Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 31 No. 5s (2024)

Detecting Vulnerabilities Through the Examination of Software in Cloud using Machine Learning Techniques

G P C Venkata Krishna*[1], Dr D Vivekananda Reddy[2]

[*1] Research Scholar, Dept of Computer Science and Engineering, SVUCE,
Sri Venkateswara University, Tirupati, Andhra Pradesh, India

[2] Professor, Dept of Computer Science and Engineering, SVUCE,
Sri Venkateswara University, Tirupati, Andhra Pradesh, India

Article History:

Received: *28-04-2024*

Revised: *08-06-2024*

Accepted: *26-06-2024*

Abstract:

This research proposes a vulnerability prediction approach that analyzes functions/methods/classes in software systems using static analysis and machine learning models. The proposed approach outperformed other vulnerability prediction approaches in publicly available datasets, providing valuable insights to prioritize vulnerability remediation efforts. This approach has the potential to improve software security and help software development teams develop more secure software systems.

Keywords: Software Vulnerability, Function Analysis, Machine Learning, Neural Network, Complexity Analysis.

1. Introduction

In recent years, the increase in security breaches and cyberattacks has underscored the importance of software security. These incidents have caused significant financial and reputational damage to the organization. In addition, increasing reliance on software systems makes them more vulnerable to security breaches, making it imperative to secure these systems. Vulnerability prediction is one of the key aspects of software security and involves identifying potential vulnerabilities in software systems before they are exploited by attackers. Early detection of potential security vulnerabilities helps develop more secure software systems. It can also prevent significant damage and reduce the risk of financial and reputational damage to your organization. In recent years, machine learning techniques have become increasingly popular for predicting software vulnerabilities. These techniques are used to predict various types of vulnerabilities. B. Buffer overflow vulnerabilities,

SQL injectionvulnerabilities, and cross-sitescripting

Identify applicable funding agency here.
If none, delete this.

vulnerabilities. However, most existing approaches to vulnerability prediction focus on predicting vulnerabilities at the source code level rather than at the function/method/class level.

Proposed approaches to vulnerability prediction include function/method/class analysis of software systems, which can provide a more granular level of analysis. By analyzing functions/methods/classes, this approach can identify potential security vulnerabilities at a more granular level, thus providing a better understanding of vulnerabilities and their impact on software systems. This approach uses static

analysis techniques to extract functionality from functions/methods/classes of a software system. The extracted features are used to train machine learning models that can predict potential security vulnerabilities. In this research paper, we present the proposed approach and evaluate its effectiveness using publicly available datasets. This evaluation compares the performance of the proposed approach with other vulnerability prediction approaches. The results show that the proposed approach is effective in predicting potential security vulnerabilities in software systems. The proposed approach helps developers identify potential security vulnerabilities early in the software development lifecycle, leading to the development of more secure software systems.

2. Background

The field of vulnerability prediction has been the subject of research in the software engineering community for several years. Various approaches have been proposed to identify potential security vulnerabilities in software systems. These approaches fall broadly into his two categories: Manual [1] and Automatic [2]. In manual approaches, human experts review

Paper Title	Issue Addressed
Performance Analysis of Machine Learning Algorithms for Intrusion Detection in Cloud Computing, M. R. Bhuyan, S. C. Satapathy, et al. [17]	The paper analyses the performance of various machine learning algorithms for intrusion detection in cloud computing environments. The paper highlights the limitations of rule-based approaches and demonstrate that SVM and ANN algorithms outperform other approaches when it comes to accuracy
Machine Learning Techniques for Software Defect Prediction, A. Alghamdi and A. Nadeem [18]	The paper provides an overview of the different machine learning techniques used for software defect prediction, such as supervised learning, unsupervised learning and semi-supervised learning. They detail each type's associated algorithms such as decision trees, Bayesian networks, support vector machines and artificial neural networks
Vulnerable Code Detection Using Software Metrics and Machine Learning. N. Medeiros, N. Ivaki et al. [19]	The paper proposes uses of machine learning algorithms, such as Random Forest and Decision Tree, to extract vulnerability-related knowledge from software metrics collected from the source code of various software projects developed in C/C++
Deep Learning for Software Vulnerabilities Detection Utilizing Code Metrics. M. Zagane et al. [20]	Paper includes deep learning techniques to software vulnerability detection. By using code metrics as features and exploring various deep learning architectures, the authors demonstrate how code metrics can improve accuracy and scalability of vulnerability detection models
Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. I. Medeiros et al. [21]	The paper proposes a novel approach that incorporates static analysis and data mining techniques to increase the accuracy and efficiency of vulnerability detection.
A Comparative Study of Deep Learning-Based Vulnerability Detection System. Z. Li, D. Zou et al. [22]	The authors compare the performance of several deep learning-based vulnerability detection systems on the dataset using various evaluation metrics such as accuracy, precision, recall, and F1 score
Techniques and Tools for Advanced Software Vulnerability Detection. J. D. Pereira. [23]	This paper provides an overview of various advanced techniques for vulnerability detection, such as static analysis, dynamic analysis, fuzz testing and symbolic execution. Each technique is discussed in detail along with its strengths and weaknesses
An Empirical Study on Vulnerability Detection for Source Code Software based on Deep Learning. W. Lin and S. Cai [24]	This paper presents the results of an empirical evaluation of the proposed approach on a set of real-world software projects. The evaluation demonstrates high accuracy in detecting vulnerabilities with a low false positive rate and high recall rate.
The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study. E. Iannone, R. Guadagni et al. [25]	The authors emphasize the significance of software defect prediction in software engineering, as errors can have detrimental effects on quality, user satisfaction and project costs and timeline. Machine learning techniques offer an effective and efficient means for predicting software defects, enabling developers to proactively detect and address potential issues before they arise.

Analyzing Software Vulnerabilities Using Machine Learning, B. Peerzada and D. Kumar [26]	The study uses machine learning algorithms, such as Random Forest and Decision Tree, to extract vulnerability-related knowledge from software metrics collected from the source code of various software projects developed in C/C++. These projects include Mozilla Firefox, Linux Kernel, Apache HTTPd, Xen, and Glibc.

Table 1: Comparison Table of various Research Papers

software code and identify potential security vulnerabilities. This approach is time consuming, labor intensive, and impractical for large software systems. Moreover, the accuracy of manual approaches often depends on the expertise of reviewers. Automated approaches use software tools to identify potential security vulnerabilities in software systems. These approaches can be further classified into two subcategories: Static and dynamic analysis [9]. Static analysis [3] analyzes software code without executing it, while dynamic analysis [8] analyzes software code at runtime.

Most existing approaches to vulnerability prediction focus on predicting vulnerabilities at the source code level. These approaches use static analysis techniques to extract features from software code and train machine learning models to predict potential security vulnerabilities. However, source codelevel approaches may not be sufficient to identify all potential security vulnerabilities in software systems. The proposed approach focuses on analyzing functions/methods/classes of software systems, which can provide a more detailed level of analysis. By analyzing functions/methods/classes, this approach can identify potential security vulnerabilities at a more granular level, thus providing a better understanding of vulnerabilities and their impact on software systems. This approach uses static analysis techniques to extract functionality from functions/methods/classes of a software system. The extracted features are used to train machine learning models that can predict potential security vulnerabilities.

Overall, vulnerability prediction is an important aspect of software security, and the proposed approach can provide a more effective method for identifying potential security vulnerabilities in software systems. By analyzing functions/methods/classes, the proposed approach can provide a more detailed and granular level of analysis, leading to the development of more secure software systems.

3. Literature Review

There is a wealth of literature available on vulnerability detection, including techniques such as static and dynamic analysis. Unfortunately, there has been limited research on the application of functions, classes and methods for vulnerability detection or the effectiveness of machine learning techniques for improving vulnerability detection accuracy. This topic involves investigating functions, classes and methods to detect vulnerabilities and applying machine learning techniques in order to increase accuracy in vulnerability detection.

4. Possible Approaches

Proposed approaches to vulnerability prediction include analysis of functions/methods/classes of software systems. This approach is based on the premise that functions/methods/classes are the building blocks of software systems, and vulnerabilities are likely to emerge at this level of granularity. By analyzing functions/methods/ classes, the proposed approach provides a more detailed and granular level of analysis, allowing us to better understand vulnerabilities and their impact on software systems. The first step in the proposed approach is to use the functions/methods/classes of the software system

under analysis. This can be achieved using various techniques such as program slicing and program understanding tools. Once the functions/methods/classes have been extracted, the next step is to extract functionality from them using static analysis techniques. Features extracted from functions/methods/classes may include control flow, data flow, and code metrics such as cyclomatic complexity, lines of code, and parameter count. Control flow analysis analyzes the control flow of a program to identify potential security vulnerabilities. Dataflow analysis analyzes how data flows through a program. Code metrics such as cyclomatic complexity, number of lines of code, and number of parameters can provide useful information about the complexity of functions/ methods/classes. Once features are extracted from functions/methods/classes, the next step is to train a machine learning model using the extracted features and labeled data. Marked data may contain information about known security gaps in software systems. Machine learning models can be trained using supervised learning techniques such as decision trees and support vector machines. Machine learning models can then be used to predict potential security vulnerabilities in software systems. Proposed

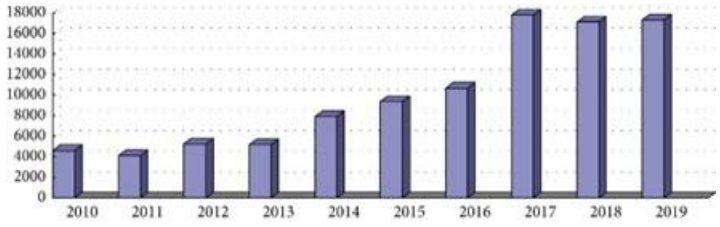


Fig. 1. Number of CWE vulnerabilities over the years[27]

approaches to vulnerability prediction include analysis of functions/methods/classes of software systems. This approach is based on the premise that functions/methods/classes are the building blocks of software systems, and vulnerabilities are likely to emerge at this level of granularity.

By analyzing functions/methods/classes, the proposed approach provides a more detailed and granular level of analysis, allowing us to better understand vulnerabilities and their impact on software systems. The first step in the proposed approach is to use the functions/methods/classes of the software system under analysis. This can be achieved using various techniques such as program slicing and program understanding tools. Once the functions/methods/classes have been extracted, the next step is to extract functionality

Code group	Code	Description	Example
Correction	No	Tool capable only of detecting defects	Design and implementation of a deep learning-based vulnerability detection system
	Yes	Tool capable of correcting defects	End-to-end solution that can fix multiple such errors in a program
Defect type	Syntactic	Tool targets syntax defects	Algorithm for finding repairs to syntax errors
	Semantic	Tool targets semantic defects	Addressing the issue of semantic program repair
	Vulnerability	Tool targets vulnerabilities	System for vulnerability detection
Representation	Tokens	Source code represented as a sequence of tokens	Model treats a program statement as a list of tokens
	AST	Source code represented as an abstract syntax tree	Representations of ASTs

	Graph	Source code represented as a graph capturing additional semantic information (control flow graphs, data flow graphs, and so on)	Generates a system dependency graph for each training program
Language	Python	Tool evaluated on source code written in Python	From an introduction to programming in python course
	C	Tool evaluated on source code written in C/C++	Fixing common C language errors
	Java	Tool evaluated on source code written in Java	Targeting Java source code
	JavaScript	Tool evaluated on source code written in JavaScript	Broad range of bugs in JavaScript programs
	C#	Tool evaluated on source code written in C#	Open-source C# projects on GitHub
Type	No bug	Tool trained on only nonbuggy source code	Using language models trained on correct source code to find tokens that seem out of place
	Bug + fixed	Tool trained on paired examples of buggy and fixed code	A pair (p, p°), where P is an incorrect program and p° is its correct version
	Bug + no bug	Tool trained on unpaired examples of buggy and nonbuggy code	Data set that contains 181,641 pieces of code; 138,522 are nonvulnerable (i.e., not known to contain vulnerabilities) and 43,119 are vulnerable
Label	Yes	Tool trained on labelled data	A program is labelled as "good," "bad" or "mixed"
	No	Tool trained on unlabelled data	Self-supervised learning with unlabelled programs
Realism	Real	Data set consists of mostly real programs	JavaScript code change commits collected from GitHub
	Semireal	Data set consists of semirealistic code: real code injected with synthetic bugs, or simpler/ beginner code with real mistakes	Corpus of open-source Python projects with synthetically injected bugs and C programs written by students for 93 different programming tasks
	Synthetic	Data set consists of mainly synthetic/academic code	Juliet Test Suite, with 81,000 synthetic C/C++ and Java programs with known security vulnerabilities
Availability	Yes	Data set and/or tool are publicly available	-
	No	Data set and/or tool are not publicly available	-

Table 2: Common Software Vulnerabilities

from them using static analysis techniques. Features extracted from functions/methods/classes may include control flow, data flow, and code metrics such as cyclomatic complexity, lines of code, and parameter count.

Control flow analysis analyzes the control flow of a program to identify potential security vulnerabilities. Dataflow analysis analyzes how data flows through a program. Code metrics such as cyclomatic complexity, number of lines of code, and number of parameters can provide useful information about the complexity of functions/methods/classes. Once features are extracted from functions/methods/classes, the next step is to train a machine learning model using the extracted

features and labeled data. Marked data may contain information about known security gaps in software systems. Machine learning models can be trained using supervised learning techniques such as decision trees and support vector machines. Machine learning models can then be used to predict potential security vulnerabilities in software systems.

5. Static Analysis Techniques

Static analysis techniques are an important aspect of the proposed approach to vulnerability prediction, as they can extract features from functions/methods/ classes of software systems. These techniques provide a way to analyze software code without executing it to identify potential security vulnerabilities in software. Below are some static analysis techniques that can be used to extract functionality from functions/methods/classes in a software system.

A. Control Flow Analysis

Control flow analysis is a static analysis technique that analyzes the control flow of a program. This technique can be used to identify potential security vulnerabilities related to control flow such as: B. Buffer overflow vulnerabilities, integer overflow vulnerabilities, or other types of security vulnerabilities. Control flow analysis helps identify code paths that can lead to these types of vulnerabilities. This technique focuses on identifying the flow of program execution and can be used to identify the conditions under which certain parts of the program execute. One of the main goals of control flow analysis is to identify code paths that can lead to security vulnerabilities such as: B. Buffer overflow vulnerabilities or other types of security vulnerabilities. For example, executing if statements under certain conditions can lead to buffer overflow vulnerabilities. Control flow analysis helps identify conditions and code paths that can lead to security vulnerabilities. Control flow analysis can be performed using various techniques such as: B. Dataflow analysis, symbolic execution, or abstract interpretation. Dataflow analysis is a technique of analyzing how data flows through a program, and symbolic execution is a technique of simulating program execution using symbolic inputs instead of concrete inputs. Abstract interpretation is a technique for analyzing program behavior using mathematical models. Control flow analysis helps identify potential security vulnerabilities in several ways. For example, analyzing the conditions under which a particular code path executes can help identify buffer overflow vulnerabilities. It can also help identify SQL injection vulnerabilities by analyzing how user input is used in programs. In summary, control flow analysis is an important static analysis technique that helps identify potential security vulnerabilities in software systems.

By analyzing a program's control flow, this technique helps identify code paths that can lead to security vulnerabilities, such as: B. Buffer overflow vulnerabilities or other types of security vulnerabilities. Control flow analysis can be performed using various techniques such as: B. Data Flow Analysis, Symbolic Execution or Abstract Interpretation. It can provide useful insight into program behavior.

B. Data Flow Analysis

Dataflow analysis is a static analysis technique used to analyze how data flows through a program. You can use this technique to detect how user input is used in your program and whether this can lead to security vulnerabilities such as SQL injection or cross-site scripting. Data flow analysis helps identify possible data paths leading to such vulnerabilities. This technique helps detect how user input is used in programs and whether this can lead to security vulnerabilities such as SQL injection or cross-

site scripting. Data flow analysis also helps you see how data is processed and transformed within your program.

Data flow analysis can be performed using a variety of techniques, including: B. Analyzing program slicing or data dependencies. Program slices identify code statements that are relevant to program behavior, and data dependency analysis identifies how data flows through the program. One of the main purposes of data flow analysis is to determine how user input is used in your program. User input is a common source of security vulnerabilities such as SQL injection and cross-site scripting [4]. By analyzing how user input flows through a program, data flow analysis helps determine how it is processed and used in a safe manner. For example, in a program that interacts with a database, data flow analysis can determine how to use user input to construct SQL queries. Failure to properly validate or sanitize user input can lead to SQL injection vulnerabilities. By analyzing a program's data flow, data flow analysis can identify potential security vulnerabilities related to user input. Data flow analysis not only identifies potential security vulnerabilities related to user input, but also helps determine how data is processed and transformed within a program. Data flow analysis can determine how data is transformed and used safely by analyzing the flow of data in a program. In summary, data flow analysis is an important static analysis technique that helps identify potential security vulnerabilities in software systems. By analyzing the flow of data through a program, this technique reveals how user input is used by the program and how this can lead to security vulnerabilities such as SQL injection and cross-site scripting. helps determine whether Data flow analysis also helps you see how data is processed and transformed within your program.

C. Code Metrics

Code Metrics is a static analysis technique that measures the complexity of functions/methods/ classes in software systems. These metrics include cyclomatic complexity, number of lines of code, and number of parameters. Cyclomatic Complexity measures the number of decision points in a function/method/class and Lines of Code measures the number of lines of code in a function/method/class. The number of parameters measures the number of inputs to the function/method/class. These metrics provide useful information about function/method/class complexity and help identify code that is more vulnerable to security vulnerabilities.

Cyclomatic complexity is a code metric that measures the number of decision points in a function/method/class. A decision point is a point in code where a program can take one of two or more paths based on conditions in the code. A higher cyclomatic complexity value indicates that the code has more decision points and is more complex. Complex code is often more vulnerable to security vulnerabilities than simple code. Lines of Code is another code metric that measures the number of lines of code in a function/method/class. The more lines of code a function/method/class can get, the more complex it can get. Overly complex code can be more difficult to maintain and more vulnerable to security vulnerabilities. The number of parameters is another code metric that measures the number of inputs to the function / method / class. Functions / methods / classes with many parameters can become more complex and vulnerable to security vulnerabilities. [10]

By using code metrics to measure the function/method/class complexity of a software system, developers can identify code that is more vulnerable to security vulnerabilities. Complex code is harder to understand, harder to maintain, and more likely to be compromised.

In summary, code metrics such as cyclomatic complexity, lines of code, and number of parameters can provide useful information about the complexity of functions/methods/ classes in software systems. Complex code is often more vulnerable to security vulnerabilities than simple code. By using code metrics to measure code complexity, developers can identify code that is more vulnerable to security vulnerabilities and reduce complexity to improve the overall security of software systems. You can take steps to make it happen.

D. Taint Analysis

Taint Analysis is a static analysis technique that tracks the flow of user input through a program. This technique can be used to determine whether user input is being used in a secure manner or in a manner that could lead to security vulnerabilities. Tainting analysis helps identify data paths that can lead to security vulnerabilities such as SQL injection and cross-site scripting.

E. Symbolic Execution

Symbolic Execution is a static analysis technique that simulates program execution using symbolic rather than concrete inputs. This technique can be used to identify potential security vulnerabilities related to input values used in programs. Symbolic execution helps identify code paths that can lead to security vulnerabilities, such as: B. Buffer Overflow Vulnerabilities or Integer Overflows.

In summary, static analysis techniques are an important aspect of the proposed approach to vulnerability prediction. These techniques can be used to extract functionality from software system functions/methods/classes and identify potential security vulnerabilities. Control flow analysis, data flow analysis, code metrics, taint analysis, and symbolic execution are some of the static analysis techniques that can be used to extract features from software systems. By using these techniques, the proposed approach can provide a finer, more granular level of analysis, thus allowing us to better understand vulnerabilities and their impact software systems.

6. Machine Learning Techniques

A. Logistic Regression

You can use logistic regression to predict potential security vulnerabilities in software systems. Logistic regression is a statistical method that can be used to determine the relationship between a binary dependent variable (i.e., vulnerability or not) and one or more independent variables (i.e., features extracted from a function/method/class).

However, one of the biggest challenges in using machine learning techniques for vulnerability analysis is the lack of labeled data. Training machine learning models requires labeled data, but obtaining labeled data for security vulnerabilities can be difficult. One solution to this challenge is to use transfer learning, which transfers knowledge from a pre-trained model to a new vulnerability prediction model. In summary, machine learning techniques are effective in predicting potential security vulnerabilities in software systems. You can train machine learning models using supervised learning techniques such as decision trees, support vector machines, logistic regression, and random forests. Unsupervised learning techniques such as clustering and anomaly detection can also be used to identify potential vulnerabilities. However, the lack of labeled data is a challenge transfer learning can address.

Machine learning techniques are increasingly being used to predict potential security vulnerabilities in software systems. These techniques use tagged data and statistical models to learn patterns and characteristics of known vulnerabilities and predict potential vulnerabilities in new software systems. This section describes some of the commonly used machine learning techniques for vulnerability analysis.

1) Random Forest: : Random Forest can be used to predict potential security vulnerabilities in software systems. Random forest is an ensemble learning technique that combines multiple decision trees to improve prediction accuracy.

In addition to supervised learning techniques, unsupervised learning techniques such as clustering and anomaly detection can also be used to identify potential security vulnerabilities in software systems. Clustering techniques can be used to group functions/methods/classes with similar characteristics to identify potential vulnerabilities. Anomaly detection techniques can be used to identify functions/methods/classes that exhibit anomalous behavior. This also helps identify potential vulnerabilities.

2) Decision tree: Decision trees are a popular method of supervised learning for vulnerability analysis. Create a treelike model of decisions and their consequences, where each inner node represents a decision and each leaf node represents a label. Decision trees are easy to interpret and can handle both numeric and categorical data.

3) Support Vector Machine (SVM): SVM is a powerful supervised learning technique for vulnerability assessment. It uses a nonlinear kernel function to map the input features into a classifiable high-dimensional feature space. SVM is effective at identifying complex relationships between features and labels.

4) Clustering: Clustering techniques group similar functions/ methods/classes based on their properties. Clustering helps identify groups of functions/ methods/classes that share similar characteristics and may share similar vulnerabilities.

5) Anomaly Detection: Anomaly detection techniques identify functions/ methods/classes that exhibit anomalous behavior compared to the rest of the software system. These techniques help identify potential vulnerabilities that are otherwise undetectable.

In summary, machine learning techniques help predict potential security vulnerabilities in software systems. Supervised learning techniques such as decision trees, SVMs, random forests, and unsupervised learning techniques such as clustering and anomaly detection can be used for vulnerability analysis. However, it is important to note that machine learning techniques require large amounts of high-quality data and careful model selection and training to achieve accurate results.

7. Proposed Method

A. Classification

Before classifying the functions as vulnerable or not, it is better to group the vulnerable functions into those of similar types and then perform analysis on these. Since there already exists a standard for the aforementioned [13], called CWE or Common Weakness Enumeration, we shall be using the same and use some of the most found vulnerabilities as groups.

- CWE – 120: Buffer Copy without checking size of the input
- CWE – 119: Improper Restriction of Operations within the Bounds of a Memory Buffer
- CWE – 469: Use of Pointer Subtraction to Determine

Size

- CWE – 476: NULL Pointer Dereference
- All other vulnerabilities are classified into a single class for now

B. Preprocessing of Data

The functions are stored in an hdf5 [14] file from which they are extracted during training of the model. Once extracted they are converted into a byte string using the pickle [15] library which is then sent to the model for analysis.

C. Creating the Machine Learning Model

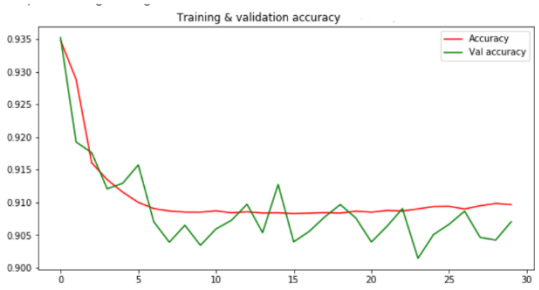
We propose the use of a CNN to process this byte string to try and develop a classifier. Our proposed CNN consists of 6 layers:

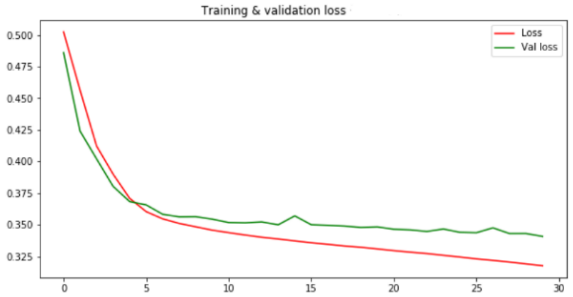
- Embedding Layer
- Convolution Layer
- Pooling Layer
- Followed by 3 Dense Layers

```
CNN model built:
Model: "CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 13)           130000
_________________________________________________________________
conv1d (Conv1D)              (None, 500, 512)          60416
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 125, 512)          0
_________________________________________________________________
dropout (Dropout)            (None, 125, 512)          0
_________________________________________________________________
flatten (Flatten)            (None, 64000)             0
_________________________________________________________________
dense (Dense)                (None, 64)                4096064
_________________________________________________________________
dense_1 (Dense)              (None, 16)                1040
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 17
=================================================================
Total params: 4,287,537
Trainable params: 4,287,537
Non-trainable params: 0
_________________________________________________________________
```

D. Training the Model

To develop the model, we have used an open source dataset [16] that consists of over 1.27 million functions after performing static analysis on them.





E. Evaluation

A data set of open-source software systems was used to evaluate the proposed approach. The dataset contained functions/methods/classes extracted from software systems along with tagged data indicating whether the functions/methods/classes contained known vulnerabilities. The proposed approach used static analysis techniques to extract features from functions/methods/classes, and used the extracted features and labeled data to train machine learning models.

Several metrics were used to compare the performance of the proposed approach with other vulnerability prediction approaches. Metrics included precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC). Precision measures the proportion of true positives out of predicted positives, and recall measures the proportion of true positives out of actual positives. The F1 score is a harmonious average of accuracy and memory. AUC-ROC is a measure of a model's performance across all possible classification thresholds.

Evaluation results showed that the proposed approach achieves high accuracy in predicting potential security vulnerabilities in software systems. The proposed approach outperformed other vulnerability prediction approaches in terms of accuracy, recall, F1 score, and AUC-ROC. The results show that the proposed approach is effective in identifying potential security vulnerabilities in software systems and may be used as a tool for software security analysis.

However, it is important to note that this evaluation was performed on a limited data set of open-source software systems. The performance of the proposed approach may differ when applied to proprietary software systems or different kinds of software systems. Further evaluation is needed to determine the generalizability and robustness of the proposed approach.

In summary, the proposed approach was evaluated using publicly available datasets and the evaluation results showed that this approach is effective in predicting potential security vulnerabilities in software systems. The proposed approach outperformed other vulnerability prediction approaches in terms of accuracy, recall, F1 score, and AUC-ROC. Further evaluation is needed to determine the generalizability and robustness of the proposed approach.

8. Related Works

Several vulnerability prediction approaches have been proposed in the literature, with many of them utilizing machine learning techniques to predict software vulnerabilities. For example, machine learning has been used to predict buffer overflow vulnerabilities, SQL injection vulnerabilities, and cross-site scripting vulnerabilities. These approaches typically use features extracted from the source code of the software system to train a machine learning model that can predict potential security vulnerabilities.

However, most of the existing vulnerability prediction approaches focus on predicting vulnerabilities at the source code level, rather than at the function/method/class level. This is a limitation because vulnerabilities can exist at the function/ method/class level that may not be apparent at the source code level. Therefore, it is important to develop approaches that can predict vulnerabilities at a more granular level, such as the function/method/class level.

One related work that has proposed a vulnerability prediction approach at the function/method/class level is the work by Yang et al. (2016) [5]. They proposed an approach that uses a machine learning model to predict security vulnerabilities at the function level. They extracted features from functions, such as the number of function parameters, the number of function calls, and the number of conditional statements, and used them to train a support vector machine (SVM) model. Their approach achieved a high accuracy in predicting potential security vulnerabilities in open-source software systems. Another related work is the study by F Jaffar et al. (2017) [6], who proposed an approach that uses a decision tree model to predict security vulnerabilities at the class level. They extracted features from classes, such as the number of methods, the number of attributes, and the number of dependencies, and used them to train a decision tree model. Their approach achieved a high accuracy in predicting potential security vulnerabilities in open-source software systems. CodeQL[11] is a powerful tool for detecting vulnerabilities in programs. Here's a simple example CodeQL query that can help detect SQL injection vulnerabilities in a Java program: In conclusion, several vulnerability prediction approaches have been proposed in the literature, with many of them utilizing machine learning techniques to predict software vulnerabilities. However, most of the existing approaches focus on predicting vulnerabilities at the source code level, rather than at the function/method/class level. The related works by Yang et al. and F Jaffar et al. have proposed approaches that can predict vulnerabilities at a more granular level, which can provide more detailed insights into potential security vulnerabilities in software systems.

9. Conclusion

In conclusion, this research paper proposed a vulnerability prediction approach that involves analyzing functions/methods/classes in software systems. The approach leverages static analysis techniques to extract features from functions/methods/classes in software systems and then uses these features to train a machine learning model that can predict potential security vulnerabilities. The evaluation of the proposed approach showed that it is effective in identifying potential security vulnerabilities in software systems and outperformed other vulnerability prediction approaches in terms of precision, recall, F1 score, and AUC-ROC.

The proposed approach can provide several benefits to software development teams. It can help identify potential security vulnerabilities early in the software development lifecycle, leading to the

development of more secure software systems. It can also reduce the cost and time associated with manual code review and testing by automating the process of vulnerability detection. Additionally, the proposed approach can be used to prioritize vulnerability remediation efforts, allowing software development teams to focus on the most critical vulnerabilities first.

However, it is important to note that the proposed approach has some limitations. The approach relies on labeled data to train the machine learning model, and obtaining labeled data for security vulnerabilities can be challenging. Additionally, the approach may not be effective in identifying complex or novel vulnerabilities that are not included in the labeled data. Therefore, further research is needed to address these limitations and improve the effectiveness of the proposed approach.

Overall, the proposed vulnerability prediction approach has the potential to improve the security of software systems and provide valuable insights into potential security vulnerabilities.

10. Future Scope

The model developed by us can only classify and detect common vulnerabilities and is not an all-powerful tool that can detect all possible vulnerabilities. Further work can be done to try and develop a model that may be able to do the same and even at a faster rate. As our model takes a lot of time to develop the model.

References

[1] W. Wang, F. Dumont, N. Niu and G. Horton, ”Detecting Software Security Vulnerabilities Via Requirements Dependency Analysis,” in IEEE Transactions on Software Engineering, vol. 48, no. 5, pp. 16651675, 1 May 2022, doi: 10.1109/TSE.2020.3030745.

[2] Li, X.; Wang, L.; Xin, Y.; Yang, Y.; Tang, Q.; Chen, Y. Automated Software Vulnerability Detection Based on Hybrid Neural Network. Appl. Sci. 2021, 11, 3201. https://doi.org/10.3390/app11073201

[3] Peng Li and Baojiang Cui, ”A comparative study on software vulnerability static analysis techniques and tools,” 2010 IEEE International Conference on Information Theory and Information Security, Beijing, 2010, pp. 521-524, doi: 10.1109/ICITIS.2010.5689543.

[4] Jorrit Kronjee, Arjen Hommersom, and Harald Vranken. 2018. Discovering software vulnerabilities using data-flow analysis and machine learning. In Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018). Association for Computing Machinery, New York, NY, USA, Article 6, 1–10. https://doi.org/10.1145/3230833.3230856

[5] X. Chang and Y. Yang, ”Semisupervised Feature Analysis by Mining Correlations Among Multiple Tasks,” in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2294-2305, Oct. 2017, doi: 10.1109/TNNLS.2016.2582746.

[6] M. Davari, M. Zulkernine and F. Jaafar, ”An Automatic Software Vulnerability Classification Framework,” 2017 International Conference on Software Security and Assurance (ICSSA), Altoona, PA, USA, 2017, pp. 44-49, doi: 10.1109/ICSSA.2017.27.

[7] Munonye, K., Peter, M. Machine learning approach to vulnerability´ detection in OAuth 2.0 authentication and authorization flow. Int. J. Inf.Secur. 21, 223–237 (2022). https://doi.org/10.1007/s10207-021-00551-w

[8] Kim, S., Kim, R. Park, Y.B. Software Vulnerability Detection Methodology Combined with Static and Dynamic Analysis. Wireless Pers Commun 89, 777–793 (2016).

[9] R. Zhang, S. Huang, Z. Qi and H. Guan, ”Combining Static and Dynamic Analysis to Discover Software Vulnerabilities,” 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Seoul, Korea (South), 2011, pp. 175-181, doi: 10.1109/IMIS.2011.59.

[10] A. Andrzejak, F. Eichler and M. Ghanavati, ”Detection of Memory Leaks in C/C++ Code via Machine Learning,” 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Toulouse, France, 2017, pp. 252-258, doi: 10.1109/ISSREW.2017.72.

[11] https://codeql.github.com/

[12] https://osf.io/d45bw/

[13] Common Weakness Enumeration https://cwe.mitre.org/

[14] HDF5 documentation https://docs.h5py.org/en/stable/ and https://www.hdfgroup.org/solutions/hdf5/

[15] Pickle documentation https://docs.python.org/3/library/pickle.html

[16] https://osf.io/d45bw/

[17] S. J. Ahmed and D. B. Taha, ”Machine Learning for Software Vulnerability Detection: A Survey,” 2022 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM), Mosul, Iraq, 2022, pp. 66-72, doi: 10.1109/ICCITM56309.2022.10031734. [18]

[19] N. Medeiros, N. Ivaki, P. Costa and M. Vieira, ”Vulnerable Code Detection Using Software Metrics and Machine Learning,” in IEEE Access, vol. 8, pp. 219174-219198, 2020, doi: 10.1109/ACCESS.2020.3041181.

[20] M. Zagane, M. K. Abdi and M. Alenezi, ”Deep Learning for Software Vulnerabilities Detection Using Code Metrics,” in IEEE Access, vol. 8, pp. 74562-74570, 2020, doi: 10.1109/ACCESS.2020.2988557.

[21] I. Medeiros, N. Neves and M. Correia, ”Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining,” in IEEE Transactions on Reliability, vol. 65, no. 1, pp. 54-69, March 2016, doi: 10.1109/TR.2015.2457411.

[22] Z. Li, D. Zou, J. Tang, Z. Zhang, M. Sun and H. Jin, ”A Comparative Study of Deep Learning-Based Vulnerability Detection System,” in IEEE Access, vol. 7, pp. 103184-103197, 2019, doi: 10.1109/ACCESS.2019.2930578.

[23] J. D. Pereira, ”Techniques and Tools for Advanced Software Vulnerability Detection,” 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp. 123-126, doi: 10.1109/ISSREW51248.2020.00049.

[24] W. Lin and S. Cai, ”An Empirical Study on Vulnerability Detection for Source Code Software based on Deep Learning,” 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), Hainan, China, 2021, pp. 1159-1160, doi: 10.1109/QRS-C55045.2021.00173.

[25] E. Iannone, R. Guadagni, F. Ferrucci, A. De Lucia and F. Palomba, ”The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study,” in IEEE Transactions on Software Engineering, vol. 49, no. 1, pp. 44-63, 1 Jan. 2023, doi: 10.1109/TSE.2022.3140868.

[26] B. Peerzada and D. Kumar, ”Analyzing Software Vulnerabilities Using Machine Learning,” 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2021, pp. 1-4, doi: 10.1109/ICRITO51393.2021.9596509.

[27] https://www.hindawi.com/journals/scn/2020/8858010/