# Source Code Vulnerability Detection Method with Multidimensional Representation

Hongyu Yang[1,2(✉)], Leyi Ying[2], and Liang Zhang[3]

[1] School of Safety Science and Engineering, Civil Aviation University of China, Tianjin 300300, China
[2] School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China
[3] School of Information, University of Arizona, Tucson, AZ 85721, USA

**Abstract.** At present, most of the source code vulnerability detection methods only rely on the source code text information for representation, and the single dimension of representation leads to low efficiency. This paper presents a source code vulnerability detection method based on multidimensional representation. Firstly, the structured text information of the source code is obtained through the abstract syntax tree of the source code; Then the source code is measured to obtain the code metrics; Finally, a deep neural network is used for feature learning to construct the source code vulnerability detection model, and the structured text features and code metrics of the source code to be detected are input into the vulnerability detection model to obtain the vulnerability detection results. The results of the comparison experiment show that the method has a good detection effect. In comparison experiments, 11 source code samples with different types of vulnerabilities were tested for vulnerability detection. The average detection accuracy of this method is 97.96%. Compared with existing vulnerability detection methods based on a single characterization, the detection accuracy of this method is improved by 4.89%–12.21%. At the same time, the miss and false-positive rates of this method are kept within 10%.

**Keywords:** Vulnerability detection · Structured representation · Abstract syntax tree · Code metrics · Deep neural network

## 1 Introduction

With the wide application of computer software in various fields, the problem of software vulnerability has become increasingly serious. Faced with a variety of software vulnerability types, how to efficiently detect vulnerabilities has become a hot issue. Vulnerability detection of source code is one of the effective means to ensure software security. At present, the methods based on code metrics and deep learning are more common source code vulnerability detection methods [1].

Code metrics [2] is a way to describe the characteristics and indicators of software code, that is, to obtain relevant defined values from software code by quantitative method.


W. Shi et al. (Eds.): SPNCE 2021, LNICST 423, pp. 132–139, 2022.
https://doi.org/10.1007/978-3-030-96791-8_10

Although code measurement is a coarse-grained representation of source code, it can represent the basic status of code to a certain extent. The vulnerability detection method based on code metrics measures the target code by using the source code metrics tool to obtain the corresponding index values and uses the machine learning algorithm to train and generate the vulnerability detector. Feren et al. [3] build a vulnerability detection model based on code metrics using machine learning algorithm and grid search algorithm and use resampling strategy to solve the imbalance of training data. Sultana [4] uses machine learning and statistical methods to track the relationship between code metrics, code patterns, and vulnerabilities, and proposes a vulnerability detection method, which is used to detect vulnerabilities in open-source software. The main drawbacks of code Metrics-based vulnerability detection methods are 1. Coarse detection granularity and poor interpretability; 2. Low precision and high false-positive rate.

With the application of deep learning technology in the field of natural language processing, researchers have focused on programming languages that share common characteristics with natural languages. Li et al. [5] introduced deep learning technology into the field of vulnerability detection for the first time and proposed a Vuldeepecker automatic vulnerability detection system, which can detect the vulnerability of source code written in C/C++ language. Nicholas et al. [6] proposed Achilles vulnerability detection method, tested on Java source code, and achieved good results, indicating that the source code vulnerability detection method based on deep learning can be applied to a variety of programming languages. The above two methods treat the source code as linear text and cannot fully represent the characteristics of the source code. To fully represent the syntax and semantics of programming language, structured representation is applied to the representation of source code. Chen Zhaoxuan et al. [7] proposed an intelligent vulnerability detection system Astor based on structural representation. The detection effect is better than the linear representation method on complex and syntax-rich data sets. The main shortcomings of the vulnerability detection method based on deep learning are as follows: 1. Need to rely on a large number of data for training; 2. The detection results of different types of vulnerabilities fluctuate greatly; 3. The precision rate and recall rate need to be improved.

To further improve the effectiveness of vulnerability detection, this paper presents a source code vulnerability detection method based on multidimensional representation to detect vulnerabilities in source code at function-level granularity. First, Structured text information is obtained by deep-first traversing the source code abstract syntax tree, then code metrics are obtained by using the source code static parsing tool. Finally, a neural network is constructed to learn the features and construct the source code vulnerability detection model.

## 2 Design of Source Code Vulnerability Detection Method

### 2.1 Method Architecture Design

The proposed source code vulnerability detection model consists of four parts: data preprocessing, data multidimensional representation, model building and training, and vulnerability detection. The core framework of the method is shown in Fig. 1. The main processes for the four parts of the vulnerability detection model are:

1. Data preprocessing: The data preprocessing phase consists of code slicing and setting supervised learning labels. This method detects vulnerabilities at the function-level granularity, so the source code data is divided into function fragments and tagged according to whether there are vulnerabilities in the function fragments.
2. Data Multidimensional Representation: To fully represent the information of function fragments, the pre-processed data is characterized from two dimensions, structured text information and code metrics. The Abstract Syntax Tree (AST) is used to characterize the text information of function fragments Define code metrics to measure function fragments.
3. Model construction and training: A neural network is constructed, and the neural network performs feature learning according to the data types of two dimensions. The neural network is trained by two kinds of representation results and preset tags to construct a vulnerability detection model.
4. Source code vulnerability detection: Use the training completed vulnerability detection model to detect vulnerabilities in the source code to be detected. The source code to be detected is pre-processed and represented in the same way as the training data, and the vulnerability detection results are obtained by entering the characterization results into the training completed vulnerability detection model.

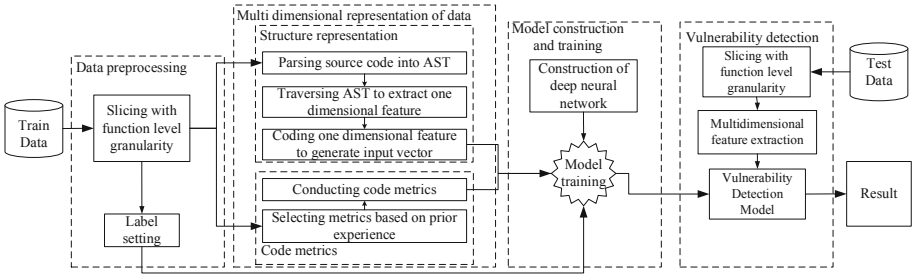


**Fig. 1.** Framework for vulnerability detection in this paper

## 3 Data Representation

To fully represent the source code characteristics, the source code is characterized from two different dimensions: code structure representation and code metrics. Code structure characterization can obtain text information of code structure, and code metrics can characterize the basic state of code.

### 3.1 Code Structural Representation

The programming language is a structured language, and the information in the source code has a clear structural relationship. Therefore, the method of characterizing natural language does not fully characterize the grammar and semantics in the source code.

To get more practical source code characteristics, the source code is characterized by a structured representation method. The structured representation method consists of the following three steps.

Step 1: Use the Java source code parsing tool javalang to parse the code, get the information of the node and edge of the AST, and generate the AST based on the information of the node and edge.
Step 2: depth-first traverses the abstract syntax tree to collect node information in turn. The result of depth-first traversing the abstract syntax tree transforms the tree data into one-dimensional text data.
Step 3: Transform one-dimensional text data into neural network input. Since the input of the neural network is vector data, it is necessary to further process one-dimensional text data. Firstly, the text data is segmented, and then the word in all the text is counted to generate a document dictionary. Finally, the vector representation of the text is generated based on the dictionary order.

### 3.2 Code Metrics

The purpose of this method is to detect the source code vulnerability at the function level, so we need to measure it at the code function level. To make the data-dependent deep learning method interact with the prior knowledge of security experts effectively and make the detection model more adaptive, it is necessary to define the code metrics manually. The code measurement processing in this method consists of two steps:

Step 1: define metrics. This paper defines the metrics of code metrics, and the main metrics used in the code metrics phase are Chidamber & Kemer metrics. Compared with the traditional McCabe metrics and Halstead metrics, chidamber & Kemer metrics are specifically proposed for an object-oriented programming language, so they are more adaptable to Java language.
Step 2: Code metrics. Code metrics using the code metrics tool yields the specific quantified value of the metrics.

## 4 Construction and Training of Neural Network Model

### 4.1 Construction of Neural Network Model

The results of multi-dimensional representation of source code are structured text information and the digital sequence generated by code measurement. Therefore, it is necessary to design a neural network for feature learning of structured text information and digital sequence and synthesize the judgment results of the two to give the final vulnerability detection results.

The neural network model constructed in this paper consists of three parts: 1. Neural network model based on self-attention (SA) mechanism [13]; 2. Deep neural networks (DNN); 3. Support vector machine (SVM). The main structure of the neural network is shown in Fig. 2.

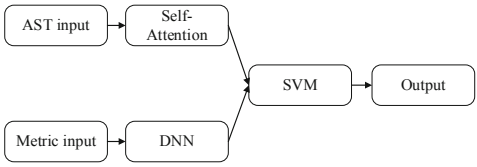


**Fig. 2.** The framework of neural network

### 4.2 Construction of Neural Network Model Based on SA

SA mechanism can reflect the direct interaction between each word and all other words in the document. Comparing long Short-Term Memory (LSTM) or Recurrent Neural Network (RNN) needs to be calculated step by step in the sequence to obtain the long-distance interdependence in the text information. The SA mechanism captures the long-distance dependencies of text information better. Therefore, SA is more suitable for structured text feature learning tasks than traditional RNN or LSTM.

The SA-based neural network constructed in this paper consists of the input layer, SA layer, full connector layer, and output layer, in which the full connector layer consists of 128 neurons. Since the calculations in SA are all linear, the function of adding a fully connected layer is to improve the fitting ability of the nonlinear characteristics of the neural network. To obtain the probability of vulnerabilities from text features, the Sigmoid activation function is applied to the output of the full connection layer.

As an activation function, the Sigmoid function maps the output of the neural network to [0, 1]. Therefore, in the text sequence feature learning phase, the learned text features are transformed into the probability of vulnerability. The SA-based neural network is trained by the results of source code structured representation and preset labels. By entering the source code structured text information into the trained neural network, the probability of vulnerabilities in the corresponding source code can be output.

### 4.3 Construction of DNN Model

The result of a code metrics is a sequence of numbers in which each element represents the specific value of the corresponding measure and there is no interdependence between the elements of the measure. Based on the above application scenarios, DNN can learn sequence features in a shorter time than traditional machine learning algorithms. Therefore, DNN is used to learn the characteristics of code metrics.

The DNN constructed in this paper consists of the input layer, two hidden layers, and the output layer. The number of neurons in each hidden layer is 64. For the input code metrics, after fitting the code metrics characteristics through two hidden layers, the output is mapped to [0, 1] using the Sigmoid function as the activation function. DNN is trained with code metrics and preset labels, and the probability of vulnerabilities in the corresponding source code can be output by entering code metrics into the DNN model after training.

### 4.4 Construction of SVM Model

To get more accurate vulnerability detection results, the output results of the two models need to be combined. Therefore, this paper takes the output of the above two models as a feature and further classifies them using Support Vector Machine (SVM) to determine whether there are any vulnerabilities in the code.

There are two main reasons for choosing SVM as the classifier in this stage: 1. SVM works well in classification tasks and its classification ideas are simple and intuitive, and it can draw its decision boundary accurately; 2. The classification method is flexible and can be used for both linear and non-linear classification by adjusting its kernel function.

Conventional SVMs are classified by drawing maximally spaced hyperplanes, but this method cannot be used for non-linear classification. Since the output of SA-based neural networks and DNNs may be linearly inseparable, the kernel function of SVM is set to classify nonlinearly. The SVM model constructed by this method uses three different kernel functions including the linear kernel, polynomial kernel, and Gaussian kernel to classify the output results of SA-based neural network and DNN respectively. In the trained SVM model, the existence probability of vulnerability based on SA neural network and DNN output is input, and the final result of vulnerability detection is output.

## 5 Experimental Design and Result Analysis

To test the source code vulnerability detection performance of this method, we compared the method with Astor [7], Code Metrics based vulnerability detection [3], Achilles [6], VulDeePecker [5] based on linear text representation. In the performance comparison experiment of source code vulnerability detection, the above five detection models were constructed using the TensorFlow framework. The performance indicators of the five models are shown in Table 1.

**Table 1.** Performance comparison of detection methods

| Detection methods | A | P | R | F1 | FPR |
|---|---|---|---|---|---|
| Code Metrics-based | 0.8575 | 0.8574 | 0.5637 | 0.6668 | 0.0332 |
| Astor | 0.9218 | 0.9392 | 0.7776 | 0.8382 | 0.0242 |
| Achilles | 0.9307 | 0.8901 | 0.6986 | 0.7446 | 0.0463 |
| VulDeePecker | 0.9390 | 0.9190 | 0.9590 | 0.9290 | 0.0490 |
| Method of this article | 0.9796 | 0.9728 | 0.9464 | 0.9585 | 0.0092 |

As can be seen from Table 1, the accuracy of vulnerability detection in this method is better than that in the other 4 methods, with a lower rate of miss and false positives. The average accuracy of code Metrics-based methods, Astor, Achilles, and VulDeePecker for detecting different vulnerabilities is 85.75%, 92.18%, 93.07%, and respectively. The average accuracy of this method for detecting different vulnerabilities is 97.96%, which is superior to the other four methods. The recall rate of this method is 94.64%, which

is higher than the other four methods, indicating that the miss rate of this method is the lowest. The false-positive rate of this method is 0.92%, which is lower than that of the other four methods.

This method can achieve good results in vulnerability detection for two reasons: 1. This method characterizes the source code from two dimensions: source code structure text information and code metrics. This method is more comprehensive than the single representation method. 2. Text information feature is an important feature in vulnerability detection. The SA-based neural network built in this paper can better capture the long-term dependencies in text information.

## 6 Conclusion

In order to further improve the accuracy of source code vulnerability detection and reduce the false alarm rate, this paper proposes a source code vulnerability detection method based on multi-dimensional representation. The source code is characterized by code measurement and structured text, and the neural network model is used for feature learning to construct a vulnerability detection model for source code vulnerability detection. The experimental results show that the proposed vulnerability detection method has higher accuracy, lower false positive rate, and false-negative rate.

This method only characterizes the source code from two dimensions, which are not comprehensive enough. The focus of future work is to explore more suitable source code representation methods for vulnerability detection and improve the representation methods to obtain better detection performance.

## References

1. Li, Z., Shao, Y.: A survey of feature selection for vulnerability prediction using feature-based machine learning. In: Proceedings of the 2019 11th International Conference on Machine Learning and Computing, pp. 36–42. Association for Computing Machinery, New York (2019)
2. Kan, S.H.: Metrics and models in software quality engineering. Softw. Guide **4**(9), 1333–1334 (2009)
3. Feren, R., Péter, H.: Challenging machine learning algorithms in predicting vulnerable JavaScript functions. In: 2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. ACM (2019)
4. Sultana, K.Z.: Towards a software vulnerability prediction model using traceable code patterns and software metrics. In: IEEE/ACM International Conference on Automated Software Engineering, pp. 1022–1025. IEEE Computer Society (2017)
5. Li, Z., Zou, D.Q.: VulPecker: an automated vulnerability detection system based on code similarity analysis. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 201–213. Association for Computing Machinery, New York (2016)
6. Saccente, N., Dehlinger, J.: Project achilles: a prototype tool for static method-level vulnerability detection of java source code using a recurrent neural network. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop. ACM (2019)
7. Chen, Z., Zou, D.: Intelligent vulnerability detection system based on abstract syntax tree. J. Cyber Secur. **2020**(4), 1–13 (2020)
8. Common Weakness Enumeration. https://cwe.mitre.org/. Accessed 6 May 2019

9. Liu, Y.: Research and Application of Code Vulnerability Detection Mechanism Based on Machine Learning, University of Electronic Science and Technology (2018)
10. Vaswani, A., Shazeer, N.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)