Research on Smart Contract Vulnerability Detection Technology Based on VCS and Ensemble Learning

Shouhan Wei

Corresponding author: 1289382022@qq.com

Jiangxi University of Science and Technology, China

Abstract. Smart contracts play a crucial role in blockchain technology, but their writing poses risks of vulnerabilities, potentially leading to serious consequences such as financial losses and system crashes. To address this, we propose a smart contract vulnerability detection method based on VCS and ensemble learning. This method first utilizes Vulnerability Candidate Slicing (VCS) technology to extract syntax and semantic features, enhancing detection capabilities. Then, it employs Word2vec, FastText, GloVe, and other embedding models to transform raw inputs into vector representations, capturing more semantic information. Finally, an ensemble learning strategy integrates multiple neural network models to improve detection performance and mitigate the limitations of individual models. Experimental results demonstrate that this method outperforms other advanced tools in the market, providing robust support for ensuring the security and stability of blockchain systems.

Keyword: Smart contract; Vulnerability detection; Deep learning; Ensemble learning

1 Introduction

In recent years, with machine learning becoming a focal point of research, many researchers have attempted to integrate code auditing with machine learning, yielding promising results[1]. For instance, Xing et al. proposed a slice matrix method[2], which slices smart contracts, extracts fine-grained vulnerability features, and combines machine learning for detection, effectively improving audit accuracy and efficiency. Liao et al. introduced a detection model called Soliaudit[3], incorporating machine learning and fuzz testing. They utilized Solidity machine code as learning features, coupled with fuzz testing to detect reentrancy and overflow vulnerabilities, providing a comprehensive perspective on vulnerability detection. Liu et al. proposed an S-gram language model[4], utilizing an N-gram-based language model S-gram to learn statistical patterns of smart contract tokens and capture advanced semantics for predicting potential vulnerabilities. This approach underscores the importance of language modeling and static semantic label sets, offering novel insights for improving vulnerability prediction accuracy. Zhuang et al. presented a contract graph and graph convolutional neural network approach[5], constructing contract graphs, analyzing syntactic and semantic structures, and using graph convolutional neural networks for vulnerability detection, better capturing the complex structures and relationships of contracts, and enhancing audit comprehensiveness. However, one of the challenges faced by these methods is the lack of publicly available datasets, which constrains further in-depth research. Therefore, this paper proposes a deep learning-based smart contract vulnerability detection technique, converting smart contract opcodes into high-level

ICBBEM 2024, March 29-31, Wuhan, People's Republic of China

DOI 10.4108/eai.29-3-2024.2347453

language code for analysis and study.

2 Related work

2.1 Vulnerability Candidate Slicing (VCS)

VCS (Vulnerability Candidate Slicing) is inspired by the concept of region proposal in object detection tasks in the field of image processing and utilizes each region proposal as the minimum granularity to train deep learning models for target detection, as shown in Figure 1. Initially, smart contracts are partitioned into smaller code segments, namely statements. This aids in enhancing the accuracy of vulnerability detection as smaller code units are easier to manage and analyze. In the preprocessing stage, VCS candidates are generated using an approach similar to region proposal in image processing. These VCS are deemed to contain more vulnerability syntax and semantic features, thus enhancing the sensitivity of vulnerability detection. To ensure that VCS contain sufficient information for detecting vulnerabilities, dependency control is conducted through controlling dependency and data dependency relationships. This ensures that VCS not only cover structural features of the code but also encompass information on data flow and control flow. Defining vulnerability features is a crucial step, including transaction amount values associated with constant calls and constant block values associated with blockchain timestamps. These features serve as matching patterns used for identifying reentrancy vulnerabilities and time-dependent vulnerabilities. Based on the matched statements, other statements related to that statement are obtained by tracking data flow and/or control flow. This helps in constructing a more comprehensive VCS, covering potential sources of vulnerabilities. VCS construction is the focus of the entire process, where all relevant statements build the final VCS used for vulnerability detection. This VCS reflects comprehensive information about potential vulnerabilities in the contract. Finally, key points are extracted from the VCS, which may contain syntax and semantic information related to vulnerabilities. This information is utilized for further analysis and detection, enhancing the understanding and discovery of potential vulnerabilities.

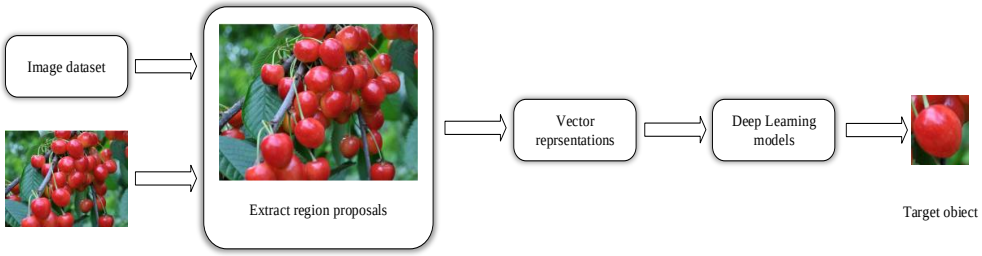

Figure 1. Deep learning for object detection.

2.2 Model Integration

To accurately identify vulnerabilities, neural network models should understand Vulnerability Candidate Slices (VCS) from multiple perspectives in order to comprehensively capture vulnerability patterns. Combining global and local considerations contributes to successfully recognizing fragile contracts, enhancing the model's comprehensiveness and accuracy. While

Convolutional Neural Networks (CNNs)[7] and Recurrent Neural Networks (RNNs)[6] perform well in processing sequential data, they have limitations in learning global dependencies due to restrictions on path length. The computational cost and limitations on path length constrain these models to handle relatively independent paths only, imposing additional restrictions on the model. In contrast, Transformers allow the model to learn potential patterns from a holistic perspective without being restricted by path length, enabling better capture of global dependencies and enhancing the accuracy of vulnerability detection.

CNN, RNN, GRU, and Bi-GRU models transmit information to edges with different weights, and pass the initial vectorized nodes m_1^t and edge lists $[m_1^t, m_2^t, \dots, m_H^t]$ to CNN and other models. For each sub-path, the output of CNN, RNN, and similar models will be the transformed path vulnerability probability M^t, represented as:

$$M^t = ([m_1^{tC}, m_2^{tC}, \dots, m_H^{tC}], [m_1^{tR}, m_2^{tR}, \dots, m_H^{tR}], \dots) \quad (1)$$

Where H represents the number of VCS for each sample. To learn the implicit dependencies of sequences, this method treats all nodes in the overall path of a sample as a sequence and passes it to a transformer to learn the vectorized representation of the overall path $[m_1^l, m_2^l, \dots, m_H^l]$. Removing connecting edges forces the model to more effectively learn long-range dependencies as it no longer relies on the influence of distant node calls. Leveraging multi-head self-attention mechanisms, the model's transformed final node representation contracts the global vulnerability probability M^l, represented as:

$$M^l = [m_1^l, m_2^l, \dots, m_H^l] \quad (2)$$

Then, the result vector representations of each model are fed into a multi-layer perceptron, and this layer is used to assign weights to the models $[n_1^t, n_2^t, \dots, n_H^t]$. For CNN, RNN, and similar models, the computed aggregated score N^t is obtained for each sub-path, represented as:

$$N^t = [n_1^t, n_2^t, \dots, n_H^t] \in R^H \quad (3)$$

For the Transformer, the aggregated score N^l is computed for the entire path, represented as:

$$N^l \in R^{|H|} \quad (4)$$

During the training process, all vulnerability scores are passed through a softmax layer to obtain the vulnerability probabilities for all paths, and cross-entropy loss is calculated.

One of the main objectives of this paper is to improve the accuracy and robustness of a single model by integrating the learning from multiple models. Therefore, the integration structure in this paper adopts five mainstream neural network models, including CNNs, RNNs, GRUs, Bi-GRUs, and Transformers. Such an integration structure design enables the model to learn vulnerability patterns from different perspectives, increasing the comprehensiveness of the model, as shown in Figure 2. Each model has its unique advantages and characteristics, and through integration, their advantages can be utilized comprehensively to enhance the performance of the vulnerability detection system.

To increase the diversity of sub-classifiers, different sub-classifiers are used, each focusing on learning different aspects of vulnerability patterns. This diversity helps improve the model's learning ability because each sub-classifier can capture different features of vulnerability patterns. By introducing sub-classifiers with different structures and algorithms, the model can better adapt to the diversified vulnerability features.

During the inference stage, users can input the vectorized data into the pretrained models to obtain recognition results. Specifically, in this paper, the vectorized code graphs are input into

these five trained models, and then transformed path representations M^t and M^l are output. Subsequently, vulnerability scores N^t and N^l are calculated. The specific formulas are as follows:

$$N^t = \frac{\sum_{j=1}^{H} \frac{N_i^t}{\sum_{i=1}^{H} N_i^t} \cdot N_i^t}{H}; 1 \le i; j \le H \tag{5}$$

Finally, the prediction results are aggregated using a weighted average method to compute the vulnerability score for each sample contract. $N^{en} = 0.5 \times N^t + 0.5 \times N^l$. If the resulting vulnerability score exceeds a threshold of 0.5, the corresponding contract is considered vulnerable. Integrating multiple methods is technically straightforward but is not merely additive; it often yields effective results in practice.[10]

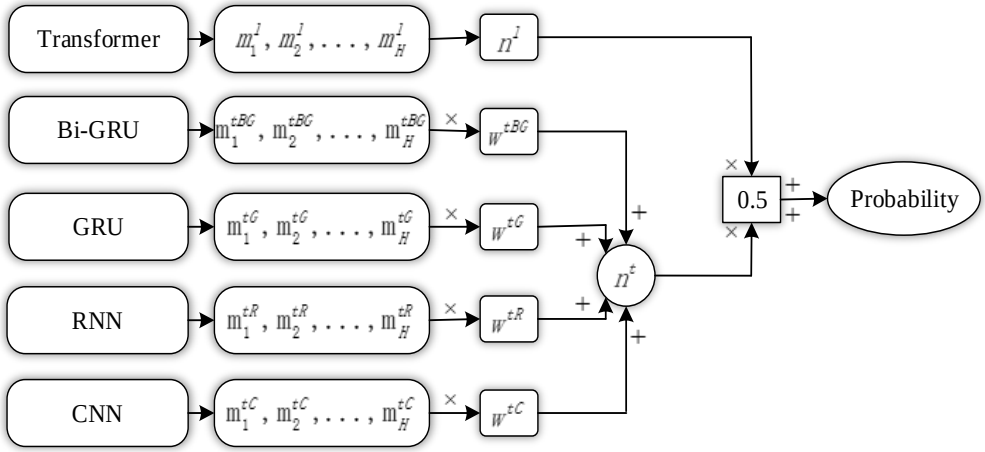

Figure 2. Integrated structure of neural network model.

3 Model Building

In this chapter, the design and implementation of the smart contract vulnerability detection method based on Vulnerability Candidate Slices (VCS) and ensemble learning are introduced, along with the construction of a new dataset. Inspired by the concept of region proposal extraction in object detection tasks for images, this detection method divides smart contracts into smaller code segments (i.e., statements), generating vulnerability candidate slices during the preprocessing stage of the framework. Several popular embedding models, including Word2vec, FastText, and GloVe, are then used to convert raw inputs (such as code tokens) into vectors acceptable by neural network models. Finally, five neural network models are integrated using ensemble learning methods. During the experimental stage, this paper adopts n-fold cross-validation, dividing the acquired smart contract dataset into n mutually exclusive subsets to enhance the performance of deep learning models in vulnerability detection tasks. Figure 3 illustrates the overall design of the proposed method. Both the training and testing stages comprise three main steps: the first step involves the extraction of smart contracts using the VCS method[11], the second step is a modular approach including different types of word embedding

models for code embedding, and the third step involves training or predicting them, and aggregating the results of the five different neural network models to output the final decision.

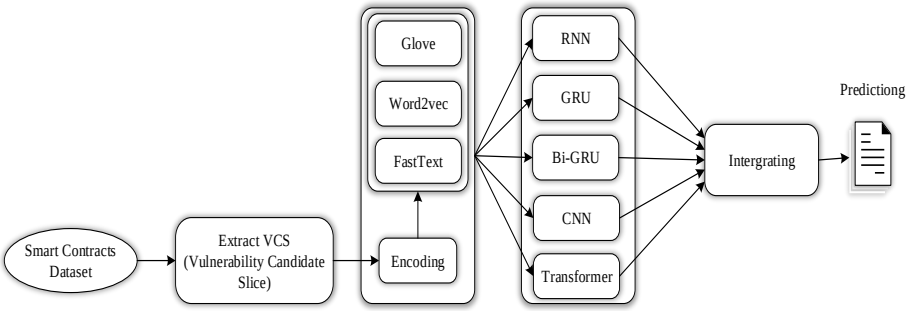

Figure 3. Smart contract vulnerability detection framework of VCS and integrated learning.

4 Experimental Analysis and Comparison

To better utilize the split dataset, this paper divided all data into two parts, with a ratio of 8:2, resulting in two subsets: 80% for training data and 20% for testing data. One of the main objectives of the deep learning-based vulnerability detection tool proposed in this paper is to overcome the limitations of rule-based static analyzers. Four static tools, Oyente, Slither and Smartcheck[12], were used as benchmarks to demonstrate the improvements brought by this approach. Two experiments were designed: error evaluation, vulnerability detection, and comprehensive strategy assessmen[13].

4.1 Error Appraisal

Table 1 presents the results of error evaluation for different models under combinations of vulnerable and non-vulnerable contracts. The experiments employed five-fold cross-validation. Based on these results, an important observation can be made. The combined error evaluation of the method integrating VCS and ensemble learning is 2.118%. This indicates that the method can provide more accurate vulnerability information even with a small training set. Comparing with other individual neural networks such as Transformer+VCS (3.318%) and CNN+VCS (6.288%), it can be observed that combining different knowledge learned by different neural networks helps improve the vulnerability prediction generalization of any single model.

Table 1. Error evaluation.

	RNN+ VCS	CNN+ VCS	GRU+ VCS	Bi-GRU+ VCS	Transformer+ VCS	Ensemble+ VCS
CV1	4.412	10.771	4.992	2.736	1.773	1.972
CV2	3.992	7.411	2.339	2.846	1.973	1.933
CV3	9.876	2.379	13.221	9.172	4.334	1.644
CV4	3.776	7.768	2.205	3.998	3.275	2.035

CV5	4.667	3.112	1.775	7.741	5.239	3.006
Overall	5.344	6.288	4.906	5.298	3.318	2.118

4.2 Vulnerability Detection

To determine whether our framework can detect different types of smart contract vulnerabilities, we evaluated five deep learning models in separate datasets for reentrancy and time dependency vulnerabilities. As shown in Table 2, the five deep learning models combined with VCS can detect these vulnerabilities to some extent, but the overall performance of individual deep learning models is poor. Among them, the CNN+VCS model achieved a P-value of 88.89% on the reentrancy vulnerability dataset, significantly higher than other individual models. Additionally, using the baseline method, the F1 scores of all models were around 70% for time dependency vulnerabilities, with the RNN model achieving the highest F1 score of 78.66% and the Transformer+VCS model achieving the highest P-value of 85.53%. In summary, deep models can automatically learn semantic knowledge from the vector representations of Solidity source code to detect different types of vulnerabilities, especially time dependency vulnerabilities. One possible reason is that samples of time dependency vulnerabilities have more distinguishable syntax or semantic features. This result demonstrates the potential application of deep learning in smart contract vulnerability detection, indicating that deep learning models have indeed learned some vulnerability features.

The article also compared the integrated models with individual models, further demonstrating the effectiveness of the five neural network models after integration. From Table 2, it is evident that our integrated model outperforms the baseline method in detecting both types of vulnerabilities[14]. Significant improvements were observed across all metrics: in detecting reentrancy vulnerabilities, the integrated model showed an increase of 0.77% in F1 score and 0.24% in P-value. In detecting time dependency vulnerabilities, the integrated model exhibited an average increase of 0.24% in F1 score and 1.46% in P-value.

Table 2. Results evaluation table of four neural network models and integrated network models.

Vulnerability type		RNN+ VCS	CNN+ VCS	GRU+ VCS	Transformer+ VCS	Ensemble+ VCS
Reentrant vulnerability	P%	69.23	88.89	70.23	75.19	89.99
	F1%	69.08	72.73	72.58	75.66	76.43
Time-dependent vulnerability	P%	84.61	65.12	84.45	85.53	85.77
	F1%	78.66	76.35	78.33	73.93	80.12

4.3 Comprehensive Strategy Evaluation

To evaluate the effectiveness of the integrated method, we compared it with state-of-the-art smart contract vulnerability detection methods from two categories. We selected three other

methods, including Smartcheck, Securify, and Oyente, which do not utilize deep learning models[15].

Table 3 presents the test results. Among the three traditional methods without a deep learning phase, Oyenete achieved an F1 score of 54.12% in reentrancy vulnerability detection, while Slither achieved an F1 score of 50.75% in timestamp dependency vulnerability detection, both of which are relatively low. This is because these methods primarily detect these two types of vulnerabilities by rudimentary checks of whether statements contain call.value/block. Compared to methods based solely on graph neural networks, the ensemble learning-based method largely outperformed the aforementioned state-of-the-art methods. In this method, the F1 score for detecting reentrancy vulnerabilities was 74.63%, an improvement of 0.32% over the graph neural network-based method and an improvement of 20.51% over the Oyenete-based method, performing the best among traditional methods. The F1 score for detecting timestamp dependency vulnerabilities was 78.12%, higher than the four aforementioned methods. These findings reveal the significant potential of the VCS-based and ensemble learning-based method for smart contract vulnerability detection.

Table 3 Results compared with the three advanced methods.

Vulnerability type		Smartcheck	Slither	Oyenete	Ensemble+VCS
Reentrant vulnerability	P%	33.72	35.13	54.12	74.63
	F1%	28.31	43.31	51.14	74.65
Time-dependent vulnerability	P%	49.12	50.75	47.23	83.74
	F1%	34.99	40.12	37.52	78.12

5 Conclusion

The article primarily introduces an intelligent contract vulnerability detection technique based on Vulnerability Candidate Slices (VCS) and ensemble learning. This method utilizes VCS technology to capture more syntax and semantic features of vulnerabilities, thereby enhancing the sensitivity of vulnerability detection. Several popular embedding models, including Word2vec, FastText, and GloVe, are employed to convert raw inputs (such as code tokens) into vectors acceptable by neural network models. Finally, ensemble learning is utilized to integrate five neural network models. During the experimental stage, the article adopts n-fold cross-validation, dividing the acquired smart contract dataset into n mutually exclusive subsets to enhance the performance of deep learning models in vulnerability detection tasks. In the analysis of experimental results, the article summarizes and analyzes the results from three aspects: error evaluation, vulnerability detection, and comprehensive strategy evaluation. The conclusion drawn is that this method outperforms other advanced detection tools in detecting reentrant vulnerabilities and timestamp vulnerabilities

References

[1] Cheng, Jieren, et al. "DDoS Attack Detection via Multi-Scale Convolutional Neural Network."

Computers, Materials & Continua 62.3 (2020).
[2] Xing, Cipai, et al. "A new scheme of vulnerability analysis in smart contract with machine learning." Wireless Networks (2020): 1-10.
[3] Liao, Jian-Wei, et al. "Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing." 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). IEEE, 2019.
[4] Liu, Chao, et al. "Reguard: finding reentrancy bugs in smart contracts." Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. 2018.
[5] Zhuang, Yuan, et al. "Smart contract vulnerability detection using graph neural networks." Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021.
[6] Graves, Alex, et al. "A novel connectionist system for unconstrained handwriting recognition." IEEE transactions on pattern analysis and machine intelligence 31.5 (2008): 855-868.
[7] Xie, Lingxi, and Alan Yuille. "Genetic cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.
[8] Jiao, Zhenyu, Shuqi Sun, and Ke Sun. "Chinese lexical analysis with deep bi-gru-crf network." arXiv preprint arXiv:1807.01882 (2018).
[9] Falender, Carol A., et al. "Defining competencies in psychology supervision: A consensus statement." Journal of clinical psychology 60.7 (2004): 771-785.
[10] Ding, Yangruibo, et al. "VELVET: a noVel Ensemble Learning approach to automatically locate VulnErable sTatements." 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2022.
[11] Zhuang, Yuan, et al. "Smart contract vulnerability detection using graph neural networks." Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021.
[12] Fynn, Enrique, Alysson Bessani, and Fernando Pedone. "Smart contracts on the move." 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2020.
[13] So, Sunbeom, et al. "Verismart: A highly precise safety verifier for ethereum smart contracts." 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.
[14] Cheng, Jieren, et al. "DDoS Attack Detection via Multi-Scale Convolutional Neural Network." Computers, Materials & Continua 62.3 (2020).
[15] Fynn, Enrique, Alysson Bessani, and Fernando Pedone. "Smart contracts on the move." 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE, 2020.