Journal of Computer Science and Frontier Technologies
ISSN Print: 3104-4204
ISSN Online: 3104-4212
CD ISSN: 3104-4220
https://doi.org/10.63313/JCSFT.9028

Vulnerability Detection of Blockchain Smart Contracts Based on GNN with Multi-Head Attention Mechanism

Xin Du
Xi`an Shiyou University, Xi`an, 710065, China
Email: 2281220299@qq.com

How to cite this paper: Du, X. (2025). Vulnerability Detection of Blockchain Smart Contracts Based on GNN with Multi-Head Attention Mechanism. *Journal of Computer Science and Frontier Technologies*, 2(1), 1–8. Print ISSN: 3104-4204; Online ISSN: 3104-4212.
https://doi.org/10.63313/JCSFT.9028
Published: 2025-12-05

Abstract

Smart contracts have been widely applied in various fields. Due to the immutability of data on the blockchain, it is of great significance to conduct smart contract vulnerability detection before data is uploaded to the chain. To address the problems of low accuracy and single vulnerability type in traditional detection methods, a blockchain smart contract vulnerability detection method based on Graph Neural Network (GNN) is proposed. This method abstracts the functions and key code segments in smart contracts into nodes in a graph, and constructs edges by leveraging data and control dependencies during code execution, thereby accurately depicting the specific graph structures of reentrancy attacks and timestamp-dependent vulnerabilities. To further enhance the model's sensitivity to key vulnerability patterns, the multi-head attention mechanism is innovatively introduced, which can effectively screen out the nodes and edges that contribute the most to vulnerability detection, suppress irrelevant or noisy information, and significantly improve the accuracy and robustness of vulnerability detection. Experimental results show that the proposed method achieves an accuracy of 85.19% in reentrancy vulnerability detection and 82.37% in timestamp-dependent vulnerability detection, demonstrating excellent vulnerability identification capability.

Keywords

Blockchain; Smart Contract; Vulnerability; Detection; GNN; Insert

1. Introduction

Smart contracts are automated protocols operating in cyberspace, first proposed by

Nick Szabo[1], which aim to use computer systems to automatically execute, verify, and propagate contract contents. Owing to the inherent characteristics of blockchain, data uploaded to the chain is immutable—once data is on-chain, it cannot be modified. Consequently, vulnerabilities in smart contracts can lead to enormous losses, making vulnerability detection of smart contracts before they are deployed on the blockchain of great significance. For instance, in 2016, The DAO[2] smart contract was exploited due to a reentrancy vulnerability, resulting in losses of tens of millions of US dollars and drawing widespread attention from the Ethereum community.

Existing traditional vulnerability detection methods mainly include symbolic execution, fuzz testing, formal verification, and taint analysis. Symbolic execution abstracts variable values into symbolic inputs and analyzes vulnerabilities by combining path constraints. Oyente[3] was the first detection tool to adopt symbolic execution, and Gasper[4] optimized the detection of high gas consumption vulnerabilities based on it, yet both still face the problem of path explosion. Fuzz testing detects vulnerabilities by randomly generating unexpected inputs to test smart contracts. Contract Fuzzer[5] was the first smart contract detection tool based on fuzz testing; subsequent tools such as EOSFuzzer[6] and GasFuzzer[7] have been improved for different platforms and vulnerability types. However, fuzz testing is limited by the accuracy of vulnerability modeling, and increasing coverage does not necessarily improve detection capability. Formal verification uses mathematical methods to verify the security of smart contracts. Bhargavan[8] et al. first formally described Ethereum instructions and conducted verification based on the F* language, while Grishchenko[9] et al. further defined four categories of security properties and performed corresponding verification, though this method is constrained by the complexity of mathematical modeling, which affects detection accuracy. Taint analysis tracks the propagation paths of untrusted data in smart contracts to detect potential security risks. Sereum, proposed by Rodler et al.[10] in 2018, modifies the Ethereum Virtual Machine interpreter to maintain shadow memory for marking tainted data and detects modifications to key variables during execution, thereby identifying reentrancy vulnerabilities.

Deep learning-based vulnerability detection methods automatically learn vulnerability features through data-driven approaches, improving detection accuracy. Tann et al.[11] used Long Short-Term Memory (LSTM) networks to model the opcode sequences of smart contracts and identify vulnerabilities via binary classification. However, LSTM is limited by sequence length and struggles to capture complex contextual information of contracts, which impairs detection accuracy.

Graph Neural Networks (GNNs) are a deep learning framework developed in recent years that can learn structured information in graph data. As a form of program code, smart contract vulnerabilities are often hidden in code logical relationships, and the logical interactions within and between contracts are particularly suitable

for graph-structured representation. To address the limitations of existing schemes, the main research contributions of this paper are as follows: A GNN-based smart contract vulnerability detection method is proposed, which abstracts code into nodes and constructs edges based on data and control flows to build graph structures for reentrancy attacks and timestamp-dependent vulnerabilities. By integrating the multi-head attention mechanism, the model's ability to capture key vulnerability features is enhanced.

2. Construction of Graph Structure for Blockchain Smart Contracts

2.1. Node Generation

When converting blockchain smart contracts into graph structures through the logical representation between code segments using the Graph Convolutional Network (GCN) method, it is necessary to construct node information in a rational manner. To improve the accuracy of vulnerability detection, nodes can be classified according to their importance in the graph structure into three categories: key nodes, secondary nodes, and return nodes. Specific descriptions of the node classification are presented in Table 1.

Table 1. Node Construction Table

Symbol	Meaning	Reentrancy Attack Vulnerability	Timestamp Vulnerability
M-Node	Custom or Built-in Functions	withdraw(uint amount) call.value()	block.timestamp
S-Node	Key Variables	balances[msg.sender]	unlockTime deadline
F-Node	Revert Functions	fallback() external	if(block.timestamp > deadline)

2.2. Edge Generation

The directed edges connecting each node are mainly divided into four types: control flow edges, data flow edges, forward execution edges, and callback execution edges. The specific definitions are presented in Table 2.

Table 2. Edge Construction Table

Symbol	Semantic Fact	Type
AH	Assert{X}	Control Flow Edge
RG	Require{X}	
IR	Revert	
IT	Throw	
IF	If{X}	
GB	If{---}else{---}	
GN	If{---}then{---}	
WH	While{---}do{---}	
FR	For{---}do{---}	
AG	Assign{X}	Data Flow Edge
AC	Acess{X}	

FW	Natural Sequence Relationship	Forward Edge
FB	Interactive Function and Revert Function	Revert Edge

2.3. Graph Construction

Graph structures can better preserve the vulnerability structures in the source code of smart contracts. By abstracting the functions and key code segments in smart contracts into nodes in the graph, and constructing edges using data and control dependencies during code execution, the specific graph structures of reentrancy attacks and timestamp-dependent vulnerabilities can be accurately characterized. Figure 1 shows the constructed graph structure for reentrancy vulnerabilities, and Figure 2 shows the constructed graph structure for timestamp vulnerabilities.

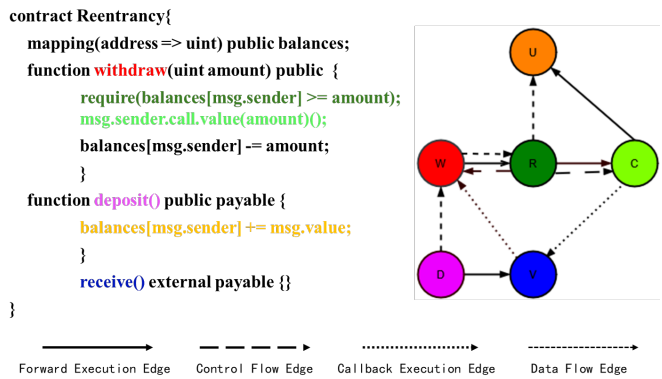

Figure 1. Graph Construction of Smart Contract Reentrancy Vulnerability

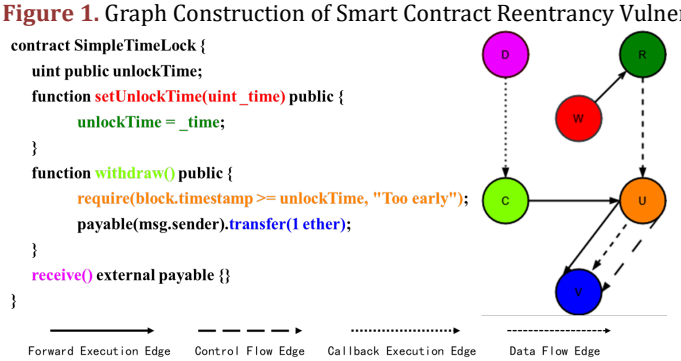

Figure 2. Graph Construction of Smart Contract Timestamp Vulnerability

3. Blockchain Smart Contract Vulnerability Detection Method Based on Graph Neural Networks

The improved model proposed in this paper enhances the feature extraction capability of traditional GCN by introducing the Graph Attention Mechanism (GAT). During the node feature aggregation process, it dynamically calculates the importance weights of different neighbor nodes, enabling the model to more accurately focus on the key code structural features related to vulnerabilities, thus significantly improving the accuracy of vulnerability detection. Figure 3 shows the blockchain smart contract vulnerability detection model based on graph neural networks.

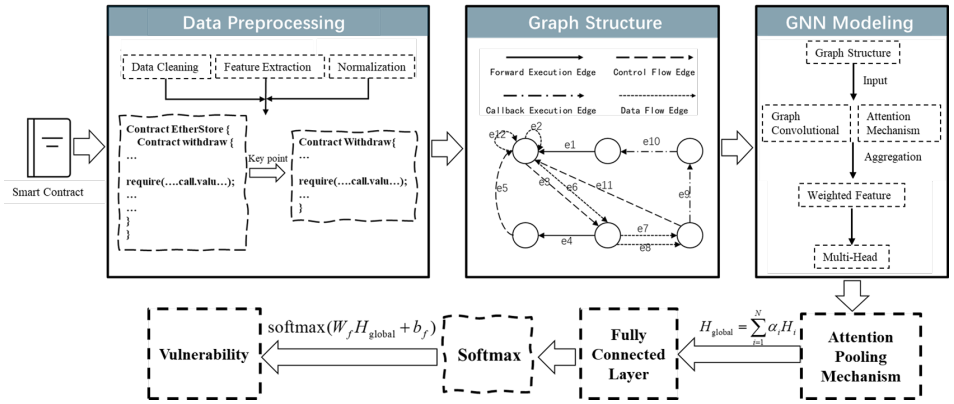

Figure 3. Blockchain Smart Contract Vulnerability Detection Model Based on Graph Neural Networks

3.1. Input Feature Preparation

Before introducing the multi-head attention mechanism into GCN, input features need to be prepared: these include the node feature matrix $H \in \mathbb{R}^{N \times d}$ (where N denotes the number of nodes and d is the feature dimension), the adjacency matrix $A \in \mathbb{R}^{N \times N}$ that represents the graph's topological structure, the degree matrix D used for normalizing the adjacency matrix, and the self-loop matrix $\tilde{A} = A + I$ which adds self-connections to enhance information flow.

GCN adopts the following formula for feature propagation and aggregation:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \quad (1)$$

Among them, $H^{(l)}$ is the input feature matrix of the l-th layer, $W^{(l)}$ is the learnable weight matrix, σ is the activation function (e.g., ReLU), and $\tilde{D}$ is the degree matrix of $\tilde{A}$.

3.2. Integration with Multi-Head Attention Mechanism

On the basis of the features extracted by GCN, the attention mechanism is incorporated to enhance the model's capability of focusing on key features. Linear transformation: The input feature matrix is linearly transformed to generate queries(Q)keys(K)and values(V)as shown in the following formulas.

$$Q_i = HW_Q^{(i)}, \quad K_i = HW_K^{(i)}, \quad V_i = HW_V^{(i)} \quad (2)$$

Among them, $W_Q^{(i)}, W_K^{(i)}, W_V^{(i)}$ are the learnable weights of the i-th attention head.

Calculating attention weights: The scaled dot-product attention is used to compute the attention scores between neighboring nodes, as shown in the following formula.

$$\alpha_{ij}^{(i)} = \frac{\exp\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)}{\sum_{j \in \mathcal{N}(i)} \exp\left(\frac{Q_i K_j^T}{\sqrt{d_k}}\right)} \quad (3)$$

Among them, d_k is the dimension of the attention head, and $\mathcal{N}(i)$ denotes the neighbors of node i.

Calculating weighted features: The weighted aggregated features are computed based on the attention weights α as shown in the following formula.

$$H_i^{(i)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(i)} V_j \quad (4)$$

Multi-head attention fusion: concatenate the outputs of multiple attention heads and integrate information via linear transformation, where $H^{(l+1)}$ is defined as:

$$H^{(l+1)} = \text{Concat}(H^{(1)}, H^{(2)}, ..., H^{(n)}) W_O \quad (5)$$

Among them, W_O denotes the projection matrix for the final output.

3.3. Attention Pooling Mechanism

The calculation of global attention weights is expressed by the following formula: an attention vector W_α is learned to compute the importance score of each node:

$$\alpha_i = \text{softmax}(H_i W_\alpha) \quad (6)$$

To calculate the global feature representation, the attention weights are used to perform weighted aggregation on the features of all nodes. The global feature calculation is expressed as:

$$H_{\text{global}} = \sum_{i=1}^{N} \alpha_i H_i \quad (7)$$

3.4. Output and Vulnerability Detection

Finally, the pooled global feature H_{global} is passed through a fully connected layer and a Softmax classifier to predict vulnerabilities:

$$P = \text{softmax}(W_f H_{\text{global}} + b_f) \quad (8)$$

Among them, W_f and b_f are learnable parameters.

4. Experiments and Analysis

4.1. Experimental Design

In this study, a publicly available smart contract dataset[12]on the Ethereum platform was selected as the experimental object. This dataset contains approximately 1,400 contracts. For model training and validation, the dataset was randomly split on a per-contract basis, with 80% used for training and 20% for testing. The Adam optimizer was adopted in the experiments, and the learning rate was set to 0.001 to improve the convergence efficiency and stability of the model.

4.2. Evaluation Metrics

In this paper, the performance of the model is evaluated using accuracy, recall, precision, and F1-score, and further compared with the results of baseline models.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$Recall = \frac{TP}{TP + FN} \text{u} \quad (10)$$

$$Precision = \frac{TP}{TP + FP} \quad (11)$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (12)$$

In the evaluation metrics, TP denotes the number of vulnerable contracts correctly identified, TN denotes the number of normal contracts correctly identified, FP denotes the number of normal contracts incorrectly labeled as vulnerable, and FN denotes the number of vulnerable contracts that are not identified. Among them, TP + FN corresponds to the actual number of vulnerable contracts, while TP + FP corresponds to the number of vulnerable contracts predicted by the model.

4.3. Experimental Results and Analysis

The method in this section is compared with three deep learning methods, namely LSTM, GRU, and Bi-LSTM, and the evaluation metrics described in Section 3.2 are adopted for the comparative experiments, as shown in Table 3.

Table 3. Comparison of the Performance of Different Models

Method	Reentrancy Vulnerability (%)				Timestamp Vulnerability (%)			
	Acc	RC	Pre	Acc	Acc	RC	Pre	Acc
LSTM	63.7	64.2	61.4	62.7	63.7	64.2	61.4	62.7
GRU	69.5	68.4	66.4	68.4	67.4	66.6	64.4	65.3
BI-LSTM	74.3	74.6	72.6	73.2	71.1	73.4	69.7	70.2
ProposedModel	85.1	81.2	81.5	80.4	80.4	80.2	80.9	81.8

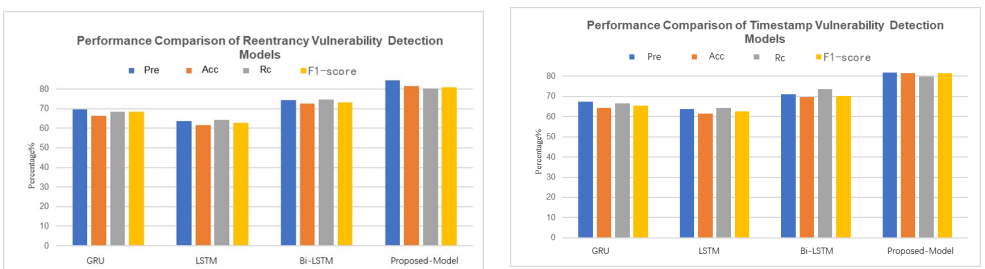

Figure 4. Performance Detection Results of Different Models

The experimental results show that the proposed model achieves the optimal performance in both types of vulnerability detection tasks. This indicates that the model in this paper has stronger feature extraction and generalization capabilities in smart contract vulnerability detection tasks.

5. Conclusion

This paper proposes a GNN-based smart contract vulnerability detection method for blockchain with a multi-head attention mechanism. By converting smart contract code into graph structures and combining the multi-head attention mechanism to enhance feature extraction capabilities, experimental results show that the method proposed in this paper can effectively detect and fix vulnerabilities. Future research can further explore the detection of more types of vulnerabilities to improve its application value in complex contract environments.

References

[1] SZABO N.Formalizing and securing relationships on public networks[J]. First Monday, 1997, 2(9): 1-21.

[2] MEHAR M I,SHIER C L,GIAMBATTISTA A,et al.Understanding a revolutionary and flawed grand experiment in blockchain:the DAO attack[J].Journal of Cases on Information Technology, 2019, 21(1): 19-32.

[3] LUU L,CHU DH, OLICKEL H, et al.Making smart contracts smarter[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna: ACM, 2016: 254-269.

[4] CHEN T, LI XQ, LUO XP, et al. Under-optimized smart contracts devour your money[C]// Proceedings of the 24th IEEE International Conference on Software Analysis, Evolution and Reengineering. Klagenfurt: IEEE,2017:442-446.

[5] JIANG B, LIU Y, CHAN WK. ContractFuzzer: Fuzzing smart contracts for vulnerability detection[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier: IEEE, 2018: 259-269.

[6] HUANG YH, JIANG B, CHAN WK. EOSFuzzer: Fuzzing EOSIO smart contracts for vulnerability detection[EB/OL]. arXiv:2007.14903, 2020. (2020-07-29)[2024-07-20].

[7] ASHRAF I, MA XX, JIANG B, et al. GasFuzzer: Fuzzing Ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities[J]. IEEE Access, 2020, 8: 99552-99564.

[8] BHARGAVAN K, DELIGNAT-LAVAUD A, FOURNET C, et al. Formal verification of smart contracts: Short paper[C]// Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. Vienna: ACM, 2016: 91-96.

[9] GRISHCHENKO I, MAFFEI M, SCHNEIDEWIND C. A semantic framework for the security analysis of Ethereum smart contracts[C]// Proceedings of the 7th International Conference on Principles of Security and Trust. Thessaloniki: Springer, 2018: 243-269.

[10] RODLER M, LI WT, KARAME GO, et al. Sereum: Protecting existing smart contracts against re-entrancy attacks[EB/OL]. arXiv:1812.05934, 2018. (2018-12-14)[2024-07-20].

[11] TANN WJW, HAN XJ, GUPTA SS, et al. Towards safer smart contracts: A sequence learning approach to detecting security threats[EB/OL]. arXiv:1811.06632, 2019. (2019-11-16)[2024-07-20].

[12] Zhuang Y , Liu Z , Qian P ,et al. Smart Contract Vulnerability Detection using Graph Neural Network[J].2020.