Proceedings of the 6th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/92/20241722

Survey of deep learning models for vulnerability detection

Jiaoyang Wang

College of Software, Taiyuan University of Technology, Shanxi, 030000, China

mekamme33@gmail.com

Abstract. With the rapid development of information technology, software security issues have become increasingly prominent, especially the importance of software vulnerability detection technology, which has continued to increase. This paper reviews vulnerability detection technology based on neural networks, with a special focus on two widely used models: Devign and CodeBERT. By analyzing the working principles and processes of these two models, their performance and advantages in dealing with vulnerability detection tasks in complex codes are explored. With the advancement of technology, an analysis is conducted on the update and development of these two different technologies from various perspectives. At the same time, this paper also analyzes the challenges that existing models may encounter and proposes future development directions, including data processing, model design improvements, and innovations in feature extraction technology, in order to improve the accuracy and efficiency of vulnerability detection technology.

Keywords: Software Vulnerability, Deep Learning, defect detection, comparative study, vulnerability classification.

1. Introduction

Software vulnerabilities are no longer unfamiliar to the public. According to the '2023 Annual Cyber Threat Security Review' report by Qualys, record-breaking 26,447 computer vulnerabilities were disclosed worldwide in 2023, a 5.2% increase from last year's 25,050. The sheer number of vulnerabilities is alarming[1]. Detecting software vulnerabilities is a crucial aspect in ensuring the security of software. With the advancement of technology, vulnerability detection methods have been evolving from traditional rule-based detection methods to machine learning-based technologies. In particular, the introduction of deep learning technology provides a new solution to the problems of automation and efficiency in vulnerability detection. At present, scholars have conducted extensive comparative studies and experiments on vulnerability detection models based on neural networks. Different models have their own advantages in handling software vulnerability detection, but they also face some common problems and challenges. For example, Krishnan et al. conducted a detailed comparison of the effects of deep learning methods in Web application security in a study and found that although these methods performed well in detecting complex vulnerabilities, they still had obvious limitations in dealing with imbalanced data sets and adversarial attacks[2]. In addition, Wartschinski et al. compared the application of deep learning models applied in Python code bases, revealing the advantages and disadvantages of different models in dealing with natural code bases[3]. This study further verified and supplemented the performance of these deep learning models in actual

DOI: 10.54254/2755-2721/92/20241722

vulnerability detection tasks, and proposed improvement suggestions for model design and data processing. This paper reviews neural network-based vulnerability detection technologies, especially two advanced models, Devign and CodeBERT, to explore their working principles, implementation processes, and challenges. This study not only explains the technical details and application scenarios of these models, but also compares their advantages and disadvantages in practical applications, and further points out the future development direction of neural network-based vulnerability detection technology. The findings of this research contribute to promoting vulnerability detection technology toward greater efficiency and intelligence.

2. Background

Software vulnerability detection is a primary method for identifying security vulnerabilities within software systems. This process involves using various tools to audit the software's code or analyze its execution to identify design errors, coding flaws, and operational failures. Based on detection principles, current methods are classified into code similarity-based vulnerability detection, rule-based vulnerability detection, and pattern-based vulnerability detection.

Code similarity-based detection methods primarily target software defects caused by code cloning. However, these methods rely on the surface features of the code, and similar code fragments do not always represent the same vulnerabilities, leading to a high false positive rate[4]. Additionally, rule-based vulnerability detection operates by presenting a set of rules and searching the source code for parts that match these rules, marking them as potential vulnerabilities. While open-source tools such as Flawfinder, Splint, Cvechecker, Cppcheck, and FindBugs provide assistance, they rely on static analysis and have differing principles of detection. These tools are based on rules derived from the knowledge and experience of security experts, which are effective for detecting certain vulnerabilities. However, the rigidity of these rules makes it difficult for them to adapt to evolving vulnerabilities, resulting in many false positives and false negatives[5]. Therefore, future research in vulnerability detection methods should focus on enabling these methods to automatically learn semantics, thereby reducing the reliance on expert knowledge in the vulnerability detection process.

Another method is code vulnerability detection based on vulnerability patterns. It does not solely depend on intricate rules tailored by security experts, but incorporates code features or heuristic guidance provided by experts, in conjunction with artificial intelligence technology, to automatically generate patterns of vulnerability. Such technology can learn the semantics of the code to detect complex variants of vulnerabilities. This type of method is divided into traditional machine learning and deep learning according to the technology. The method using traditional machine learning technology requires experts to define vulnerability features, such as function input, parameter type, text identifier frequency, abstract syntax tree, code attribute graph, open source project submission information, entropy and suffix tree, etc., and then learn and classify vulnerability patterns through machine learning algorithms. The method based on deep learning, however, uses heuristic guidance strategies given by security experts to extract corpus from the source code and uses deep learning algorithms to automatically learn and generate vulnerability patterns, which are then employed to detect code vulnerabilities. This significantly reduces the workload of experts in the workflow.

3. Research of deep learning vulnerability detection

Neural networks are computational models that simulate the structure of human brain neurons, processing and analyzing complex data through layers of interconnected nodes. In recent years, neural networks have made significant advancements in various fields, including the technology for detecting vulnerabilities. This paper focuses on reviewing two neural network-based vulnerability detection models: Devign and CodeBERT.

DOI: 10.54254/2755-2721/92/20241722

3.1. Devign

3.1.1. Model overview. The Devign model is designed based on Graph Neural Networks (GNN), with its core functionality centered on utilizing the Code Property Graph (CPG) to capture the complex semantics and structural features of program code[6]. The CPG integrates Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG), creating a comprehensive graphical structure. Combining these various code representations into a unified graph structure, provides a holistic view for program analysis, thereby enabling more effective detection of security vulnerabilities in the code.

The primary components of the Devign model are as follows:

Graph Neural Network Layer: Devign processes the Code Property Graph using Graph Convolutional Networks (GCN). Each GCN layer deeply learns representations of code components (such as functions, variables, etc.), considering various dependencies between nodes.

Feature Fusion: Through multiple GCN layers, Devign aggregates information from neighboring nodes at each layer, gradually forming an advanced representation of global code semantics. This layered aggregation mechanism enables the model to understand complex program structures and semantics.

Classifier: The representations of all nodes in the graph are combined through global average pooling to form a graph-level feature representation, which is used for the final vulnerability prediction.

According to previous research, the performance of the Devign model varies across different types of vulnerabilities. Here are its specific performance and analysis in detecting different types of vulnerabilities:

The Devign model has excellent performance in buffer overflows and numerical errors[7]. These two types of vulnerabilities are more difficult to detect in traditional program analysis, which requires tracking variable values and loop reasoning. Its GNN architecture can effectively capture these complex dependencies and data flows, and therefore performs well in these types of detections.

The performance of the Devign model is significantly affected when dealing with complex vulnerabilities (such as resource errors and input validation errors) that span control flow and data flow dependencies across functions[7][8]. Since these vulnerability types require a deep understanding of the semantic structure of the code, Devign falls short in this high-level semantic understanding, causing its accuracy and F1 scores to perform poorly in these cases. Furthermore, the training data set may not adequately cover all resource types and modes of input validation, resulting in insufficient model generalization in these areas.

There is great variability in the performance of the Devign model when faced with different data sets, especially when the data set contains multiple types of vulnerabilities, this variability is more obvious[6].

3.1.2. Updates based on graph neural network models. As the demand for vulnerability detection continues to increase, the Devign model faces challenges in practical applications. In order to further improve the Devign model's vulnerability detection efficiency and accuracy in complex code bases, researchers have made many improvements and optimizations to the model. These improvements are mainly focused on graph simplification technology and enhanced graph neural network levels, which enhance the performance of the Devign model in large-scale, complex code bases by reducing the model's computational burden and improving its representation learning capabilities.

1)Implementation of Graph Simplification Techniques: Graph simplification primarily aims to reduce the number of nodes and the complexity of edges in the graph, helping to decrease the computational burden and enhance processing speed. The specific methods include:

Node Merging: Combining nodes with similar functions or attributes into a single node, reducing the total number of nodes while maintaining the integrity of the graph's structure and semantics.

Edge Simplification: Optimizing the connections between nodes by removing unnecessary edges preserving key data and control flow paths.

DOI: 10.54254/2755-2721/92/20241722

Distance Compression: Utilizing algorithms to minimize the maximum distance between nodes in the graph, thereby enhancing the efficiency of information transfer.

2)Enhanced Graph Neural Networks: Enhanced graph neural networks improve the model's representation learning ability by introducing more efficient graph convolutional layers and new graph learning mechanisms:

Edge-aware Graph Convolutional Networks: This new type of graph convolutional network can distinguish between different types of edges, such as data flow edges and control flow edges, allowing the model to handle the complexity of code graphs more precisely.

Multi-scale Feature Fusion: By fusing node features at different levels, the model can learn both local and global code characteristics, enhancing the understanding of deep code semantics.

Dynamic Attention Mechanism: Dynamically adjusting the relationship weights between different nodes allows the model to adaptively focus on more relevant parts of the code according to the task requirements.

According to Wen et al., by simplifying the graph, the accuracy of the model has been significantly improved compared to the unsimplified one: the accuracy on the Reveal dataset has increased from 87.99% to 92.71%; by introducing the edge-aware graph convolutional network (EA-GCN) and the kernel scaling representation module, the performance of the model has been significantly enhanced. On the Reveal dataset, the F1 score increased from 37.27% to 48.48% after incorporating the EA-GCN module, and from 30.14% to 48.48% after integrating the kernel scaling representation module[9].

These technologies make the Devign model more efficient and accurate when processing large and complex codebases. Graph simplification enhances the model's processing speed, while enhanced graph neural networks provide richer and more precise code representations, thereby improving the accuracy of vulnerability detection.

3.2. CodeBERT

3.2.1. Model overview. CodeBERT is a Transformer-based pre-trained model designed for hybrid tasks involving programming languages and natural languages. It learns the syntax and semantics of programming languages and the association between code and its annotations by pre-training on a large number of open-source code bases and corresponding natural language annotations. This bimodal training strategy enables CodeBERT to demonstrate excellent performance in a variety of programming-related tasks, such as code search and code documentation generation[10].

Using the CodeBERT model for defect prediction mainly includes the following steps:

Fine-tuning the model: After CodeBERT is pre-trained on a large amount of code and annotation data, it needs to be fine-tuned for specific vulnerability detection tasks.

Feature extraction: CodeBERT can automatically extract important features from the code. These features are not just the surface text of the code, but also the structure and contextual information of the code. In this way, the model can identify complex patterns and potential security issues in the code.

Prediction and evaluation: Use the fine-tuned model to analyze new code snippets to determine whether these codes may contain vulnerabilities. The model will provide a probability score indicating the likelihood of a vulnerability in the code. The model's predictions can then be verified through actual code review or testing tools.

Output explanations: To allow developers to better understand the model's predictions, CodeBERT can provide detailed explanations of the predictions. For instance, it is capable of indicating which lines of code may contain issues or providing an explanation for why the code is deemed to be risky.

Based on Ruitong LiU, CodeBERT significantly outperforms other mainstream methods in detection performance on four vulnerability datasets, as shown in Table1[11].

DOI: 10.54254/2755-2721/92/20241722

Table 1. Performance metrics of various models on datasets with specific CWEs

Model	F1(%) DiverseVuL	Draper VDSIC	Devign	Reveal
Bert	17.83	78.8	56.11	36.04
TextCNN	10.03	67.86	58.37	22.94
TextCCN	13.27	67.64	38.7	20.69
CodeBERT	23.44	82.39	61.62	38.19

Additionally, the research found that CodeBERT's performance trends on different types of vulnerabilities were similar to those of Devign, but the overall results were higher than those of the Devign model[7]. CodeBERT exhibits advantages in semantic understanding. By being pre-trained on large-scale code repositories and natural language data, it can better comprehend the semantic relationships between code and its annotations, excelling in handling complex code.

3.2.2. Parameter-based fine-tuning. According to the research by Alexey Shestov et al., the performance of CodeBERT in vulnerability detection tasks can be enhanced through meticulous parameter tuning [12]. By fine-tuning specific aspects such as adjusting the learning rate and batch size, and employing focal loss to address class imbalance, CodeBERT becomes more effective at identifying potential vulnerabilities in source code. Experimental results show that these optimization measures significantly improve the model's accuracy and generalization capabilities, demonstrating the critical role of targeted parameter adjustments in enhancing the performance of specific tasks[12].

4. Discussion

In current research, in addition to the aforementioned Devign and CodeBERT models, there are many mainstream neural network-based vulnerability detection techniques. For example, VulEye utilizes GNN for vulnerability detection, specifically targeting vulnerabilities in PHP applications[13]. The model combines Gated Recurrent Units and GCN to detect vulnerabilities such as SQL injection, cross-site scripting, and operating system command injection. Experiments in SQL-Labs and XSS-Labs show that VulEye achieves an accuracy of 88.41% for SQLI vulnerability detection and 72.22% for XSS vulnerability detection. VulEye further optimizes vulnerability detection through the SDG slicing method, significantly improving detection accuracy compared to traditional CFG and PDG representation methods, especially in multi-class classification scenarios.

Current research indicates that the effectiveness of deep learning methods for vulnerability detection significantly diminishes when applied in real-world settings[14]. This downturn in performance is largely attributed to the quality of training data and oversimplifications in model choice, where models often learn features irrelevant to the actual causes of vulnerabilities, such as specific variables or function names instead of genuine vulnerability characteristics.

To overcome these limitations, future directions could explore:

Data Handling: Enhancing the diversity and realism of datasets to avoid issues like data duplication and imbalance, enabling models to learn more representative features of vulnerabilities.

Model Design: Employing more complex and effective neural network architectures to better capture the intricate semantics and logic of code, moving beyond simplistic tag-based or token-based models.

Feature Extraction Techniques: Developing novel methods to improve the understanding of code context and dependencies, thus enhancing the accuracy and robustness of vulnerability detection.

Such improvements could pave the way for neural network-based vulnerability detection technologies to achieve greater accuracy and efficiency in the future, providing stronger safeguards for software security.

DOI: 10.54254/2755-2721/92/20241722

5. Conclusion

In conclusion, this paper has reviewed neural network-based vulnerability detection technologies, focusing on the architectures, performance, and advancements of models like Devign and CodeBERT. Through a detailed analysis of these models, their potent capabilities in capturing the complex semantics and structural characteristics of code have been illuminated. However, these models also face practical challenges, such as issues related to the realism of datasets and inherent limitations in model design. Future research can further explore data processing and model design. The dataset explored may not comprehensively cover all code types in real-world environments, thereby limiting the extent to which the research results can be generalized. Looking ahead, enhancing data processing techniques and refining model architectures will be crucial steps to improving the robustness and accuracy of vulnerability detection. Innovations in feature extraction methods and the development of more comprehensive datasets will further enable these models to adapt more effectively to real-world applications. In summary, this study provides useful references and inspiration for the future development of vulnerability detection technology based on neural networks. However, further exploration is necessary in terms of dataset richness and model complexity to achieve more efficient and accurate results in practical applications.

References

[1] "TotalCloud Security Insights." Qualys, Inc., 2023, www.qualys.com/2023/totalcloud-security-insights/.

[2] Kadar et al. "A Comparative Analysis of Deep Learning Approaches for Enhancing Security in Web Applications." 2023. The Proceedings of the International Conference on Smart City Applications. Cham: Springer Nature Switzerland.

[3] Lin, Guanjun, et al. 2021. "Deep neural-based vulnerability discovery demystified: data, model and performance." Neural Computing and Applications 33.20: 13287-13300.

[4] Xu, Yixiao, et al. 2023. "A Review of Code Vulnerability Detection Techniques Based on Static Analysis." International Conference on Computational & Experimental Engineering and Sciences. Cham: Springer Nature Switzerland.

[5] Croft, Roland, et al. 2021. "An empirical study of rule-based and learning-based approaches for static application security testing." Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM).

[6] Zhou, Yaqin, et al. 2019. "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks." Advances in neural information processing systems 32.

[7] An cong et al. 2021. "An empirical study on software defect prediction using codebert model." Applied Sciences 11.11: 4793.

[8] Sindhwad, Parul V., et al. 2024. "VulnArmor: mitigating software vulnerabilities with code resolution and detection techniques." International Journal of Information Technology: 1-16.

[9] Wen, Xin-Cheng, et al. 2023. "Vulnerability detection with graph simplification and enhanced graph representation learning." 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE.

[10] Feng, Zhangyin, et al. 2020. "Codebert: A pre-trained model for programming and natural languages." arXiv preprint arXiv:2002.08155.

[11] Liu, Ruitong, et al. 2024. "Source Code Vulnerability Detection: Combining Code Language Models and Code Property Graphs." arXiv preprint arXiv:2404.14719.

[12] Shestov, Alexey, et al. 2024. "Finetuning large language models for vulnerability detection." arXiv preprint arXiv:2401.17010.

[13] Lin, Chun, et al. 2023. "VulEye: A novel graph neural network vulnerability detection approach for PHP application." Applied Sciences 13.2: 825.

[14] Chakraborty, Saikat, et al. 2021. "Deep learning based vulnerability detection: Are we there yet?." IEEE Transactions on Software Engineering 48.9: 3280-3296.