete binary AST tree to preserve its structural relations from the AST nodes. Similarly, an RNN model has also been proposed called Tree-LSTM which leverages its bottom-up calculation to integrate outputs of all AST child nodes to construct a binary AST tree [18], [20].

B. Code Gadgets

Code gadgets are a composition of multiple lines of program statements that have a semantic correlation in terms of data dependency and control dependency [3] by implementing program slicing [22]. Program slicing can be categorized into two type of slices: forward and backward [3]. Forward slices are the slices of code that are received from an external input, such as file o sockets; whereas, backward slices do not receive an direct input externally from the environment in which the program is run [2], [21]. This decomposition of programs by analyzing their data flow and control flow [22] allows the reduction of the lines of codes and focuses on the key points of library/API function calls, arrays, or pointers. VulDeePecker [3] only accounts for data dependency in the code gadgets [8], [21] as it uses the commercial tool Checkmarx [25] and performs forward and backward slices for each argument and then assembled to form code gadgets. In Zagane's model [23] the dataset contains 420,627 code slices which consist of 56395 vulnerable code slices and the code metrics of each slice is calculated for their deep learning model.

C. Code Property Graph (CPG)

Code Property Graph (CPG) is an amalgamation of classical data structures representing a source i.e. abstract syntax tree, control flow graph (CFG), and program dependence graph (PDG) [26]. The REVEAL [1] vulnerability prediction framework utilizes a modified CPG rather than the data structure represented by Yamaguchi *et al.* [26] by adding a data-flow graph (DFG) in conjunction with the existing CPG to capture additional context about the semantic in the code. Devign [5] takes a similar approach of CPG in source code representation by merging the concepts of AST, CFG, DFG, and Natural Code Sequence (NCS) into a joint graph.

D. Lexed Representation

Russell *et al.* [6] constructed a custom lexer for C/C++ code that formed a code representation of 156 tokens as the vocabulary size. This methodology included keywords, operators, separators while excluding irrelevant code in terms of compilation. The lexer converted the code to tokens of three different types. String, character, and float literals were lexed to type-specific placeholders while integer literals were tokenized digit-by-digit due to relevance to vulnerability detection. Types and function calls from common libraries are mapped to their generic versions. Zheng *et al.* [19] implemented this custom lexer for word-level tokenization on texts from the Draper VDISC dataset.

E. Semantics-based Vulnerability Candidates (SeVC)

Semantics-based Vulnerability Candidates are the various statements that are semantically related to the Syntax-Based Vulnerability Candidates (SyVC) by extraction of its program slices [19]. In order to conceptualize SeVCs the term SyVC needs to be explored. SyVCs extraction requires the conversion of source code to an AST which contains multiple consecutive tokens. The AST is traversed to locate a code element that matches the defined vulnerability syntax which is labelled as a SyVC [4]. SyVCs are transformed into SeVCs by program slicing [22] to capture the semantic relation of statements based on data dependency and control dependency. Joern [24] tool extracts PDGs of each SyVC; then, program slices are generated from interprocedural forward and backward slices [27] which are transformed to SeVCs [4].

IV. Deep Learning Models

Deep Neural Networks were inspired from the biological aspects of the human brain and nervous system. It forms networks similar to the human nervous system and mimics the thought mechanisms of humans by training itself with the data provided. Deep learning has been successful in image classification and captures nonlinear effects of variables with high-level feature representation. Thus, deep learning techniques have been implemented to extract features from texts (source code) and then train its model to understand and detect the vulnerability in the software.

A. CNN (Convolutional Neural Network)

CNN is a deep learning model that was introduced mainly to analyze features in images but it has also been implemented on feature extraction of source codes to learn its vector representations [12]. Furthermore, since codes do not possess extensive features that are present in images, CNN can extract the features as well as understand the structure of the program by learning patterns and relationships between contexts in the source code [9]. In the study of Russell *et al.* [6] the CNN implemented had a filter size of 9 and 512 filters paired with batch normalization and ReLU to determine the sequential tokens.

B. RNN (Recurrent Neural Network)

RNN allows longer token dependencies to be extracted than CNN [6] as its "memory" contains information from the previous and next tokens [9]. RNN used in Russel *et al.* [6] had a hidden size of 256 and was maxpooled to generate a fixed size of vector representations.

LSTM is an extended version of the RNN architecture that can learn long-term dependencies. LSTM can overcome the limitations of a conventional statistical model (such as

ARIMA) by capturing non-linearity of sequential data and simultaneously generating more precise forecasting for time-series data [11]. LSTM addresses the vanishing gradient problem of RNN [50]. The building block of LSTM architecture is a memory block, which consists of a memory cell that can preserve information of the preceding time step with self-recurrent connections. BLSTM is a variation of LSTM deep learning model but builds upon the one-way model by pairing it with another CNN model called GRU to enforce two-way LSTM i.e. BLSTM. μVulDeePecker [8] implemented a multi-feature fusion method of BLSTM that could extract information from global-feature learning model, local-feature learning model and feature-fusion model.

GRU is an alternative version of LSTM that was introduced to avoid the vanishing gradient problem and boost the efficiency of LSTM [17]. GRU has a less complicated architecture than LSTM, with a reduced number of parameters to learn. It consists of two gates: update and reset gates, (In comparison, three gates are included in LSTM.) GRU solves the vanishing gradient problems of RNN by leveraging the two gates through controlling what information should be passed to the future states [14]. The input and forget gates in LSTM models are integrated into the GRU update gate, which determines how much information from previous steps should be passed to the future time steps. Similarly, to the output gate in LSTM, the reset gate in the GRU incorporates new input and previous memory and determines how much memory of past information should be forgotten.

V. Datasets

The datasets observed in this field of research when implementing Deep Learning models are source code in the C/C++ language. These datasets are popular because an in-depth review has been done in quantifying the amount of vulnerable and malicious code present in the dataset as shown in Figure 2.

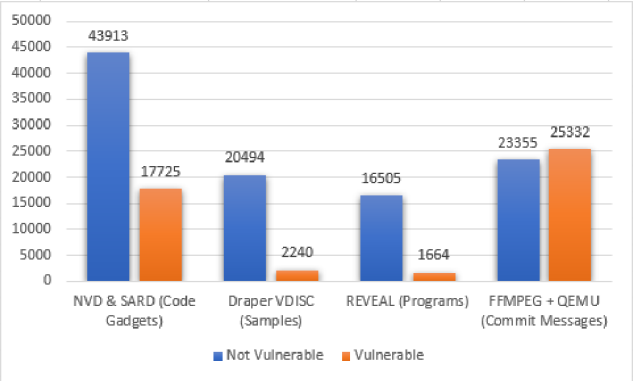

Fig. 2. Overview of Datasets

A. NVD and SARD

Software Assurance Reference Dataset (SARD) [28] contains production, synthetic, and academic security flaws or vulnerabilities, and National Vulnerability Database (NVD) [29] contains vulnerabilities in production software [3]. The dataset is made up of C/C++ programs and software products [4] which contain 61,638 code gadgets, including 17,725 code gadgets that are vulnerable and 43,913 code gadgets that are not vulnerable [31] as shown in Figure 5. Among the 17,725 code gadgets that are vulnerable, 10,440 code gadgets correspond to buffer error vulnerabilities (CWE-119) and the rest 7,285 code gadgets correspond to resource management error vulnerabilities (CWE-399) [3]. The dataset is named as Code Gadget Database (CGD) [31] and its extensions are Semantics-based Vulnerability Candidate (SeVC) dataset [32] (used during SeSeVR [4]) and Multiclass Vulnerability Dataset (MVD) [33] (used during μVulDeePecker [8]).

B. Draper VDISC

The dataset consists of the source code of 1.27 million functions mined from open source software, labelled by static analysis for potential vulnerabilities [30]. Draper VDISC dataset has C/C++ codes from open source projects: Debian Linux Distribution [34], Public git repositories on GitHub [35] and, SATE IV Juliet Test Suite of NIST Samate project [36] where the first 2 source projects are real and the last one is synthetic. The Debian package releases provide a selection of very well-managed and curated code while, the GitHub dataset provides a larger quantity and wider variety of (often lower-quality) code and the SATE IV Juliet Test Suite contains synthetic code examples with vulnerabilities from 118 different Common Weakness Enumeration (CWE) [6], [7].

C. REVEAL

ReVeal is a real-world source code dataset where vulnerabilities are tracked from Linux Debian Kernel and Chromium open-source projects. This dataset contains C/C++ sources and are well-maintained public projects with large evolutionary history and both represent important program domains (OS and browsers) and plenty of publicly available vulnerability reports.[1] Fixed issues with publicly available patches can be collected using Bugzilla for Chromium and Debian security tracker for Linux Debian Kernel. The ReVeal dataset contains a total of 22,734 samples, with 2240 non-vulnerable and 20,494 vulnerable samples as seen in Figure 3.

D. FFMPeg+Qemu

FFMPeg+Qemu is a balanced, real world dataset collected from Github repositories that consist of 4 large C-language open-source projects that are popular among developers and diversified in functionality, i.e., Linux Kernel, QEMU, Wireshark, and FFmpeg [5]. The labelling of this dataset was done manually based on commit messages and domain experts [15]. Figure 5 presents the balanced nature of the dataset with 23355 non-vulnerable commits messages and 25332 which are vulnerbale.

TABLE III
DEEP LEARNING MODELS FOR VULNERABILITY ANALYSIS

Author	Dataset	Dataset Type	Feature Representation	Source Code Representation	Vector Representation	DL Models
Russel *et al.* [6]	DRAPER VDISC	Synthetic, Semi-synthetic, Real	Token	NLP approach (Convolutional and Recurrent Feature Extraction)	word2vec	CNN + RF, RNN + RF
S. Chakraborty *et al.* [1]	REVEAL, FFMPeg+Qemu	Real	Graph	CPG	word2vec	GGNN + MLP + Triplet Loss
G.Tang *et al.* [2]	SARD & NVD	Synthetic, Semi-synthetic	Token	Code Gadgets	doc2vec	KELM
Z. Bligin *et al.* [7]	DRAPER VDISC	Synthetic, Semi-synthetic, Real	Token	Binary AST	Array Representation	MLP, CNN
Z.Li *et al.* [3]	SARD & NVD	Synthetic, Semi-synthetic	Token	Code Gadgets	word2vec	BLSTM
D.Zou *et al.* [8]	SARD & NVD	Synthetic, Semi-synthetic	Token	Code Gadgets + Code Attention	word2vec	BLSTM
Z.Li *et al.* [4]	SARD & NVD	Synthetic, Semi-synthetic	Token	SeVCs	word2vec	BGRU
Y.Zhou *et al.* [5]	FFMPeg+Qemu	Real	Graph	AST+CFG+DFG+NCS	word2vec	GGNN
S. Liu *et al.* [18]	FFMPeg+Qemu	Real	Graph	AST	word2vec	BLSTM
Guo *et al.* [18]	SARD & NVD	Synthetic, Semi-synthetic	Token	Code Gadgets	word2vec	CNN+LSTM

VI. CHALLENGES AND FUTURE WORK

The review acknowledges the infancy of the research field of deep learning-based software vulnerability detection. There are multiple problems that are unresolved but this indicates the need for further research to solve these issues for better discovery of vulnerable codes. Thus, we have compiled a number of challenges and possible future research directions which have been assessed by the conclusive remarks of previous works.

Dataset is an integral part of training a vulnerability prediction model. As surveyed in this paper, various studies utilize various datasets, like SARD & NVD or REVEAL dataset, to train their models which indicates the lack of a standardized benchmarking dataset that covers most CWE vulnerabilities. Thus, a unified and standardized metric for evaluations of the deep learning-based models cannot be produced. Furthermore, deep learning models require huge amounts of training data to provide excellent performance but the current datasets are insufficient in this regard. Thus, development of such vulnerability datasets is an essential direction for research to resolve the issue of dataset with ground truth. The ratio of vulnerable code to non-vulnerable code tends to be staggeringly unbalanced as seen in Figure 5. Non-vulnerable codes are abundant but vulnerable codes are a huge minority that results in deep learning vulnerability prediction models to have insufficient training data for detection of the vulnerabilities and cause overfitting in the model. There exists class imbalance in specific CWE vulnerabilities as well such as CWE 469 present in the Draper VDISC dataset leading to poor prediction where as CWE 119 exists in multitude leading to best prediction of this vulnerability in the model during cross validation [7]

Deep learning models possess nonlinearity and hidden layers that make it difficult to interpret the behaviour that lead to a vulnerability prediction begs the following questions: Is the model accurate? Is the the prediction on vulnerability discovery reliable? What is the reason for the classification of vulnerable or non-vulnerable in a specific piece of the source code? This problem has been tackled by two methods by researchers. Firstly, the use of LIME [10] to create linear models of the neural networks for simple interpretation. Secondly, the introduction of code attention to the source code representation [8] that enables researchers to obtain the attention vectors and quantitatively measure how much attention was given to a specific network in the midst of all the neural networks in the deep learning models. Further research can lead to better source code analysis which can explain deep learning-based models in vulnerability prediction.

VII. CONCLUSION

The emergence of vast software applications with increasing complexity and this continual popularity of software in the information era will require automatic deep learning-based models to learn and detect vulnerabilities in these softwares. In this survey, we review the studies conducted on the implementation of deep learning technology for source code vulnerability analysis. At first, a general overview is discussed in the formation of deep learning models for source code vulnerability detection. A detailed summary of the state-of-the-art source code representations are explored which can decompose the text of source code and still conserve its behaviour to feed into the deep learning models for training. Next, the deep learning models implemented for vulnerability detection are outlined with brief examples of of the hidden layers used in the trained models. Lastly, based on the existing studies mentioned in this paper, the challenges and directions for future work in this research field are stated.

ACKNOWLEDGMENT

The work is partially supported by the U.S. National Science Foundation (NSF) Awards 2100134, 2100115, 1723578, 1723586, and SunTrust Fellowship Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF amd SunTrust.

REFERENCES

[1] S. Chakraborty, R. Krishna, Y. Ding and B. Ray, "Deep Learning based Vulnerability Detection: Are We There Yet," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2021.3087402.

[2] T. Gaigai, Y. Lin, R. Shuangyin, M. Lianxiao, Y. Feng, W. Huiqiang. (2021). An Automatic Source Code Vulnerability Detection Approach Based on KELM. Security and Communication Networks. 2021. 1-12. 10.1155/2021/5566423.

[3] 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.

[4] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, S. Wang, and J. Wang, "Sysevr: A framework for using deep learning to detect software vulnerabilities," arXiv preprint arXiv:1807.06756, 2018.

[5] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in Advances in Neural Information Processing Systems, 2019, pp. 10 197-10 207

[6] 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 Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA 2018). IEEE, 2018, pp. 757- 762.

[7] Z. Bilgin, M. A. Ersoy, E. U. Soykan, E. Tomur, P. Comak and L. Karacay, "Vulnerability Prediction From Source Code Using Machine Learning," in IEEE Access, vol. 8, pp. 150672-150684, 2020, doi: 10.1109/ACCESS.2020.3016774.

[8] D. Zou, S. Wang, S. Xu, Z. Li and H. Jin, "VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection," in IEEE Transactions on Dependable and Secure Computing, vol. 18, no. 5, pp. 2224-2236, 1 Sept.-Oct. 2021, doi: 10.1109/TDSC.2019.2942930.

[9] Wu, Jiajie. "Literature review on vulnerability detection using NLP technology." arXiv preprint arXiv:2104.11230 (2021).

[10] X. Rong, "word2vec parameter learning explained", arXiv preprint arXiv:1411.2738, 2014.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique", Journal of artificial intelligence research, vol. 16, pp. 321-357, 2002

[12] J. Wang, M. Huang, Y. Nie and J. Li, "Static Analysis of Source Code Vulnerability Using Machine Learning Techniques: A Survey," 2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD), 2021, pp. 76-86, doi: 10.1109/ICAIBD51990.2021.9459075.

[13] "Common weakness enumeration," CWE. [Online]. Available: https://cwe.mitre.org/top25/archive/2021/2021cwetop25.html. [Accessed: 23-Feb-2022].

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

[15] Zhuang, Yufan, *et al.* "Software Vulnerability Detection via Deep Learning over Disaggregated Code Graph Representation." arXiv preprint arXiv:2109.03341 (2021).

[16] 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.

[17] Chimmula, Vinay Kumar Reddy, and Lei Zhang. "Time series forecasting of COVID-19 transmission in Canada using LSTM networks." Chaos, Solitons Fractals 135 (2020): 109864.

[18] S. Liu, G. Lin, Q. L. Han, S. Wen, J. Zhang and Y. Xiang, "DeepBalance: Deep-Learning and Fuzzy Oversampling for Vulnerability Detection," in IEEE Transactions on Fuzzy Systems, vol. 28, no. 7, pp. 1329-1343, July 2020, doi: 10.1109/TFUZZ.2019.2958558.

[19] W. Zheng, A. O. A. Semasaba, X. Wu, S. A. Agyemang, T. Liu and Y. Ge, "Representation vs. Model: What Matters Most for Source Code Vulnerability Detection," 2021 IEEE SANER, 2021, pp. 647-653, doi: 10.1109/SANER50967.2021.00082.

[20] Zeroual A, Harrou F, Dairi A, Sun Y. Deep learning methods for forecasting covid19 time-series data: a comparative study. Chaos, Solit Fractals 2020;140:110121.

[21] Z. Li, D. Zou, J. Tang, Z. Zhang, M. Sun and H. Jin, "A Comparative Study of Deep Learning-Based Vulnerability Detection System," in IEEE Access, vol. 7, 2019, doi: 10.1109/ACCESS.2019.2930578.

[22] M. Weiser, "Program Slicing," in IEEE Transactions on Software Engineering, vol. SE-10, no. 4, pp. 352-357, July 1984, doi: 10.1109/TSE.1984.5010248.

[23] M. Masum, M.A. Masud, M. I. Adnan, H. Shahriar, S. Kim, "Comparative study of a mathematical epidemic model, statistical modeling, and deep learning for COVID-19 forecasting and management, Socio-Economic Planning Sciences", Volume 80, 2022, 101249, ISSN 0038-0121, https://doi.org/10.1016/j.seps.2022.101249.

[24] "Joern," May 11, 2019, Accessed: May 4, 2022. [Online]. Available: https://github.com/ ShiftLeftSecurity/joern

[25] Checkmarx, Accessed: May 4, 2022. https://www.checkmarx.com/.

[26] F. Yamaguchi, N. Golde, D. Arp and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs," 2014 IEEE Symposium on Security and Privacy, 2014, pp. 590-604, doi: 10.1109/SP.2014.44.

[28] NVD, National Vulnerability Database, 2022. Accessed: May 4, 2022. [Online] https://nvd.nist.gov/.

[27] F. Tip, "A survey of program slicing techniques," J. Prog. Lang., vol. 3, no. 3, 1995.

[29] NIST Software Assurance Reference Dataset Project, 2022. Accessed: May 4, 2022. [Online] https://samate.nist.gov/SRD/index.php.

[30] Louis Kim and Rebecca Russell, Draper VDISC Dataset - Vulnerability Detection in Source Code, 2021. Accessed: May 4, 2022. [Online] https://osf.io/d45bw/

[31] NDSS, Database of "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection", 2018. Accessed: May 4, 2022. [Online] https://github.com/CGCL-codes/VulDeePecker

[32] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen. SySeVR: "A Framework for Using Deep Learning to Detect Software Vulnerabilities". IEEE TDSC. 2021. doi: 10.1109/TDSC.2021.3051525.

[33] Multiclass Vulnerability Dataset (MVD), 2019. Accessed: May 4, 2022. [Online] https://github.com/muVulDeePecker/muVulDeePecker

[34] Debian-The Universal Operating System. Accessed: May 4, 2022. [Online]. Available: https://www.debian.org/

[35] GithubâDistributed Version Control Software. Accessed: May 4, 2022. [Online]. Available: https://github.com/

[36] P. E. Black and P. E. Black, Juliet 1.3 Test Suite: Changes From 1.2. Gaithersburg, MD, USA: US Department of Commerce, National Institute of Standards and Technology, 2018.

[37] "CVE website," Accessed: May 4, 2022. [Online]. Available: https://cve.mitre.org.

[38] Anjum N, Latif Z, Lee C, Shoukat IA, Iqbal U. MIND: A Multi-Source Data Fusion Scheme for Intrusion Detection in Networks. Sensors. 2021; 21(14):4941. https://doi.org/10.3390/s21144941

[39] https://cybersecurityworks.com/blog/cyber-risk/pegasus-spyware-snoops-on-political-figures-worldwide.html

[40] A. Ramadan, A. Bahaa, O. Ghoneim. A systematic review of the literature on software vulnerabilities detection using machine learning methods. Information Bulletin in Computers and Information , 4, 1, 2022, 1-9. doi: 10.21608/fcihib.2022.87660.1058

[41] Y. Kageyama (2022). Toyota's Japan Production Halted Over Suspected Cyberattack. ABC News. Accessed: May 4, 2022. [Online] https://abcnews.go.com/Technology/wireStory/toyotas-japan-production-halted-suspected-cyberattack-83155113

[42] "CVE website," CVE vulnerability data, 2022. Accessed: May 4, 2022. [Online] https://www.cvedetails.com/browse-by-date.php

[43] M.J.H. Faruk, S. Santhiya, S. Hossain, V. Maria, X. Li. (2022). "Software Engineering Process and Methodology in Blockchain-Oriented Software Development: A Systematic Study". 20th IEEE/ACIS SERA 2022.

[44] U. Paramita, M.J.H. Faruk, N. Mohammad, M. Mohammad, S. Hossain, U. Gias, B. Shabir, R. Akond, A. Sheikh. (2022). "Evolution of Quantum Computing: A Systematic Survey on the Use of Quantum Computing Tools". 1st Int. Conf. on AI in Cybersecurity (ICAIC).

[45] D. Radjenovic, M. Hericko, R. Torkar, and A. zivkovic, "Software fault prediction metrics: A systematic literature review," Inf. Softw. Technol., vol. 55, no. 8, pp. 1397-1418, Aug. 2013.

[46] D. Votipka, R. Stevens, E. Redmiles, J. Hu, and M. Mazurek, "Hackers vs. Testers: a comparison of software vulnerability discovery processes," in Proc. IEEE Symp. Secur. Privacy (SP), May 2018, pp. 374-391.

[47] M. Mohammad, S. Hossain, H. Hisham, M.J.H. Faruk, V. Maria, K. Md, R. Mohammad, A. Muhaiminul, C. Alfredo, W. Fan. (2021). "Bayesian Hyperparameter Optimization for Deep Neural Network-Based Network Intrusion Detection". 10.1109/BigData52589.2021.9671576.

[48] M. Mohammad, M.J.H. Faruk, S. Hossain, Q. Kai, L. Dan, A. Muhaiminul. (2022). "Ransomware Classification and Detection With Machine Learning Algorithms". 10.1109/CCWC54503.2022.9720869.

[49] M.J.H. Faruk, S. Hossain, V. Maria, B. Farhat, S. Shahriar, K. Abdullah, W. Michael, C. Alfredo, L. Dan, R. Akond, W. Fan. (2021). "Malware Detection and Prevention using Artificial Intelligence Techniques". 10.1109/BigData52589.2021.9671434.

[50] K. Kim, D. K. Kim, J. Noh and M. Kim, "Stable Forecasting of Environmental Time Series via Long Short Term Memory Recurrent Neural Network," in IEEE Access, vol. 6, pp. 75216-75228, 2018, doi: 10.1109/ACCESS.2018.2884827.

[51] S. M. Ghaffarian and H. R. Shahriari, âSoftware vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey,' ACM Comput. Surv., vol. 50, no. 4, p. 56, Nov. 2017.

[52] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Appl. Soft Comput., vol. 27, pp. 504-518, Feb. 2015.