Cyber Security Vulnerability Detection Using Natural Language Processing

Kanchan Singh
Computer Science and Technology
UMIT, SNDT Women's University
Mumbai, India
kanchan2311singh@gmail.com

Sakshi S Grover
Computer Science and Technology
UMIT, SNDT Women's University
Mumbai, India
groversakshi1998@gmail.com

Ranjini Kishen Kumar
Computer Science and Technology
UMIT, SNDT Women's University
Mumbai, India
ranjinikk26@gmail.com

***Abstract*—Cybersecurity is the practice of preventing cyberattacks on vital infrastructure and private data. Government organisations, banks, hospitals, and every other industry sector are increasingly investing in cybersecurity infrastructure to safeguard their operations and the millions of consumers who entrust them with their personal information. Cyber threat activity is alarming in a world where businesses are more interconnected than ever before, raising concerns about how well organisations can protect themselves from widespread attacks. Threat intelligence solutions employ Natural Language Processing to read and interpret the meaning of words and technical data in various languages and find trends in them. It is becoming increasingly precise for machines to analyse various data sources in multiple languages using NLP. This paper aims to develop a system that targets software vulnerability detection as a Natural Language Processing (NLP) problem with source code treated as texts and addresses the automated software vulnerability detection with recent advanced deep learning NLP models. We have created and compared various deep learning models based on their accuracy and the best performer achieved 95% accurate results. Furthermore we have also made an effort to predict which vulnerability class a particular source code belongs to and also developed a robust dashboard using FastAPI and ReactJS.**

***Index Terms*—Cybersecurity, NLP, Source Code, Detection**

I. Introduction

Cyber security aims to protect machines from attacks. With technological advancement, businesses are getting interconnected. Organizations' ability to defend themselves against mass attacks is being called into doubt as cyber threats grow in both intensity and sophistication at an alarming rate. Detecting a security breach currently takes an average company 197 days, while containing it takes 69 days, according to IBM's Cost of Data Breach Report 2020 [1]. Companies are at risk of substantial financial and operational losses, unscheduled downtime, and poorer productivity due to this prolonged incident response. Language processing has become more critical in computer-human (natural) language interactions [2], and it's vital to train computers to process and analyze massive amounts of language data. Many studies have discovered that NLP approaches can be applied to more cybersecurity-focused applications, such as detecting flaws in written code. Cyberattacks are frequently the result of bugs in software. The Common Vulnerabilities and Exposures (CVE) list, updated yearly, shows that software vulnerabilities are still a significant threat. Inaccuracy or misleading detection plagues many old methods for identifying errors in code. Developing machine learning technologies that can alleviate the limitations of older methods is necessary because of the intricacy of these vulnerabilities. NLP may be able to discover common security flaws sooner rather than later because their patterns and structures are well-defined.

Our main contributions are as follows:

- We are building a large open text-based dataset comprising raw C/C++ code. We will be using 2021 CWE, i.e., Common Weakness Enumeration's Top 25 Most Dangerous Software Weaknesses [3].
- Second, we tested various deep learning models to extract contextual information from source code files.
- Using API in visualizing the detection model as a dashboard

The rest of this paper consists of seven sections. In Section II we give an overview of the recent work in software vulnerability detection using both static and dynamic code analysis as well as the related deep learning work in transformers and using natural language processing to extract features from computer code. Section III presents the details C/C++ code files dataset we have built, its natural sources, and how it was created. Section IV describes the deep learning and NLP models we used for software vulnerability detection. Then, Section V compares the performance of these modules on software vulnerability detection with the dataset we developed. Section VI and VII talk about the APIs we used to deploy the models as a dashboard for vulnerability detection.

II. Related Work

There has been much research regarding NLP being adapted to a more cybersecurity-focused application, such as finding bugs in written code. There are many analysis tools available today to identify common vulnerabilities. However, traditional code analysis techniques have been imprecise, inefficient, and full of errors. We referred and researched through 10 different papers from reputed journals to study

2022 IEEE World AI IoT Congress (AIIoT) | 978-1-6654-8453-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/AIIOT54504.2022.9817336

the work in this field.

The NLP model proposed by Noah Ziems and Shaoen Wu (2021) [4] can be used to derive situational meaning from structured texts. Automated software vulnerabilities in source code can be discovered using deep learning NLP models and transfer learning from written English. Using the NIST NVD/SARD databases, a dataset of more than 100 categories of vulnerabilities is preprocessed from over 100,000 C programming files. They feed a long string of C code into their deep learning software vulnerability model training, followed by the use of a tokenizer to decode the string. Once the words are encoded, the encoder transforms them into vectors. Word vector representations were then fed into the model in token or chunk form, depending on the model's implementation. The BERT model and its derivatives outperformed all of the typical NLP deep learning models tested when it came to accuracy

Mamoru Mimura and Ryo Ito's(2021) [5]tests show that malware can be detected using NLP approaches in an actual environment. For their detection strategy, they employed language models and classifications. Words from malicious and benign data are combined to create a language model during the training phase. Extracting lexical characteristics from the model is done. Executable files are classified as malicious or benign during the testing process based on the built-in language model and trained classifier. To represent lexical characteristics, they employ a Doc2vec or LSI model. The SVM, Doc2Vec, Doc2Vec+MLP, and LSI+MLP were the top performers.

Rebecca L. Russell, Louis Kim, et al. [6] established a machine learning method for large-scale vulnerability detection at the function level using open-source C and C++ code. There are millions of open-source functions in the database they've built. They used findings from three different static analyzers to label it as a technique to supplement already labeled vulnerability datasets with additional information. An automated vulnerability identification system was developed using deep feature representation learning to translate selected source code directly from this dataset and was also evaluated on code from basic software packages and the NIST SATE IV benchmark datasets.

The study by Xin Li; Lu Wang et al. (2021) [7] used hybrid neural networks to learn the code vulnerability pattern. Their hybrid neural network's convolutional and recurrent layers discovered the local and global features. "Text-CNN" had a detection time of 1.5 s and a training time of 11.0 s/epoch at the very least. However, its precision and recall were the lowest. Bi-GRU outperformed "Text-CNN" by a large margin. Accuracy, precision, recall, and F1-score were above average for their proposed hybrid neural network, with 99.0%, 98.35%, 99.0%, and 98.65%, respectively

Researchers Gaigai Tang, Lianxiao Meng, et al.(2021) [8] studied neural network algorithms to discover software vulnerabilities automatically. They used layered symbolization to generate a symbolic representation of the code gadget as a starting point. Among the code gadgets, there are two distinct areas. Detection models are trained using the first, and potential security issues are discovered using the second. The doc2vec tool is used in the next stage to transform the symbolic form into a low-dimensional vector representation. Bi-LSTM and RVFL neural network techniques are used to train the detection model.

A deep learning-based vulnerability detection system, Vulnerability Deep Pecker (VulDeePecker), was designed and implemented by researchers Zhen Li, Deqing Zou, et al.(2018) [9] to alleviate human specialists from the time-consuming and subjective effort of manually identifying software vulnerability features. A code gadget consists of several (not necessarily consecutive) lines of code that are semantically related. They proposed utilizing these code gadgets to represent programs and then transforming them into vectors. There were fewer false negatives (and reasonable false positives) with VulDeePecker in the experiments. In addition, the researchers used VulDeePecker on three other software packages (Xen, Seamonkey, and Libav) and discovered four unreported vulnerabilities.

The SySeVR: A Framework created by Zhen Li, Deqing Zou, et al. (2021) [10] uses a stratified 5-fold cross-validation to train eight standard models: a linear Logistic Regression (LR) classifier, a neural network with a single hidden layer Multi-Layer Perceptron (MLP), a DBN, a CNN, and four According to their findings, when compared to LSTM and GRU, bidirectional RNNs (i.e., BiLSTM and BGRU) may enhance both the FNR and the F1-measure on average by 4.5% and 2.3%, respectively. According to the researchers, bidirectional RNNs (particularly BGRU) are more effective than CNNs.

Even though machine learning techniques have long been used to address computer security challenges, the quickly evolving Deep Learning technology has recently aroused substantial attention in the security field, as highlighted in the review paper by Yoon-Ho Choi, Peng Liu, and others [11]. Deep Learning applications' eight computer security issues include security-oriented program analysis, defending against return-oriented programming attacks (ROP), achieving control-flow integrity (CFI), defending against network attacks, malware classification, and system-event-based anomaly detection, memory forensics, and software security fuzzing.

The popularity of online transactions and data exchange has grown exponentially in recent years. SQL injections are more common in databases that are connected. Maheli Ahmed and Mohammed Nasir Uddin [12] offered a SQL

injection detection approach using NLP and the Ensemble Learning Algorithm. NLP generates feature patterns, and a bag-of-words model is developed. Using the new model, it is now possible to classify SQL injection payloads more precisely.

An even broader spectrum of security issues can be detected using NLP-based models. More research is needed to build more accurate classifiers and user-friendly dashboards.

III. Development of Dataset

For creating a unique and latest dataset, the project uses the 2021 CWE Top 25 Most Dangerous Software Weaknesses list. Over 70 thousand C/C++ code files of the corresponding weaknesses were downloaded from the SARD dataset [13]. Each of the individual files contain the vulnerable code with respect to its CWE type, and its counterpart safe code.

A. SARD

The Software Assurance Reference Dataset (SARD) [13] is an ever-expanding collection of programs in various languages such as C, C++, Java and PHP with clearly located security flaws. They address over 150 different types of vulnerabilities, including SQL injection, cross-site scripting (XSS), buffer overflow, and the usage of a faulty cryptographic algorithm.

B. Dataset Creation

All of the 70 thousand files were crawled through and the vulnerable and safe codes separated from each of them while eliminating unimportant files. The data was then cleansed by removing comments and all instances of ”good” and ”bad”. Finally, it was labelled accordingly.

C. Dataset Qualities

- The dataset contains almost 1 million rows which are well balanced across programming languages, vulnerable and safe code.
- It covers 24 types of weaknesses and vulnerabilities.

IV. Natural Language Processing for Cyber Security Vulnerability Detection

In this research, we have trained and developed various deep learning models for natural language processing. The models include: LSTM, BiLSTM, BERT, BERT for sequence classification and CodeBERT. In an effort to substantially create a comparison between different models, the models are trained over the entire dataset.

A. Inputs and Outputs

The input to the cyber security vulnerability detection model comprises of source code in C/C++. The input is treated as string, and the BERT tokenizer breaks down the string into tokens. These tokens are then encoded into vectors. As the next process, these vectors are then fed into the respective models. Further, the output of the model predicts whether the input source code is good (safe) or bad (vulnerable).

B. LSTM

Long Short Term Memory(LSTM) maintain the information being passed from each timestamp, which in turn avoids vanishing or exploding gradients. LSTM also has the ability to understand representation when trained on structured code written in C [14]. LSTMs are quite effective at learning structures and syntax in raw source code files (for C).
Under the LSTM model designed by us, the code symbols are an input to the model. The output contains the probability of the vulnerability in the source code.

C. Bidirectional LSTM

LSTM moves forward across a sequence in only one direction. This leads to loss of information at the beginning in cases when the sequences are large. As by the time LSTM reaches the end, the information received in the beginning is lost. Hence, to overcome this problem, Bidirectional LSTM was introduced. BiLSTMs solves the above problem, simply by running two different LSTMs - one going from start to end and the other in the opposite direction [15]. Once, the LSTMs finish moving across the sequence, the outputs obtained during the forward and backward LSTMs are combined for a final prediction.
For our dataset, BiLSTMs play crucial role as the input source code is a long sequence.

D. BERT for Sequence Classification

Bidirectional Encoder Representation from Transformers(BERT), is based on the transformers, which allows every output to be connected with an individual input. BERT was designed to pre-train deep bidirectional representations from unlabeled text by conditioning on both left and right contexts in all layers. BERT models can be fine-tuned with different hyper-parameters for developing models to perform specific tasks. [16] BERT uses the Masked Language Model, under masked models, it randomly masks some tokens from the input and gives prediction according to the original vocabulary based only on the context.

E. CodeBERT

CodeBERT is a bimodal pre-trained model for programming and natural languages. It was specifically designed to learn code documentations, source codes, etc for providing robust support with input as source codes. It is a transformer based architecture, enabling the use of ”bimodal” data of natural language and programming language pairs.
CodeBERT is shown to achieve better performance on source code as input. [17]

V. Performance Evaluation

The following tableI, highlights the comparison between the respective models, which were trained over our dataset.

Based on the above mentioned metrics, we chose to use CodeBERT for this classification task.

Model Comparison				
Model	Accuracy	Precision	Recall	F1 score
LSTM	87%	74%	75%	79%
BiLSTM	93%	93%	93%	93%
BERT	87%	88%	88%	87%
BERT for Sequence Classification	94%	95%	94%	94%
CodeBERT	94%	94%	95%	94%

TABLE I: Performance Evaluation for Vulnerability Classification

VI. Multiclass Classification

We have made an effort to predict to which vulnerability class a particular source code belongs. Based on our previous findings, we decided to use BERT with Sequence Classification and CodeBERT because of their high accuracy and precision. The following table II compares the model's evaluation across BERT for Sequence Classification and CodeBERT

Model Comparison				
Model	Accuracy	Precision	Recall	F1 score
BERT for Sequence Classification	95%	91%	88%	88%
CodeBERT	95%	96%	93%	94%

TABLE II: Performance Evaluation for Multiclass Classification

We compared both of the models and used CodeBERT because of its excellent performance. The model gives one of the 24 classes as its predicted output.

VII. Pipeline

To create a robust dashboard for detecting vulnerability in source code, we developed an end to end deep learning pipeline using FASTAPI and ReactJS.
Source code files in C/C++ languages can be uploaded and a subsequent result can be obtained, highlighting whether the code is vulnerable or not. The CWE(Common Weakness Enumeration) detection helps to understand, the type of vulnerability present. The dashboard presents a detailed overview of the vulnerability and basic debugging information. Our system can detect from over 24 types of vulnerability

A. Backend API

FastAPI is a high performance web framework build over Python to develop Application Programming Interface. FastAPI provides high performance, reduces latency in API response time and is efficient. For our model, we created 2 APIs - one to classify the source code as good or bad and the other to classify the CWE found in the vulnerable code. We hosted our models on the huggingface inference and used them subsequently for the development of the APIs.

B. Frontend Framework

ReactJS is a javascript based framework for developing the client side of applications. We used ReactJS for creating the dashboard. On the dashboard, any user can upload the source code file, this input is send to backend via POST. The FastAPI APIs generate the predictions and return the predicted classes. The predictions are then displayed on the dashboard, along with a brief overview and debugging information of the vulnerability if found.

VIII. Conclusion

In this paper, we first created a system that treats software vulnerability detection as a Natural Language Processing (NLP) issue, with source code represented as texts. Then we address automated software vulnerability detection using the current advanced deep learning NLP model. Then, we created a few deep learning models to detect and categorize software vulnerabilities in source code and compared their accuracy. Extensive research has proven that the CodeBERT model can find and categorize software vulnerabilities in code with outstanding results. Furthermore, we note that it is critical to keep the contextual information throughout the whole sequence for this specific job. Other researchers can utilize the created collection of software vulnerabilities to enhance it further.

References

[1] "Cost of a data breach report 2020 - IBM." [Online]. Available: https://www.ibm.com/security/digital-assets/cost-data-breach-report/1Cost%20of%20a%20Data%20Breach%20Report%202020.pdf.[Accessed:16-Mar-2022.

[2] J. Peacock, "How NLP is transforming cyber risk and compliance," CyberSaint Security. [Online]. Available: https://www.cybersaint.io/blog/ai-cybersecurity.[Accessed:16-Mar-2022.

[3] CWE. 2021 cwe top 25 most dangerous software weaknesses, 2021. [Online]. Available: https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html

[4] N. Ziems and S. Wu, "Security vulnerability detection using Deep Learning Natural Language Processing," IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2021.

[5] M. Mimura and R. Ito, "Applying NLP techniques to malware detection in a practical environment," International Journal of Information Security, 2021. Workshops (INFOCOM WKSHPS), 2021.

[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," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018.

[7] X. Li, L. Wang, Y. Xin, Y. Yang, and Y. Chen, "Automated vulnerability detection in source code using minimum intermediate representation learning," Applied Sciences, vol. 10, no. 5, p. 1692, 2020.

[8] G. Tang, L. Meng, H. Wang, S. Ren, Q. Wang, L. Yang, and W. Cao, "A comparative study of neural network techniques for automatic software vulnerability detection," 2020 International Symposium on Theoretical Aspects of Software Engineering (TASE), 2020.

[9] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," Proceedings 2018 Network and Distributed System Security Symposium, 2018.

[10] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "SySeVR: A framework for using deep learning to detect software vulnerabilities," IEEE Transactions on Dependable and Secure Computing, pp. 1–1, 2021.

[11] Y.-H. Choi, P. Liu, Z. Shang, H. Wang, Z. Wang, L. Zhang, J. Zhou, and Q. Zou, “Using deep learning to solve computer security challenges: A survey,” Cybersecurity, vol. 3, no. 1, 2020.
[12] . Ahmed and M. N. Uddin, “Cyber attack detection method based on NLP and Ensemble Learning Approach,” 2020 23rd International Conference on Computer and Information Technology (ICCIT), 2020.
[13] “Software Assurance Reference Dataset”, [Online]. Available: https://samate.nist.gov/SARD/. [Accessed: 15- Mar- 2022].
[14] A. Karpathy, J. Johnson and L. Fei-Fei, ”Visualizing and Understanding Recurrent Networks”, arXiv.org, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.1506.02078.
[15] M. Schuster and K. K. Paliwal, ”Bidirectional recurrent neural networks,” in IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, Nov. 1997, doi: 10.1109/78.650093.
[16] J. Devlin, M. Chang, K. Lee and K. Toutanova, ”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, arXiv.org, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.1810.04805.
[17] Z. Feng et al., ”CodeBERT: A Pre-Trained Model for Programming and Natural Languages”, arXiv.org, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2002.08155. [Accessed: 15- Mar- 2022].