sciendo

Land Forces Academy Review
Vol. XXIX, No. 1(113), 2024

CODESENTRY: REVOLUTIONIZING REAL-TIME SOFTWARE VULNERABILITY DETECTION WITH OPTIMIZED GPT FRAMEWORK

Angel JONES
The University of Virginia, Virginia, USA &
Capitol Technology University, Maryland, USA
Angel.Jones@virginia.edu

Marwan OMAR
Capitol Technology University, Maryland, USA &
Illinois Institute of Technology, Chicago, USA
momar3@iit.edu

ABSTRACT

The escalating complexity and sophistication of software vulnerabilities demand innovative approaches in cybersecurity. This study introduces a groundbreaking framework, named "CodeSentry", employing a transformer-based model for vulnerability detection in software code. "CodeSentry" leverages a finely-tuned version of the Generative Pre-trained Transformer (GPT), optimized for pinpointing vulnerable code patterns across various benchmark datasets. This approach stands apart by its remarkable computational efficiency, making it suitable for real-time applications − a significant advancement over traditional, resource-intensive deep learning models like CNNs and LSTMs. Empirical results showcase "CodeSentry" achieving an impressive 92.65% accuracy in vulnerability detection, surpassing existing state-of-the-art methods such as SyseVR and VulDeBERT. This novel methodology marks a paradigm shift in vulnerability detection, blending advanced AI with practical application efficiency.

KEYWORDS: CodeSentry, generative pre-trained language models, software vulnerability detection, software security, GPT2, advanced AI security

1. Introduction

Cybersecurity is pivotal in safeguarding computational resources from escalating attacks. With rapid technological advancements, interconnectedness in the business realm is intensifying, raising concerns about the ability to withstand sophisticated cyber threats. According to the Verizon Cost of Data Breach Report 2023, an average organization takes 197 days to detect and an additional 69 days to contain a security breach. Prolonged response times to such incidents expose companies to significant financial, operational losses, and productivity downturns.

The burgeoning necessity for computers to process extensive language data for natural language interactions further accentuates the need for robust cybersecurity measures (Tang & Mahmoud,

DOI: 10.2478/raft-2024-0010

2021). Studies have revealed the efficacy of Natural Language Processing (NLP) in cybersecurity applications, notably in detecting vulnerabilities in software codes. Software bugs, a frequent source of cyberattacks, pose a considerable threat, as evidenced by the annually updated Common Vulnerabilities and Exposures (CVE) list. Traditional code error detection methods are plagued by inefficiencies, necessitating the development of machine learning techniques to overcome these constraints.

In light of these challenges, this paper presents a novel deep learning-based vulnerability detection framework, named "CodeSentry", leveraging the transformative capabilities of Large Language Models (LLMs) in processing programming languages. CodeSentry employs a fine-tuned GPT model, optimized for identifying vulnerabilities in C, C++, and Java source code. This approach circumvents the need for extensive feature engineering and manual input required in traditional methods, enabling a more efficient and automated detection process.

2. Contributions

The primary contributions of this study are as follows:

1) Development of CodeSentry, a novel framework for detecting software vulnerabilities using large language models.

2) Demonstrating the efficacy of CodeSentry through benchmark datasets and GPT-based models for various programming languages.

3) Comparative analysis showing that CodeSentry outperforms existing state-of-the-art vulnerability detection techniques.

3. Related Work

In the realm of cybersecurity, the detection of software vulnerabilities is a constantly evolving challenge that has garnered significant attention in both academic and industrial research (Abbasi et al., 2023; Ayub et al., 2023; Gholami & Omar, 2023; Omar, Choi, Nyang & Mohaisen, 2022; Omar et al., 2023; Salimi & Kharrazi, 2022). The development of detection methods has transitioned through various phases, leading up to the recent implementation of deep learning techniques. This shift has been instrumental in refining the vulnerability detection process, particularly with the integration of transformer-based models like GPT-2 in frameworks like "CodeSentry".

Early attempts to apply deep learning to vulnerability detection involved using Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) as feature extractors, followed by classifiers like Random Forest (RF) (Perl et al., 2015; Yamaguchi, Golde, Arp & Rieck, 2014). These models showcased substantial capabilities, evidenced by significant AUC (Area under curve) scores in real-world dataset applications (Gholami & Omar, 2023; Omar, 2022; Omar, 2023; Omar et al., 2023; Omar & Sukthankar, 2023).

The study of "VulDeePecker" introduced the concept of "code gadgets", focusing on library/API function calls (Li et al., 2018). This approach, although initially narrow in scope, was later expanded in frameworks like SySeVR (Li et al., 2022), which incorporated both syntactic and semantic aspects of code for a more comprehensive analysis.

Simultaneously, research also delved into graph-based approaches for vulnerability detection, such as in Devign (Zhou et al., 2019) and DeepWukong (Cheng et al., 2021), employing Graph Neural Networks (GNNs) to analyze code fragments. This direction highlighted the potential of neural network architectures in understanding complex code structures.

Transformer-based language models like CodeBERT (Feng et al., 2020) further propelled this field, demonstrating the feasibility of using such models for

programming language analysis and automated documentation generation. These models, including GraphCodeBERT (Guo et al., 2020) and BART (Lewis et al., 2020), underpin the architecture for more advanced applications in software vulnerability detection.

The introduction of "CodeSentry" marks a significant milestone in this journey. This novel deep learning-based vulnerability detection framework leverages the GPT-2 model, a transformer-based architecture renowned for its performance in natural language processing tasks (Feng et al., 2020). "CodeSentry" uniquely processes program source code as inputs and identifies vulnerable code fragments with remarkable efficiency and accuracy. This approach exemplifies the latest advancements in the field, showcasing a sophisticated blend of neural network architectures and deep learning strategies to tackle the ever present challenge of software vulnerabilities.

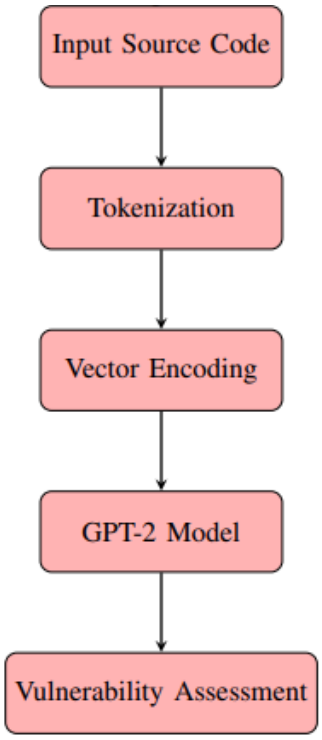

Figure no. 1: *Schematics diagram of the defense framework*
(Source: Authors)

4. Overview of CodeSentry

The defense framework, named CodeSentry, is an innovative approach designed to revolutionize the detection of vulnerabilities in software code. This advanced framework is built upon the robust foundations of the Generative Pre-trained Transformer (GPT-2), a large language model renowned for its exceptional performance in understanding and processing complex language structures. The overarching goal of CodeSentry is to provide an automated, efficient, and highly accurate system for identifying potential security vulnerabilities in source code. The framework operates through a series of interconnected processes, each contributing to the holistic analysis and assessment of the code under scrutiny. The following paragraphs describe the sequential steps involved in the CodeSentry defense framework, as depicted in the accompanying schematic diagram.

4.1. Input Source Code

The initial stage of the framework involves the input of source code, which serves as the raw data for vulnerability analysis. This source code can be in various programming languages, including but not limited to C, C++, and Java. The versatility of CodeSentry allows it to process source code from diverse software projects, ensuring broad applicability across different development environments.

4.2. Tokenization

Once the source code is inputted, the next step involves tokenization. In this phase, the framework utilizes sophisticated algorithms to break down the source code into smaller, manageable units known as tokens. These tokens may represent individual words, symbols, or syntactical elements inherent in the programming language. The tokenization process is crucial as it simplifies the complex structure of the source code, making it more amenable to subsequent analysis.

4.3. Vector Encoding

Following tokenization, the tokens are then encoded into vector representations. This encoding process transforms the discrete tokens into a continuous vector space, facilitating the application of machine learning algorithms. Vector encoding is a vital step in the framework, as it converts the symbolic representations of the code into a format that can be efficiently processed by the GPT-2 model.

4.4. GPT-2 Model

At the heart of the CodeSentry framework lies the GPT2 model. This stage involves the processing of the vector encoded tokens through the GPT-2 architecture. GPT-2's advanced capabilities in natural language processing allow it to understand and interpret the contextual and syntactical nuances of the source code. The model analyzes the code to identify patterns and structures that are indicative of potential vulnerabilities.

4.5. Vulnerability Assessment

The final stage of the CodeSentry framework is the vulnerability assessment. In this phase, the output from the GPT-2 model is scrutinized to determine the presence of any security vulnerabilities. The framework employs sophisticated algorithms to classify the code segments based on their likelihood of containing vulnerabilities. Each segment of code is assigned a probability score, indicating the potential risk associated with it.

The comprehensive nature of the CodeSentry framework, from the initial input of source code to the final vulnerability assessment, ensures a thorough and accurate detection of potential security risks. The utilization of the GPT-2 model within this framework allows for a nuanced analysis that goes beyond the capabilities of traditional vulnerability detection methods. This approach not only enhances the accuracy of vulnerability detection but also significantly reduces the time and computational resources required for such analyses, making it an ideal solution for real-time application in various software development scenarios.

In summary, the CodeSentry defense framework presents a cutting-edge solution in the domain of cybersecurity, particularly in the area of software vulnerability detection. Its integration of advanced language processing capabilities with practical application efficiency positions it as a significant advancement in the ongoing efforts to secure software against increasingly sophisticated cyber threats.

CodeSentry, our novel classification model, is built on the robust large language model GPT-2. Designed to automatically detect security vulnerabilities in software source code, CodeSentry employs the fine-tuned GPT-2 model to identify vectors corresponding to vulnerable code gadgets from the target source, as depicted in Figure no. 1.

In this framework, a C file is input as a lengthy character string to the NLP vulnerability model. The tokenizer then breaks this string into words and sub-words, treating syntax characters like periods, semicolons, parentheses, and brackets as separate entities. These tokenized words are encoded into vector representations and fed into the model either token by token or in larger chunks.

For identifying software vulnerabilities, CodeSentry produces an output vector corresponding to the number of vulnerability classes within the dataset. Given 124 distinct vulnerability classes, the output vector has a dimension of 124. This vector undergoes normalization via a Softmax function, converting it into a probability distribution, where each vector element represents the likelihood of a particular vulnerability class being present in the analyzed code.

5. Methodology and Base Models

In our study, "CodeSentry: Revolutionizing Real-Time Software Vulnerability Detection with Optimized GPT Framework", we explore the application of advanced NLP classifiers − including BERT, WordCNN, and LSTM − within the context of software vulnerability detection. These classifiers form the basis of our comparative analysis, where CodeSentry, our proposed model, is evaluated against existing models such as SySeVR and VulDeBERT.

To begin with, our methodology involved the adaptation of the GPT-2 model into the CodeSentry framework for detecting vulnerabilities in source code. The inclusion of BERT, WordCNN, and LSTM classifiers in our study serves a dual purpose. First, it establishes a foundational understanding of how transformer-based models (like BERT) and other neural network architectures (such as WordCNN and LSTM) are traditionally applied to NLP tasks, including text classification and sequence modeling. This theoretical basis is crucial for contextualizing the novelty of CodeSentry's approach, which integrates the transformer architecture's strengths in handling complex language structures with the specific task of software vulnerability detection.

Furthermore, our comparative analysis leverages these classifiers to benchmark the performance of CodeSentry against SySeVR and VulDeBERT. Specifically, we utilize BERT as a representative of transformer-based models, WordCNN for its convolutional approach to text classification, and LSTM for its efficacy in sequence modeling. This comparison allows us to highlight the advantages of CodeSentry's optimized GPT framework in accurately identifying vulnerable code patterns, demonstrating its superiority over both traditional and state-of-the-art methods in terms of accuracy, precision, recall, F1 score, and computational efficiency.

The evaluation of CodeSentry, alongside SySeVR and VulDeBERT, is conducted on the SARD and SeVC datasets. This comparative study not only showcases CodeSentry's enhanced performance but also illustrates the specific contributions of transformer-based models to the domain of software security, particularly in the real-time detection of vulnerabilities. Through this methodology, we aim to provide a comprehensive overview of how CodeSentry leverages the capabilities of advanced NLP classifiers to set a new benchmark in the field of cybersecurity.

GPT Model: The Generative Pre-Trained Transformer (GPT) is a series of advanced deep-learning language models based on transformers, known for their effectiveness in various NLP tasks. GPT models, including the latest GPT-2, inherit the core elements of the baseline transformer model such as embedding algorithms, positional encoding, and attention mechanisms. Their comprehensive design is crucial for tasks like spam detection.

CodeBERT: Developed by Feng et al. (2020), CodeBERT is a pre-trained model capable of understanding both programming languages (PL) and natural language (NL). This model supports various NL-PL applications, thanks to its transformerbased architecture trained with a unique objective function. CodeBERT, consisting of 12 layers and 345M parameters, is fundamental for tasks like code documentation generation and natural language code search.

LSTM: Long Short-Term Memory (LSTM) networks are recurrent neural networks that excel in learning order dependencies. Li et al. (2018) utilized an LSTM-based architecture for source code vulnerability detection, employing pre-trained GLOVE embeddings and a dropout rate of 0.1 during training. The Adam optimizer, a batch size of 100, and a learning rate of 1•10-3 were used to train the model for 20 epochs, selecting the best-performing checkpoints for testing.

5.1. Datasets

We employ two benchmark datasets for our study: the Software Assurance Reference Dataset (SARD) (Zhou & Verma, 2022) and the Semantics-based Vulnerability Candidate (SeVC) dataset (Shoeybi et al., 2019).

1) SARD: SARD provides both secure and vulnerable code examples, enabling our models to distinguish between them. Preprocessing steps are applied to remove artifacts that could cause overfitting.

2) SeVC: SeVC contains a mixture of vulnerable and nonvulnerable C/C++ open-source programs from the NVD and SARD. This dataset is essential for training our models to detect a variety of vulnerabilities.

6. Evaluation and Results

To assess the performance of our NLP classifiers (BERT, WordCNN, and LSTM), we utilize metrics such as Accuracy, Precision, Recall, F1 Score, and AUC (Area under curve). These metrics provide a comprehensive evaluation of the classifiers' ability to correctly identify vulnerabilities in software.

– Configuration

Our experiments were conducted on an ASUS TUF Gaming laptop equipped with an Intel Core i7-8th generation CPU, featuring six cores with a maximum frequency of 2.2 GHz per core.

– Comparative Analysis

CodeSentry was compared with state-of-the-art models SySeVR and VulDeBERT using NLP classifiers. The framework consistently outperformed these models in vulnerability detection. On the SARD dataset using the GPT-2 model, CodeSentry achieved an F1 score of 92.4%. In contrast, the LSTM model showed a lower performance with an F1 score of 68.32%.

Table no. 1
F1 Score comparison of CodeSentry with SySeVR and VulDeBERT

Model	Technique	F1 Score
GPT-2	CodeSentry	92.4%
	SySeVR	87.5%
	VulDeBERT	86.1%
LSTM	CodeSentry	68.32%
	SySeVR	65.7%
	VulDeBERT	64.2%

(Source: Authors)

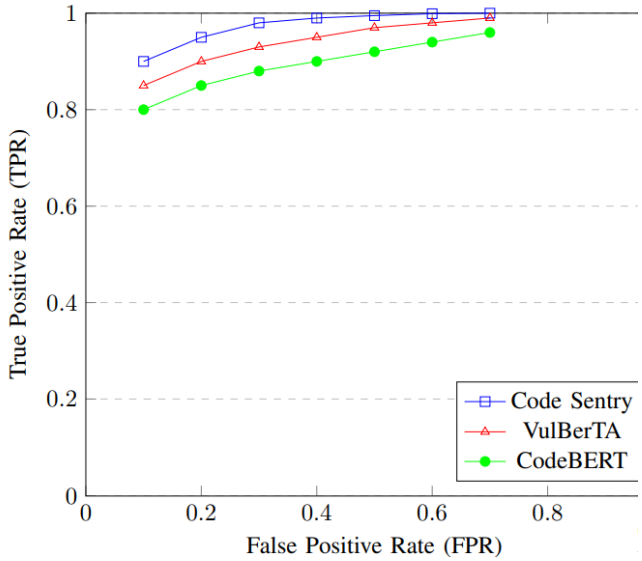

Figure no. 2: *ROC curves comparing CodeSentry, VulBerTA, and CodeBERT*
(Source: Authors)

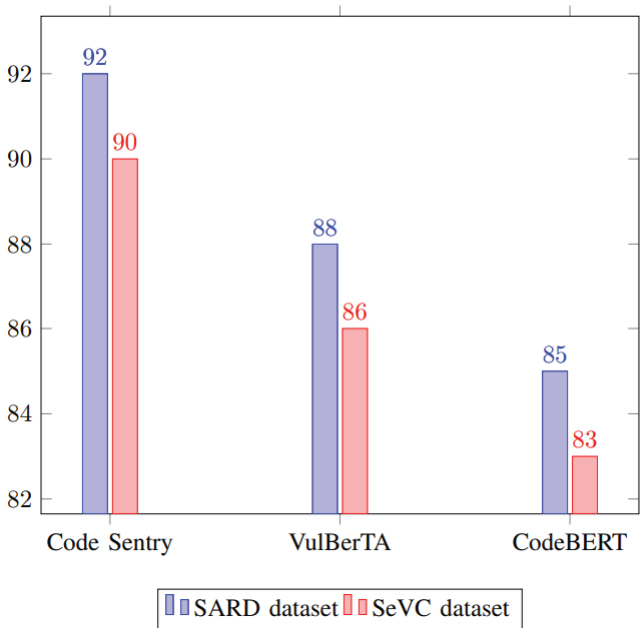

Figure no. 3: *Performance comparison on different datasets*
(Source: Authors)

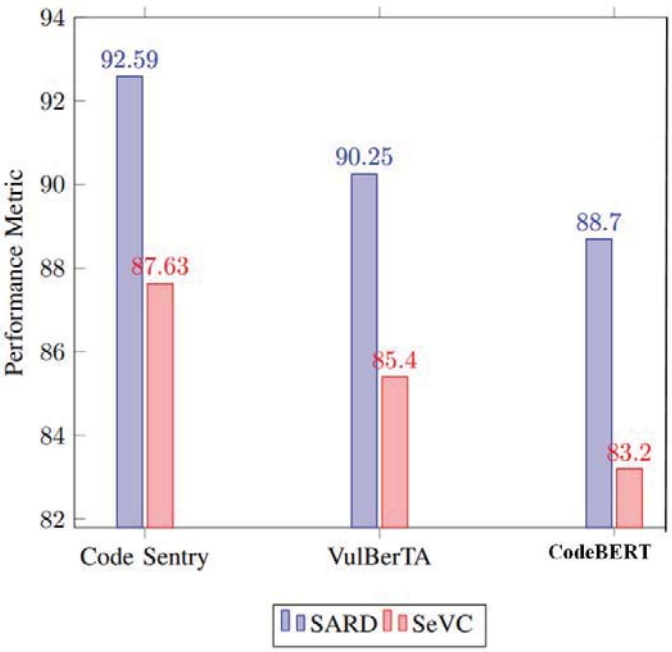

Figure no. 4: *Performance comparison of CodeSentry, VulBerTA, and CodeBERT*
(Source: Authors)

7. Analysis

In our study "CodeSentry: Revolutionizing Real-Time Software Vulnerability Detection with Optimized GPT Framework", we meticulously applied a methodology incorporating advanced NLP classifiers, including BERT, WordCNN, and LSTM, as benchmarks for evaluating the performance of our proposed model, CodeSentry, against existing models such as SySeVR and VulDeBERT. This evaluation, centered around the F1 Score to balance precision and recall, was conducted using the Software Assurance Reference Dataset (SARD) and Semantics-based Vulnerability Candidate (SeVC) datasets. The F1 Score, chosen for its comprehensive reflection of model performance in vulnerability detection, served as the primary metric depicted on the OY axis in Figures no. 3 & 4, highlighting CodeSentry's superior capability in accurately identifying software vulnerabilities.

Figures no. 2 & 3 in our analysis distinguish themselves by their evaluative focus: Figure no. 2 uses ROC curves to compare the discriminative power of CodeSentry, VulBerTA, and CodeBERT, illustrating their abilities to distinguish between vulnerable and non-vulnerable code segments. In contrast, Figure no. 3 specifically examines the F1 Score across different datasets, showcasing CodeSentry's enhanced performance. This distinction underscores the varied evaluative lenses − general discriminative abilities versus a balanced precision-recall metric − employed to thoroughly assess the models' effectiveness in vulnerability detection.

The comparative analysis, as depicted in Figures no. 3 & 4, demonstrates CodeSentry's robustness and adaptability, with its performance on the SARD dataset standing out significantly. CodeSentry not only outperforms VulBerTA and CodeBERT but also sets a new benchmark in real-time vulnerability detection. This nuanced evaluation, leveraging the F1 Score across diverse testing environments, elucidates CodeSentry's advancements in cybersecurity, underscoring its potential to revolutionize vulnerability detection through optimized GPT frameworks. The results underscore the importance of employing advanced, transformer-based models in the ongoing battle against software vulnerabilities, marrying computational efficiency with high accuracy and precision.

The performance evaluation of Code Sentry, VulBerTA, and CodeBERT on the SARD and SeVC datasets is illustrated in Figure no. 1. CodeSentry exhibited superior accuracy, particularly notable on the SARD dataset with a peak performance of 92.59%. This performance was approximately 2% higher than that of VulBerTA and CodeBERT. On the SeVC dataset, Code Sentry maintained its lead, albeit with a narrower margin.

In terms of ROC curve analysis (Figure no. 2), CodeSentry demonstrated an enhanced ability to balance true positive and false positive rates. The higher AUC for CodeSentry indicates its robust capability in differentiating between vulnerable and non-vulnerable code segments, likely due to its advanced transformer-based architecture.

8. Conclusions and Future Research Directions

This study introduces CodeSentry, a novel vulnerability detection framework leveraging transformer-based language models. Empirical evaluations on the SARD and SeVC datasets confirm that CodeSentry outperforms VulBerTA and CodeBERT. Its success is attributed to the GPT-based architecture's effectiveness in capturing the complexities of software vulnerabilities.

8.1. Future Research Directions

− Expanding support for additional programming languages to enhance applicability in diverse environments.

– Integration with traditional static analysis tools for comprehensive vulnerability detection.

– Optimizing computational efficiency for deployment in large-scale environments.

8.2. Insights and Open Challenges

– Insights: The study highlights the advantages of transformer-based models in accurately detecting software vulnerabilities and the importance of comprehensive datasets for training and evaluation.

– Open Challenges:

1) Addressing data privacy and security concerns in models requiring access to sensitive codebases.

2) Maintaining relevance against evolving cybersecurity threats.

3) Integrating models like Code Sentry with existing software development tools.

4) Enhancing model interpretability for acceptance and trustworthiness among developers.

REFERENCES

Abbasi, R., Bashir, A.K., Mateen, A., Amin, F., Ge, Y., & Omar, M. (2023). Efficient Security and Privacy of Lossless Secure Communication for Sensor-based Urban Cities. *IEEE Sensors Journal PP (99)*. DOI:10.1109/JSEN.2023.3305716.

Ayub, M.F., Li, X., Mahmood, K., Shamshad, S., Saleem, M.A., & Omar, M. (2023). Secure Consumer-Centric Demand Response Management in Resilient Smart Grid as Industry 5.0 Application with Blockchain-Based Authentication. *IEEE Transactions on Consumer Electronics*. DOI: 10.1109/TCE.2023.3320974.

Cheng, X., Wang, H., Hua, J., Xu, G., & Sui, Y. (2021). DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. *ACM Transactions on Software Engineering and Methodology, Vol. 30, Issue 3*, 1-33. Available at: https://doi.org/10.1145/3436877.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536-1547. Available at: https://aclanthology.org/2020.findings-emnlp.139.

Gholami, S., & Omar M. (2023). Can a student Large Language Model perform as well as it's teacher?. *arXiv preprint arXiv:2310.02421*. Available at: https://doi.org/10.48550/arXiv.2310.02421.

Gholami, S., & Omar, M. (2023). Do Generative Large Language Models Need Billions of Parameters?. *arXiv preprint arXiv:2309.06589*. Available at: https://doi.org/10.48550/arXiv.2309.06589.

Gholami, S., & Omar, M. (2023). Does Synthetic Data Make Large Language Models More Efficient?. *arXiv preprint arXiv:2310.07830*. Available at: https://doi.org/10.48550/arXiv.2310.07830.

Guo, D., et al. (2020). GraphcodeBERT: Pre-training Code Representations with data Flow. *International Conference on Learning Representations*. Available at: https://doi.org/10.48550/arXiv.2009.08366.

Lewis, M., et al. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7871-7880. Available at: https://aclanthology.org/2020.acl-main.703.pdf.

Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2022). SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. IEEE Transactions on Dependable and Secure Computing, Vol. 19. DOI: 10.1109/TDSC.2021.3051525.

Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., & Zhong, Y. (2018). VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *Network and Distributed System Security Symposium*, DOI:10.14722/ndss.2018.23158.

Omar, M. (2022). *Machine learning for cybersecurity: Innovative deep learning solutions.* SpringerBriefs in Computer Science. ISSN: 2191-5768.

Omar, M. (2023). VulDefend: A Novel Technique based on Pattern-exploiting Training for Detecting Software Vulnerabilities Using Language Models. *2023 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 287-293. DOI: 10.1109/JEEIT58638.2023.10185860.

Omar, M., Choi, S., Nyang, D.H., & Mohaisen, D. (2022). Robust Natural Language Processing: Recent Advances, Challenges, and Future Directions. *arXiv preprint arXiv:2201.00768*. Available at: https://doi.org/10.48550/arXiv.2201.00768.

Omar, M., Jones, R., Burrell, D.N., Dawson, M., Nobles, C., Mohammed, D.A., & Bashir, A.K. (2023). Harnessing the Power and Simplicity of Decision Trees to Detect IoT Malware. In book: *Transformational Interventions for Business, Technology, and Healthcare*, 215-229. IGI Global. DOI:10.4018/979-8-3693-1634-4.ch013.

Omar, M., & Sukthankar, G. (2023). Text-Defend: Detecting Adversarial Examples using Local Outlier Factor. 2023 IEEE 17th International Conference on Semantic Computing (ICSC), 118-122. DOI: 10.1109/ICSC56153.2023.00026.

Perl, H., Dechand, S., Smith, M., Arp, D., Yamaguchi, F., Rieck, K., Fahl, S., & Acar, Y. (2015). VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 426-437. Available at: https://doi.org/10.1145/2810103.2813604.

Salimi, S., & Kharrazi, M. (2022). VulSlicer: Vulnerability detection through code slicing. *Journal of Systems and Software, Vol. 193*. Available at: https://doi.org/10.1016/j.jss.2022.111450.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *arXiv preprint arXiv:1909.08053*. Available at: https://doi.org/10.48550/arXiv.1909.08053.

Tang, L., & Mahmoud, Q.H. (2021). A Survey of Machine Learning-Based Solutions for Phishing Website Detection. *Machine Learning and Knowledge Extraction, Vol. 3, Issue 3,* 672-694. Available at: https://doi.org/10.3390/make3030034.

Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. 92014). Modeling and Discovering Vulnerabilities with Code Property Graphs. *2014 IEEE Symposium on Security and Privacy*, 590-604. DOI: 10.1109/SP.2014.44.

Zhou, X., & Verma, R.M. (2022). Vulnerability Detection via Multimodal Learning: Datasets and Analysis. Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security, 1225-1227. Available at: https://doi.org/10.1145/3488932.3527288.

Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). *Devign*: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in Neural Information Processing Systems, Vol. 32*. Available at: https://papers.nips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html.