A Survey on Vulnerability Prediction using GNNs

Evangelos, D., Katsadouros
University of West Attica
katsadouros.v@uniwa.gr

Charalampos, Z., Patrikakis
University of West Attica
katsadouros.v@uniwa.gr

ABSTRACT

The massive release of software products has led to critical incidents in the software industry due to low-quality software. Software engineers lack security knowledge which causes the development of insecure software. Traditional solutions for analysing code for vulnerabilities suffer from high false positives and negative rates. Researchers over the last decade have proposed mechanisms for analysing code for vulnerabilities using machine learning. The results are promising and could replace traditional static analysis tools or accompany them in the foreseeable future to produce more reliable results. This survey presents the work done so far in vulnerability detection using Graph Neural Networks (GNNs). Presents the GNNs architectures, the graph representations, the datasets, and the results of these studies.

CCS CONCEPTS

• **Security and privacy**; • **Software and application security**; • **Software security engineering**;

KEYWORDS

Software, Privacy, Software Quality

ACM Reference Format:
Evangelos, D., Katsadouros and Charalampos, Z., Patrikakis. 2022. A Survey on Vulnerability Prediction using GNNs. In *26th Pan-Hellenic Conference on Informatics (PCI 2022), November 25–27, 2022, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3575879.3575964

1 INTRODUCTION

In the digital era, software products are in demand in people's daily lives. The massive release of software in the 21st century has led to significant risks over people's personal data. As Joseph Menn reports in a recent Reuters article [1], even the largest companies in the software industry struggle to deliver safe products to their customers. In the last decade, we faced several disastrous data breaches, with severe consequences as regards violation of privacy and personal information, such as the Yahoo data breach in 2013, the Facebook data breach in 2019, and the My Fitness Pal data breach in 2018 [2], all of which resulted to the exposing of billions (for the first case) and millions (for the second and third case) of users' data.

PCI 2022, November 25–27, 2022, Athens, Greece

ACM ISBN 978-1-4503-9854-1/22/11…$15.00
https://doi.org/10.1145/3575879.3575964

As depicted in [3], security incidents like data breaches have proliferated over the years. And though software technology provides tools that can assist in detecting attacks and security breaches, software professionals still lack the tools that will assist them in delivering secure and reliable software. According to Rob Lemos, an award-winning freelance journalist who covers security-related topics, software engineers have failed at security, and there is an equally increasing need for security knowledge and expertise in software engineers [4]. This knowledge and mechanisms can help them develop secure software, thus reducing the risk of attacks and data breaches.

Unfortunately, the long working hours required by software engineers during application developments make it difficult for them to dedicate extra time to analyze their code in search of security flaws or keeping us with the latest developments in the software industry regarding security flaws. Thus, the availability of mechanisms that could inform them or warn about security flaws in their code and assist in solving them would be a valuable aid.

All the above have led to increased research efforts on security topics, with the topic of vulnerability prediction drawing the research community's attention. Several studies have been published on vulnerability prediction mechanisms, with most of them focusing on identifying some of the most common vulnerabilities based on OWASP's top ten[1].

In general, we can divide vulnerability prediction mechanisms into two distinct categories. The first category includes legacy approaches which consist of static analysis, dynamic analysis, pattern-based technics, etc. The other category includes mechanisms that use machine learning, which is the most recent and most effective approach based on recent research. In the last few years, a new approach arrived in vulnerability detection using deep neural networks, the use of Graph Neural Networks (GNNs). Graph Neural Networks have proved as a strong option in vulnerability detection achieving accuracies of over 90% [5], [6].

In this paper, we present the existing work on vulnerability detection, emphasising on State-of-the-Art approaches based on the use of Deep Neural Networks and especially Graph Neural Networks, while we present the existing research gaps in vulnerability detection using these approaches. The rest of the paper is organized as follows: Chapter 2 presents the related work in vulnerability detection. Chapter 3 is a brief introduction to the fields of Graph Neural Networks and Vulnerability Prediction. Chapter 4 presents the taxonomy of vulnerability prediction work, and the work that has been done in this field and finally, the Conclusions section presents the results of this work, the gaps in the field of vulnerability prediction and the opportunities for future work.

[1] https://owasp.org/www-project-top-ten/#

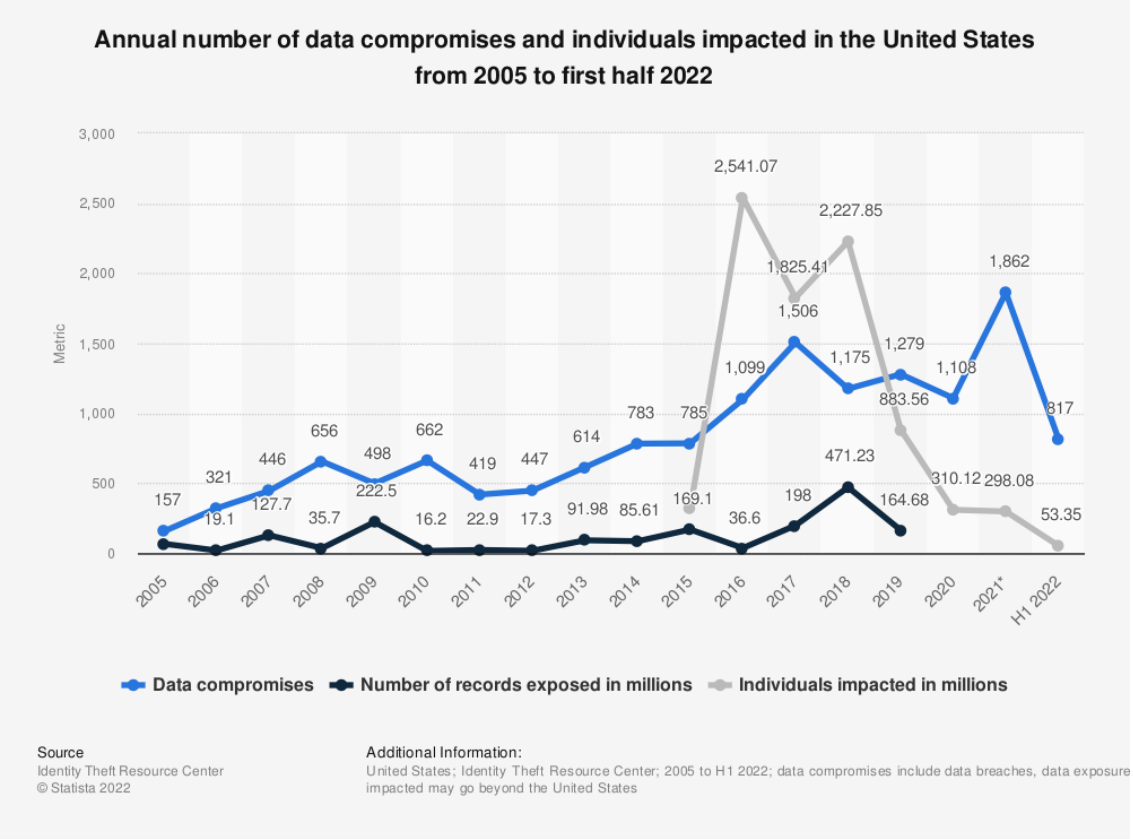

Figure 1: Data Breaches Graph [3]

2 VULNERABILITY PREDICTION USING MACHINE LEARNING

The use of Source code static analysis is a fertile method of debugging by examining source code. However, the use of static analysis for detecting vulnerabilities suffers from high rates of false positives [7]. Over the last decade, Machine Learning (ML) has been proposed as a possible solution to this problem. Many researchers have conducted studies on software vulnerability prediction using ML and their results are quite promising, since the deployment of ML-powered tools demonstrate low rates of false positives and negatives and high overall accuracy rates [7], [8]. The era of vulnerability prediction using ML started with studies that utilised conventional ML and continued with studies that used Deep Neural Network (DNN) architectures. Many studies have been conducted on summarizing vulnerability prediction using machine learning research.

In [9], an extensive analysis of vulnerability prediction using ML technics is presented. In particular, the authors define four categories of vulnerability prediction studies (Vulnerability Prediction Models based on Software Metrics, Anomaly Detection Approaches, Vulnerable Code Pattern Recognition, and Miscellaneous Approaches). In this study, the researchers denote the problem of low performance in vulnerability prediction using conventional ML, and in particular, the major problem is the high rates of false positives. Most of the studies presented here detect vulnerabilities as a binary classification problem so, there is a gap in multiclass classification which is much more practical than binary classification. Finally, a remarkable observation is that the traditional software metrics do not really help in vulnerability prediction task as seem not to be tight related to software vulnerabilities.

Another extensive survey [10] describes the studies conducted in vulnerability prediction using Deep Learning (DL). The authors of the survey propose a new categorization on the field of vulnerability prediction using DL. Their proposed categorization consists of four distinct categories, based on the code feature representation, the graph-based feature representations, the sequence-based feature representations, the text-based feature representations, and the text-based feature representations. One of the most important findings of this research is the fact that the more complex a neural network is the less code analysis is required. Less code analysis it also means that there is no need for code security experts. The DNNs give us the ability to feed the directly the code without spending time on feature engineering which requires security knowledge. Another interesting conclusion of this work is that it is difficult to define code vulnerability features as the code vulnerability patterns is too diverse.

In their work Shen and Chen [11] analysed software automatic analysis from three aspects: software vulnerability detection, software program repair, and software defect prediction. In this survey, alike the previous mentioned earlier, researchers denote that vulnerability prediction using DNN is still giving high rates of false positives. They also report that the use of automatic vulnerability prediction using DNNs doesn't offer good performance in large-scale and diverse projects. Each model is trained and limited to specific programming language and vulnerability types thus, its impractical to use to for vulnerability prediction in real world projects. In case of defect prediction, they stated that the research on the cross-project prediction is limited. Finally, the authors identify five potential areas of future research: feature generation, model selection, datasets, performance evaluation, and feature parameters.

In [12], Zeng et al. analyzed 22 studies in vulnerability prediction using ML and they tried to identify the four studies considered as game changers in the area of vulnerability prediction. Each game changer defines a contribution domain in the field of vulnerability prediction using ML. The four contribution domains are features representations, code analysis and processing, model design, and finally application. Based on their research there is a wide-open area for research in vulnerability detection. Furthermore, the authors state that there is a need for research on vulnerability localization because file-level or function level code might contain thousands of lines of code and there is a need for a real-world public dataset consisted of thousands of samples in order to effectively train a model, and as a final conclusion they state that the use of Graph Neural Networks (GNNs) could be a solution in information loss like the code structural information.

3 VULNERABILITY PREDICTION USING GNNS

Having completed the reference to the most characteristic surveys describing the use of Machine Learning in vulnerability prediction, in the following, we will elaborate on studies presenting the use of GNNs for accomplishing even better prediction results.

3.1 Binary Classification

The study of Zhou et al. in [13] can be considered as the first study in vulnerability prediction using GNNs. In this study, researchers tried to tackle the problem of vulnerability prediction by training a GNN using a custom crafted dataset. The dataset they crafted is consisted of functions written in C/C++ which extracted from 4 open-source projects Linux Kernel, QEMU, Wireshark, and FFmpeg giving in total 58965 graph samples from which the 27652 are vulnerabilities samples and the rest of them non-vulnerabilities. So, the dataset they created it's not perfectly balanced but at the same time is not extremely imbalanced as well. They proposed a DNN to treat vulnerability prediction as a binary classification problem. Their model consists of three steps, the graph embedding layer of composite code semantics, the gated graph recurrent layers and a conv layer. To evaluate their model, they used 4 ML prediction methods (Metrics + Xgboost, 3-layer BiLSTM, 3-layer BiLSTM + Att and CNN) and their model outperformed all the aforementioned methods by achieving accuracies and F1 scores around 75%.

Wang et al. [5], proposed a GNN and an automated data collection mechanism for vulnerability prediction. In details, they proposed an extended version of gated graph neural network [14] which is a stacked neural network which consists of four embedding models based on Gated Recurrent Unit [15]. They feed their network with graph samples represented with an extended version of the AST, which they extend to have eight different edge types, and for the nodes, they use word2vector [16] to embed the nodes to numerical continuous vectors. Furthermore, their model aggregates information through relation graphs instead of the nodes the most GNNs propose. To train their model they used data from CVE, NVD, and open-source projects and for evaluation purposes they used samples written in C, Java, PHP, and Swift (gathered from SARD, NVD, and open-source projects). Finally, their proposed model trained as a binary and multi-class classifier achieving on average accuracy of 92%.

In [17] Wu et al. presented a binary classifier for vulnerability prediction in C/C++ code. Their study proposed the use of Simplified Code Property Graphs (SCPG) which combines the AST and CFG to a more descriptive graph representation of source code. Their approach can be described as three steps, the first one is the transformation of samples to graph representation using the Joern[2] tool, the second step is the graph representation learning and the final step is the classification. For the second step they tried three different GNNs, Graph Attention Network (GAN) [18], Graph Convolutional Network (GCN) [19], and Gated Graph Network (GGN) [14] with the first one (GAT) achieving the best performance (F1 score of 83.6%). To train their models they used the SARD dataset and to reduce its bias they applied prepossessing to each sample by replacing function names with FUN# and variables name with VAR# where # is the incremental number of each function and variable respectively. Finally, they trained two more classifiers using CNNs and RNNs in order to compare their performance with GNNs, with GNNs performing slightly better than them by achieving much better results in Recall measurement and, as such, better F1 score.

The study of Cao et al. in [20] proposed using bidirectional graph neural network (BGNN) along with CNNs to create a binary classifier for detecting vulnerabilities in code written in C/C++. To train their model they manually crafted a dataset from four open-source projects (Linux Kernel, FFmpeg, Wireshark, and Libav) and they proposed a new graph representation, the Code Composite Graph (CCG) which combines AST, CFG, and DFG to a joint directed graph with edges that link the nodes and each link has a label of AST, CFG or DFG respectively. Same as [17], they used Joern to extract graphs from the code samples and after that they vectorized the nodes using word2vec. Each node of the CCG contains the embeddings of the leaf nodes of AST concatenated with the type of the node (e.g., Identifier). Finally, their findings showed that their model outperforms the state-of-the-art (SoTA) approaches they chose to compare it with, by achieving accuracy of ~75% and F1 score of ~77%. Also, they showed that using directed graphs can give much better than using undirected graphs.

Nguyen et al. [21] proposed a graph neural network model for detecting vulnerabilities in multiple programming languages, so, they proposed a language agnostic mechanisms as a solution to vulnerability prediction task. They proposed a neural network consisted of a graph neural network (GCN or GGNN) with residual connections

[2] https://joern.io/

among layers followed by a readout layer for predicting vulnerabilities. In their proposed model the use of GCN achieved the best accuracy of 63.69 % which outperformed many SoTA models like Devign, CodeBERT, and GraphCodeBERT. To train and test their model they used the CodeXGLUE dataset [22] which contains in total 27518 manually labeled samples.

In study [23], Sahin et al. proposed a mechanism for predicting vulnerable versions of code using GNNs by exploiting not only code but commits as well. In particular, they tested Graph Convolutional Networks and GraphSAGE along with other deep learning and machine learning architectures. To train the GNN models they collected samples from the Wireshark project and to feed them to a GNN models they transformed the collected into AST representation. Also, they used a similar method to word2vec to learn nodes' features. To identify the location of vulnerability into the code they used the commits along with the SZZ algorithm. Finally, the use of GraphSAGE for detecting vulnerable code achieved the best score by achieving F1 score of 74.4% and AUC-ROC score of 96.0%.

In [24], Hin et al. presented a work on statement level vulnerability prediction. In particular, they created their own dataset of C/C++ samples to train a model that utilizes Transformers along with GNNs in a node classification problem. They used CodeBERT to obtain embeddings from functions and statements and after that they concatenate functions' embeddings obtained by codeBERT along with the graph embeddings obtained from a GAT layer to predict if a statement is vulnerable or not. Their proposed method outperformed the SoTA in vulnerability prediction in statement level, although their model achieved poor results in detecting vulnerabilities.

Similarly to [24], Cao et al. [25] proposed a statement-level vulnerability classifier utilizing GNNs and binary node classification for detecting memory-related vulnerabilities. In details, they trained a Flow-Sensitive GNN (FS-GNN) which embeds not only the statement nodes but flow edges as well. To feed their model with data they created their own dataset using as sources the SARD and CVE and to transform the samples into PDGs they used Joern. Furthermore, they enriched PDG representation with additional semantic information and utilized Doc2Vec [26] in order to transform statements into numerical vectors. To overcome the problem of imbalanced nodes (most of the nodes were non-vulnerable nodes) they used GraphSMOTE [27] which add synthetic nodes generated based on the similarity of nodes. Finally, their model (FS-GNN) achieved accuracy of 77.5 % and F1 score of 59.5 % and outperformed many static-analysis approaches (PCA, Saber, Flawfinder, Rats, and Infer) and deep learning approaches (VulDeePecker [7], SySeVR [8], and Devign [13]).

Finally, the last study presented in this paper is the one conducted by Ding et al. [28], which as the previous, proposes a binary node classifier for detecting vulnerability at the statement level. In this study, the researchers proposed using ensemble learning to learn both local and long-range dependencies in code. They independently trained a transformer model for capturing long-range dependencies and a GGNN for capturing local dependencies. To train their ensemble model, they first used Juliet (SARD) dataset as it contains much more samples than any publicly available dataset. After that, they fine-tuned their model using the D2A dataset, which consists of real-world code samples, in order for the model to generalize in real-world projects. For feeding with inputs, the two models (transformer and GGNN) converted samples into CPG for the GNN model, and for the transformers, they used the embeddings of the AST statements in sequence. Their approach achieved great results when evaluated in the Juliet dataset with accuracy and F1 score of ~99 %. However, evaluating it on the D2A dataset (real-world samples) did not perform that well by barely achieving an accuracy of 59.3 % and an F1 score of 59.0%. Finally, their research showed that their approach outperforms traditional static analyzers, and using an ensemble model can give better results than using a single model like GNN or transformers. Furthermore, the use of fine-tuning on a real-world dataset can mitigate the problem of generalization.

3.2 Multiclass Classification

In [29] researchers Rabheru et al. proposed a multiclass classifier to tackle the problem of vulnerability prediction in PHP. They trained their model in both function and file level in order to compare the performance of the model. In details, they propose a hybrid DNN consisting of two parallel layers: the first one is multi-layer bidirectional Gated Recurrent Unit (GRU) [30] and the second one is a Graph Convolution Network (GCN) [19]. Using the first one the authors were able to extract feature from sequence of token while using the second one they were able to extract structural feature from CFGs. In front of these layers there are two embedding layers which transforms tokens and graphs to vectors of real numbers to be feed to the next layers. Their classifier is a four-label classifier which predict XSS, SQLi, and OSCI vulnerabilities. To train their model they used the SARD dataset and samples extract from real-world projects from Github. They trained their model using the combination of these two and they also trained using explicitly each one of them in order to compare the results of their model trained only in synthetic samples and in real-world samples. An interesting result of this study is that when their model trained and tested on the SARD dataset gave great results (F1 score close to 95%) but when it was tested in the real-world samples it failed giving an F1 score of ~6.5%. Finally, the model trained at file-level performed slightly better than the one trained at function-level despite the fact that the file-level samples contain a lot of unrelated code to a vulnerability.

In [6], Cheng et al. proposed a multi-class GNN model for predicting vulnerabilities in source code and is one of the few studies that performs vulnerability prediction on statement level which is the finest-grained level of analysis. In this study researchers used a GNN architecture consisted of multiple graph convolution and graph pooling blocks and a graph readout layer for creating a flatten representation of a graph in order to perform the classification. Before feeding the samples to the neural network they pass samples through a pre-processing layer which performs token symbolization which transform functions and variables name into more generic forms like VAR0 for a variable and FUN0 for a function name. After that, the statements are passed through Doc2Vec [26] to create a vector representation of each statement. To train their model they used a dataset, crafted from the SARD dataset and two real world projects (lua and redis), of more than 600K slice samples most of them being safe samples and most of the extracted

Study	GNN	Graph Representation	Dataset	Accuracy/F1 Score
[13]	Gated Graph Recurrent Networks	Composite Graph (AST, CFG, DFG, and NCS)	Custom Dataset (Linux Kernel, QEMU, Wireshark, FFmpeg)	72.26% / 73.26%
[5]	Extended Graph Gated Neural Network	Extended AST	Custom Dataset (Multiple Real-World Projects + Synthetic Samples)	~92% / ~94%
[17]	Graph Attention Network, Graph Convolutional Network, and Gated Graph Neural Network	Simplified CPG	SARD	88.9% / 83.9%
[20]	Bidirectional Graph Neural Network	CCG	Custom Dataset (Linux Kernel, FFmpeg, Wireshark, and Libav)	74.7% / 76.8%
[21]	GNN with Residual Connection	Unique token-focused graph & Index-focused graph	CodeXGLUE	63.69% / -
[23]	GraphSAGE & Graph Convolutional Network	AST	Wireshark	- / 74.4%
[24]	Graph Attention Network, and Graph Convolutional Network	PDG, and CDG	Real-World Custom Dataset (Big-Vul)	- / 36.0%
[25]	Flow-Sensitive Graph Neural Network	Extended PDG	Custom Memory-Related Vulnerabilities Dataset (SARD & CVE)	74.1% / 65.2%
[28]	Gated Graph Sequence Neural Network	CPG	Juliet (SARD)	58.9% / 59.0%
[29]	Graph Convolutional Network	CFG	SARD & Real-World Projects	File Level: 96.56% / 96.11 Function Level: 85.84% / 75.05%
[6]	Graph Convolutional Network, Graph Attention Network, and k-dimensional GNNs	PDG	SARD & Real-World Projects (lua & redis)	97.4% / 95.6%

slices being part the SARD dataset. Finally, they achieved very good results (95.6 of F1 score for the ten vulnerabilities) and their model was pretty fast at the inference time.

3.3 Summarizing the surveyed work

From the presented survey on the work done globally on the GNN assisted vulnerability prediction, it is obvious that the re is no standardised way on presenting the results of the proposed ideas, since different metrics are used by the researchers. However, there is a common base upon which all work can be summarized and presented, and this includes the reporting of the graph representation adopted, dataset used, and the accuracy and F1 score achieved (with the exception of [23] and [24], where only F1 score is presented. This highlights the need for adoption of a common way to evaluate the efficiency of use of GNNs in detecting vulnerabilities. The following table includes the summary of the work surveyed, in the manner described in this paragraph

4 CONCLUSIONS

In this study we presented the state of the art in vulnerability prediction using GNNs. Most of the research is focused on finding mechanisms for binary classification for vulnerability prediction using GNNs. Binary classification in real-world is not that helpful as cannot give any sight on what is the vulnerability in particular. Multi-class can be more helpful for software engineers, though the research on multi-class classification is quite limited. Although the recent research tried to introduce mechanisms for finding methods on detecting vulnerabilities in the finest granularity level (e.g., statement level) but the results of that research are not promising. Furthermore, most of the studies predict vulnerabilities at the function level

Another issue is the lack of real-world datasets. As S. Chakraborty et al. [31] observe, the performance of many DL approaches for finding vulnerabilities decreases if we test them on real-world and unseen code. Thus, it is very important in this field the construction of a large database that will consists of real-world code samples instead of synthetic ones. Furthermore, is really important to test developed models on unseen samples to test their ability to generalize in code that the models have never seen before. Therefore, the area of vulnerability prediction requires a mechanism that will provide researchers with real-world vulnerability samples for training and validation purposes and at the same time will construct a test set that will be completely different from the training-validation set. Finally, is also important to identify duplicates or very similar samples in a dataset and removing them as they have negative impact on the generalization of a trained model.

ACKNOWLEDGMENTS

The implementation of the doctoral thesis was co-financed by Greece and the European Union (European Social Fund-ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning»in the context of the Act "Enhancing Human Resources Research Potential by undertaking a Doctoral Research" Sub-action 2: IKY Scholarship Programme for PhD candidates in the Greek Universities

REFERENCES

[1] Joseph Menn. 2021. Major tech companies struggle to plug holes in logging software. Reuters. Retrieved February 22, 2022 from https://www.reuters.com/technology/major-tech-companies-struggle-plug-holes-logging-software-2021-12-16/

[2] Michael Hill and Dan Swinhoe. 2021. The 15 biggest data breaches of the 21st century. CSO Online. Retrieved February 22, 2022 from https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

[3] Data breaches and individuals impacted U.S. 2022. Statista. Retrieved September 28, 2022 from https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/

[4] Rob Lemos. Security liability is coming for software: Is your engineering team ready? TechBeacon. Retrieved February 22, 2022 from https://techbeacon.com/security/security-liability-coming-software-your-engineering-team-ready

[5] Huanting Wang, Guixin Ye, Zhanyong Tang, Shin Hwei Tan, Songfang Huang, Dingyi Fang, Yansong Feng, Lizhong Bian, and Zheng Wang. 2021. Combining Graph-Based Learning With Automated Data Collection for Code Vulnerability Detection. IEEE Transactions on Information Forensics and Security 16, (2021), 1943–1958. DOI:https://doi.org/10.1109/TIFS.2020.3044773.

[6] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. ACM Trans. Softw. Eng. Methodol. 30, 3 (May 2021), 1–33. DOI:https://doi.org/10.1145/3436877

[7] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. Proceedings 2018 Network and Distributed System Security Symposium (2018). DOI:https://doi.org/10.14722/ndss.2018.23158

[8] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2022. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Trans. Dependable and Secure Comput. 19, 4 (July 2022), 2244–2258. DOI:https://doi.org/10.1109/TDSC.2021.3051525

[9] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. ACM Comput. Surv. 50, 4 (August 2017), 56:1-56:36. DOI:https://doi.org/10.1145/3092566

[10] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software Vulnerability Detection Using Deep Neural Networks: A Survey. Proc. IEEE 108, 10 (October 2020), 1825–1848. DOI:https://doi.org/10.1109/JPROC.2020.2993293

[11] Zhidong Shen and Si Chen. A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques. Security and Communication Networks, 16.

[12] Peng Zeng, Guanjun Lin, Lei Pan, Yonghang Tai, and Jun Zhang. 2020. Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey. IEEE Access 8, (2020), 197158–197172. DOI:https://doi.org/10.1109/ACCESS.2020.3034766

[13] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. arXiv:1909.03496 [cs, stat] (September 2019). Retrieved January 10, 2022 from http://arxiv.org/abs/1909.03496

[14] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2017. Gated Graph Sequence Neural Networks. Retrieved August 13, 2022 from http://arxiv.org/abs/1511.05493

[15] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Retrieved August 13, 2022 from http://arxiv.org/abs/1406.1078

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13), Curran Associates Inc., Red Hook, NY, USA, 3111–3119.

[17] Yuelong Wu, Jintian Lu, Yunyi Zhang, and Shuyuan Jin. 2021. Vulnerability Detection in C/C++ Source Code With Graph Representation Learning. In 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC), IEEE, NV, USA, 1519–1524. DOI:https://doi.org/10.1109/CCWC51732.2021.9376145

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [cs, stat] (February 2018). Retrieved April 1, 2022 from http://arxiv.org/abs/1710.10903

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. arXiv:1609.02907 [cs, stat] (February 2017). Retrieved April 1, 2022 from http://arxiv.org/abs/1609.02907

[20] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. 2021. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. Information and Software Technology 136, (August 2021), 106576. DOI:https://doi.org/10.1016/j.infsof.2021.106576

[21] Van-Anh Nguyen, Dai Quoc Nguyen, Van Nguyen, Trung Le, Quan Hung Tran, and Dinh Phung. 2021. ReGVD: Revisiting Graph Neural Networks for Vulnerability Detection. arXiv:2110.07317 [cs] (October 2021). Retrieved January 10, 2022 from http://arxiv.org/abs/2110.07317

[22] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. Retrieved August 22, 2022 from http://arxiv.org/abs/2102.04664

[23] Sefa Eren Şahin, Ecem Mine Özyedierler, and Ayse Tosun. 2022. Predicting vulnerability inducing function versions using node embeddings and graph neural networks. Information and Software Technology 145, (May 2022), 106822. DOI:https://doi.org/10.1016/j.infsof.2022.106822

[24] David Hin, Andrey Kan, Huaming Chen, and M. Ali Babar. 2022. LineVD: Statement-level Vulnerability Detection using Graph Neural Networks. Retrieved August 29, 2022 from http://arxiv.org/abs/2203.05181

[25] Sicong Cao, Xiaobing Sun, Lili Bo, Rongxin Wu, Bin Li, and Chuanqi Tao. 2022. MVD: memory-related vulnerability detection based on flow-sensitive graph neural networks. In Proceedings of the 44th International Conference on Software Engineering, ACM, Pittsburgh Pennsylvania, 1456–1468. DOI:https://doi.org/10.1145/3510003.3510219

[26] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. 9.

[27] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. 2021. GraphSMOTE: Imbalanced Node Classification on Graphs with Graph Neural Networks. DOI:https://doi.org/10.1145/3437963.3441720

[28] Yangruibo Ding, Sahil Suneja, Yunhui Zheng, Jim Laredo, Alessandro Morari, Gail Kaiser, and Baishakhi Ray. 2021. VELVET: a noVel Ensemble Learning approach to automatically locate VulnErable sTatements. arXiv:2112.10893 [cs] (December 2021). Retrieved January 10, 2022 from http://arxiv.org/abs/2112.10893

[29] Rishi Rabheru, Hazim Hanif, and Sergio Maffeis. 2021. A Hybrid Graph Neural Network Approach for Detecting PHP Vulnerabilities. Proceedings of the 36th Annual ACM Symposium on Applied Computing (March 2021), 1687–1690. DOI:https://doi.org/10.1145/3412841.3442132

[30] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. DOI:https://doi.org/10.48550/arXiv.1412.3555

[31] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2020. Deep Learning based Vulnerability Detection: Are We There Yet? Retrieved August 29, 2022 from http://arxiv.org/abs/2009.07235