Online ISSN: 3104-9338
Print ISSN: 3104-932X

Research on Fine-grained Detection Method of Honey Pot Contracts Based on LSTM and Fuzzing

Chenran Xi

Taiyuan Normal University

Received: December 27, 2025

Revised: January 18, 2026

Accepted: January 19, 2026

Published online: January 28, 2026

To appear in: *International Journal of Advanced AI Applications,* Vol. 2, No. 2 (February 2026)

* Corresponding Author: Chenran Xi (1215819301@qq.com)

Abstract. Honey pot contract operation code sequences exhibit strong concealment, significantly increasing detection complexity. To address this, this study proposes a fine-grained detection method based on LSTM and Fuzzing. By analyzing frequency differences across operation codes in different honey pot contract types, we calculate their occurrence rates and assign high initial weights to high-frequency operation codes. The weight mechanism is then integrated into the LSTM model to calculate operational code contribution levels and importance scores, enabling extraction of high-scoring critical operation codes. The research employs Fuzzing fuzz testing technology to generate initial test case sets and defines their deconstruction methods. Using case identifiers and functional codes, we validate interaction logic vulnerabilities in honey pot contracts through mutation factor probability matrices. By constructing source code graph structures using critical operation codes and interaction logic vulnerabilities, we update and aggregate vector nodes with global accumulation pooling functions to generate graph-level vectors. These graph-level vectors are then fed into graph attention networks, with cross-entropy loss functions jointly determining honey pot contract types. Test results demonstrate that the proposed method achieves sub-3 false positives for six honey pot contract types, demonstrating high precision in fine-grained detection.

Keywords: *LSTM Model; Fuzzing Testing; Smart Contract Honeypot; Fine-grained Detection*

1. Introduction

Honeypot contracts, a novel type of smart contract emerging in recent years, differ from traditional vulnerability contracts and stealth contracts. They employ deceptive tactics like

fabricated funding pools and conditional locking mechanisms to infiltrate target users and devices, ultimately stealing assets or tampering with data, posing significant security risks. Current detection methods primarily rely on control flow matching, analyzing logical trap timing patterns through symbolic code execution and identifying vulnerabilities via state space evolution. However, this approach fails to comprehensively cover attack paths, resulting in high false positive rates. Therefore, there is an urgent need for a high-precision detection method to mitigate honeypot contract attacks.

In current research on contract vulnerability detection, scholars have proposed various methodologies. Specifically, Reference [1] employs entity-relation-entity triplet embedding to extract variable features, combines neural networks with bidirectional long short-term memory networks to model global temporal dependencies, and utilizes SoftMax classifiers for vulnerability classification. While this approach visualizes critical code segments through weight distribution for rapid root cause identification, it struggles with dynamic logic processing and often misses context-sensitive vulnerabilities. Reference [2] constructs program dependency graphs based on contract features, concatenates semantic features via graph convolutional networks for vulnerability classification. This method effectively reduces sample data size while preserving critical code segments and lowering computational complexity. However, its slicing granularity control introduces redundant information that disrupts key dependency chains, thereby increasing detection errors.

Furthermore, most existing research focuses on general vulnerability detection, lacking specialized analysis methods for the unique logical traps and interactive deception mechanisms of honeypot contracts. Honeypot contracts often embed covert malicious logic within normal business processes, making it difficult for traditional static analysis and dynamic execution methods to capture their coordinated attack behaviors across contracts and transactions. Therefore, a hybrid detection framework combining temporal modeling and fuzz testing has become an important direction for improving detection accuracy.

Building on the aforementioned research context, this study employs LSTM and Fuzzing techniques to conduct granular detection of honeypot contracts, thereby providing a security solution with low false positives and high coverage for the blockchain ecosystem.

2. Technical Framework and Research Overview

2.1 Evolution of Smart Contract Security Detection Techniques

The field of smart contract security detection has evolved from early rule-based pattern

matching into a comprehensive system integrating static analysis, dynamic testing, and machine learning. Static analysis methods, such as symbolic execution and formal verification, can systematically traverse the contract state space but face the path explosion problem when dealing with complex control flows and external calls. Dynamic analysis methods, particularly fuzzing, trigger runtime exceptions by generating random or semi-structured inputs, yet their effectiveness heavily depends on the design of initial seeds and mutation strategies. In recent years, data-driven methods represented by deep learning have provided a new paradigm for contract security analysis. These methods can automatically learn vulnerability representation patterns from vast amounts of contract code, significantly enhancing the automation and generalization capabilities of detection.

2.2 Key Advances in Deep Learning for Contract Security Analysis

In the process of applying deep learning to contract security, model architectures have evolved from sequence models to graph neural networks. Sequence models represented by LSTM and BiLSTM can effectively capture long-range dependencies in opcode sequences but have limitations when processing structured semantics across functions and contracts. Graph Neural Networks (GNNs), by abstracting contracts into control flow graphs, data flow graphs, or hybrid graph structures, better preserve the topological semantics of code and have demonstrated excellent performance in detecting vulnerabilities such as reentrancy and improper access control. However, most existing methods treat contracts as static code for analysis and fail to fully consider the dynamic nature of interactive logic and state evolution, which is precisely the core mechanism by which honeypot contracts achieve deception.

2.3 Special Challenges in Honeypot Contract Detection

The detection of honeypot contracts faces three core challenges:

(1) High Concealment: Malicious logic is often disguised within normal business code, harmless state variables, or compiler features, making it difficult to identify through syntax or simple patterns.

(2) Interaction Dependency: Attack triggers usually depend on specific sequences of external calls or state conditions; single-dimensional code analysis cannot reconstruct the complete attack chain.

(3) Adversarial Evolution: Honeypot designers actively evade known detection patterns (e.g., replacing high-frequency opcodes, control flow obfuscation), requiring detection methods to possess continuous adaptation capabilities.

Current methods based on control flow matching or symbolic execution can identify some logic traps but struggle to achieve high-precision, fine-grained classification and root cause localization of honeypots.

2.4 Overall Technical Framework of This Paper

To address the aforementioned challenges, this paper proposes a three-layer integrated fine-grained detection framework of "Feature Screening - Interaction Verification - Graph Structure Classification," as shown in Figure 1.

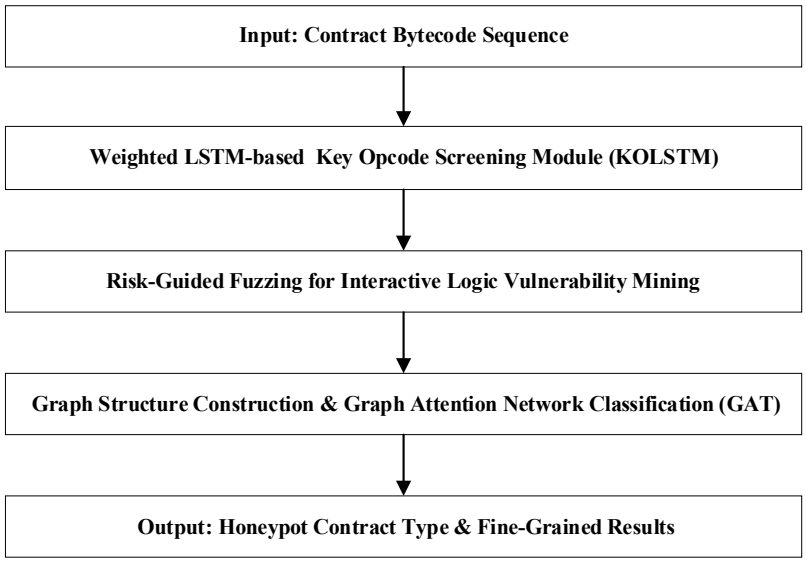

Figure 1. The Proposed Fine-Grained Honeypot Contract Detection Framework

The core innovations of this framework are:

(1) Introducing an opcode weighting mechanism that combines frequency statistics with semantic importance to enhance LSTM's sensitivity to potential malicious code.

(2) Designing a risk-guided fuzzing strategy that uses key opcodes to direct mutation, enabling in-depth testing of interactive logic.

(3) Constructing an "opcode-vulnerability" association graph that integrates static code features with dynamic interactive behaviors, achieving end-to-end fine-grained classification through a Graph Attention Network.

2.5 Comparative Advantages Over Existing Methods

Compared to traditional methods, the proposed framework offers the following advantages:

(1) Comprehensive Coverage: It combines code sequence analysis with interactive behavior

verification, avoiding blind spots inherent in single-perspective detection.

(2) Strong Adaptability: Through dynamic weight adjustment and feedback-driven fuzzing, it can adapt to the adversarial evolution of honeypot contracts.

(3) High Interpretability: The processes of key opcode screening and graph structure construction provide traceable semantic evidence for detection results, aiding security analysts in root cause localization.

(4) This framework provides a closed-loop solution for honeypot contract detection, spanning from feature extraction and behavior verification to structural classification, laying a theoretical foundation for the method design and experimental validation in subsequent chapters.

3. Design of Fine-grained Detection Method for Honeycomb Contract

3.1 Key Operation Code Screening of Honey Pot Contracts Based on LSTM

Since different types of honeypot contracts contain distinct operation codes with varying frequencies, we first calculate the average occurrence frequency of each operation code within the contracts, then assign higher initial weights to high-frequency operation codes [3]. The calculation formula is as follows:

$$f_p = (\alpha_p || \beta) + (g || v)$$

$$w_p = \frac{\partial \|e\|_2^2}{f_p W_o}$$

In the above expression, the notations are defined as follows: α_p denotes the base distribution of operation p in the contract, β denotes the null string used for encoding in the contract, g denotes the actual hidden code, v denotes the state variable, f_p denotes the occurrence count of operation p, ∂ denotes the call address of the target account, e denotes the conditional jump instruction, W_o denotes the hidden state update parameter, and w_p denotes the initial weight of operation p.

The weight initialization strategy draws inspiration from the TF-IDF concept in information retrieval, adapted for opcode sequence analysis. In honeypot contracts, frequently appearing opcodes (e.g., CALL, SELFDESTRUCT, JUMPI) are often associated with sensitive behaviors such as fund transfer and conditional jumps, yet their importance varies significantly across contract types. Therefore, this paper considers not only frequency but also introduces a

"contract discriminability" factor to prevent commonly occurring opcodes (e.g., PUSH, DUP) from dominating model attention due to their universal high frequency. In practice, if an opcode appears frequently across most contracts, its initial weight is appropriately attenuated, thereby focusing more on opcode patterns distinctive to honeypots.

In conventional Long Short-Term Memory (LSTM) models, an operation code weight mechanism is introduced to develop an enhanced long short-term memory network called KOLSTM. By implementing a weighted update strategy for input and hidden gates, the system calculates the weight contribution of high-frequency operation codes, as shown in the following formula:

$$y_p = sig\ mod\left(log\left(\frac{D}{D_1 + 1}\right)u + w_p\right)$$

In the above expression, D denotes the opcode vector input at the current moment, D_1 refers to the output of the forgetting gate, u represents the proportion of the cell state output relative to the hidden state, and y_p stands for the weight contribution quantization value corresponding to operation p.

The importance score is calculated based on the weight contribution of the operation code during model training, as shown in the following formula:

$$a_p = \frac{\sum_{p=1}^{n} y_p \bullet I}{\theta_h - j_o}$$

In the above expression, n denotes the number of contracts, I represents the indicator function, θ_h represents the word vector expression of the weighted average operation code; j_o represents the adjustable parameter matrix; and a_p represents the importance evaluation score of the operation code p.

Based on the importance score of operation codes, the S top-performing codes are selected as the construction operation codes, followed by vulnerability mining in contract interaction logic.

3.2 Fuzzing-based Vulnerability Mining of Contract Interaction Logic

Fuzzing is a fuzz testing technique for general network protocols. In honeypot contract detection, it selects key operation codes based on their characteristics to test and identify interaction logic vulnerabilities.

Based on the risk level defined by input space and key operation codes, the initial test case set is generated. This set consists of three parts: message header, function code, and data code

[4]. The decomposition method is shown in Table 1.

Table 1. Test Case Decomposition Method

message field	span
transaction identifier	Unlimited matching values
protocol identifier	The default value is 0
command identifier	The default value is 0
fill character	15
element ID	1~256
option code	0~255
state changing code	1~535
element ID	1~17
Length identifier	0~535

To reduce the selection frequency of test data objects and simplify the computational process, the variation factors and their values of each identifier and function code in the message field are merged. Based on the characteristics of normalized value ranges, the probability of variation factors for identifiers and function codes is determined [5]. As shown in the following formula:

$$P = (p_0, p_1, \ldots, p_m) = \begin{bmatrix} b(y_k = 0|x_u) \\ \\ b(y_r = a_p|x_u) \end{bmatrix}$$

In the above expression, p_m denotes the mutation probability of the m-th function code, b represents a random variable, y_k stands for the numerical mapping of the *k*-th identifier, x_u refers to the input message template, y_r denotes the numerical mapping of the *r*-th function code, a_p represents the importance evaluation value of opcode *p*, and *P* denotes the mutation probability matrix.

To improve the path coverage of fuzzing tests, this paper designs a risk-guided directional mutation algorithm. The algorithm first marks the test message fields containing key opcodes based on their importance scores. Subsequently, a hierarchical mutation strategy is adopted: high-risk fields (e.g., state confusion codes, option negotiation codes) undergo multiple rounds of random mutation and boundary value testing, while medium- and low-risk fields undergo lightweight random perturbations. Additionally, a feedback mechanism is introduced, where code coverage and state change records after each test execution are used as inputs to dynamically adjust the mutation factor probability matrix, enabling iterative deep exploration of potential honeypot logic. The algorithm flow is as shown in Algorithm 1.

By analyzing the correlation distribution among public codes, custom codes, and reserved codes in the testing protocol, we deploy a blockchain-based testing environment. In this environment, the mutation probability matrix of function codes and identifiers serves as the

input combination for triggering anomalies. Initial test cases are used to validate vulnerabilities. If the data fields of all identifiers and function codes in the message domain display as "empty", it confirms the presence of honeypot logic in the contract, requiring further detection of vulnerability types in the honeypot contract.

Input: Initial test case set T, key opcode list K, mutation rounds R
Output: Vulnerability-triggering test case set V
Step 1. Initialize vulnerability-triggering test case set: V ← ∅
Step 2. For each mutation round r = 1 to R do
Step 3. For each test case test ∈ T do
Step 4. Identify overlapping message fields:
Let F_{overlap} be the set of message fields in test that contain opcodes from K
Step 5. For each field f ∈ F_{overlap}, select mutation strategy:
strategy(f) = random_mutation if risk(f) = high
strategy(f) = boundary_testing if risk(f) = high
strategy(f) = light_perturbation if risk(f) ∈ {medium, low}
where risk(f) is determined by the opcode importance score
Step 6. Generate new test case: test' = mutate(test, strategy(f))
Step 7. Execute test' in local chain deployment environment
Step 8. If execution triggers abnormal state or "empty data field":
V ← V ∪ {test'}
Step 9. End for
Step 10. Update mutation factor probability matrix based on coverage feedback:
M_{mut}^(r+1)←update_matrix(M_{mut}^(r), coverage_data)
Step 11. End for
Step 12. Return V

Algorithm 1. Risk-Guided Fuzzing Mutation Algorithm for Honeypot Contract Detection.

3.3 Fine-grained Detection of Honey Pot Contracts

3.3.1 Overview of the overall testing process

The complete detection process, from opcode filtering to graph attention network classification, forms a closed-loop chain, as illustrated in Figure 2. The first step involves filtering key opcodes using an improved KOLSTM model, while generating suitable test cases with the help of Fuzzing technology to explore potential interaction logic vulnerabilities in the contract, providing core feature support for subsequent detection. The second step involves using the filtered key opcodes as nodes in a graph structure, and the discovered interaction logic vulnerabilities as connecting edges between nodes, to construct a source code graph structure

that accurately represents the core features of the contract. The third step involves updating each node vector based on the structural relationship matrix between nodes and edges, and then aggregating all updated node vectors through a global accumulative pooling function to generate a graph-level vector that comprehensively reflects the overall characteristics of the contract. The fourth step involves inputting the graph-level vector into the fully connected layer of the graph attention network, combining it with the cross-entropy loss function to minimize the deviation between the predicted type and the actual type, completing model training and precise classification of honeypot contract types, ultimately achieving fine-grained detection.

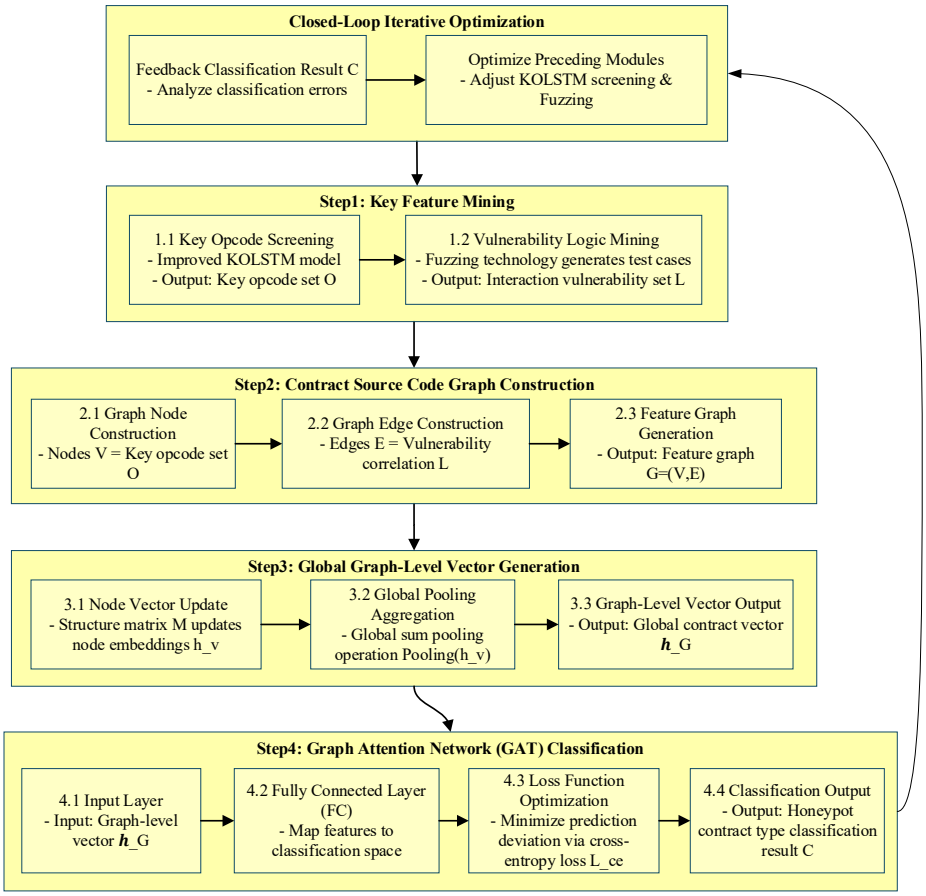

Figure 2. Closed-Loop Linkage for Fine-Grained Detection of Honeypot Contracts

This process achieves a full-chain analysis from code feature extraction, interactive testing to graph structure modeling, combining the advantages of static analysis and dynamic verification. It can effectively identify covert honeypot logic that is only triggered under specific transaction sequences.

3.3.2 Specific implementation process

To mitigate the impact of non-critical lexical elements in honeycomb contracts on contract

type identification, we leverage the filtered key operation codes derived from interaction logic vulnerability mining to construct a source code structure diagram. Operation codes serve as graph nodes, while logical vulnerabilities function as connecting edges. By analyzing the structural relationship matrix between nodes and edges, we update the node vectors of contract vulnerabilities [6]. The expression is as follows:

$$h_{\varepsilon} = \sum_{\varepsilon=1}^{Q} P\varsigma_{\varepsilon} \bullet \psi$$

In the above expression, Q denotes the number of structure graph nodes, P represents the mutation probability matrix, ς_{ε} stands for the parameter matrix of the ε-th node, ψ refers to the coverage rate of key opcodes in the contract, and h_{ε} denotes the update vector of node ε.

On this basis, the update vectors of all nodes are aggregated using the global cumulative activation function to generate the graph-level vector H, which is given by:

$$H = R(h_{\varepsilon} | \kappa_{\delta} \in V)$$

In the above expression, R denotes the global cumulative activation function, κ_{δ} represents the δ-th token in the contract, and V denotes the token set.

The graph-level vector of the contract is fed into the fully connected layer of the graph attention network, where a cross-entropy function is introduced to minimize the deviation between the output vulnerability type and the actual type. This enables the training of the classification network, ultimately determining the corresponding category for the honeypot contract to be detected [7]. As shown in the following formula:

$$Output = HT(x) + \sum_{c=1}^{\xi} \frac{1-\chi}{\xi}$$

In the above expression, $T(x)$ denotes the cross-entropy function, ξ represents the total number of vulnerable contract types, χ denotes the training sample subset, and $Output$ denotes the output vulnerable contract type.

The source code graph structure is constructed by exploiting critical operation codes and interaction logic vulnerabilities. The global accumulation pooling function is used to update and aggregate the vector of structural nodes, thereby generating graph-level vectors. These vectors are then input into the graph attention network, where the cross-entropy loss function is employed to output the type of honeypot contract, achieving fine-grained detection of honeypot contracts.

4. Case Study Analysis

4.1 Experimental Preparation

The experimental dataset used in this study is HD-DATA-NORMAL, containing 1,200 honeypot contracts that cover six distinct categories, as detailed in Table 2.

Table 2. Types of Honey Jar Contracts and Corresponding Instance Numbers

order number	Honey Pot Contract Type	Instance count	core deception
1	Ultra-long hidden space	200	Hide key code with extra-long spaces
2	logical trap	200	Use state variable preset
3	Uninitialized pointer type	200	Using the default behavior of uninitialized storage pointers in Solidity
4	inherited conflict	200	Variable Overwriting Caused by Inheritance Conflict
5	Gambling game type	200	pseudorandom number generation vulnerability
6	compiler exploit	200	The Error of Encoding the Empty String Parameter by Compiler

Using AFL++ v4.15c as the fuzzing tool, 100 test cases were generated through smart contract compilation and deployment. Ten test accounts were configured using a blockchain simulator. The LSTM model was employed to decompose the account contract bytecode into operation code vectors, constructing [contract address, operation code vector, label] triplets. The input sequence length was set to 256, with the first five key operation code weights assigned in order as 0.223,0.152,0.110,0.964, and 0.523. The batch size was 64, the training rounds were 50, and the queue size was 100. Based on the honeypot contract types shown in Table 2, the attack process was manually simulated to verify the model's classification effectiveness.

4.2 Experimental Results

The proposed honeypot contract detection method, combined with the SBERT-CNN-BiLSTM-Attention-based approach and the program slicing-graph neural network method, were applied to identify contract vulnerabilities. Figure 3 presents the false positive rates for these three methods across six distinct honeypot contract types.

Figure 3 clearly demonstrates that when applying the literature-based method to six specific honeypot contract categories, the resulting false positive count significantly exceeds that of our proposed method. This indicates that neither approach can accurately identify the specific vulnerability types of these honeypots. In contrast, the design-based method achieves sub-3 false positives across all six contract types, enabling fine-grained detection. These results validate our method's effectiveness in reducing misclassification risks while demonstrating high

detection accuracy and practical applicability.

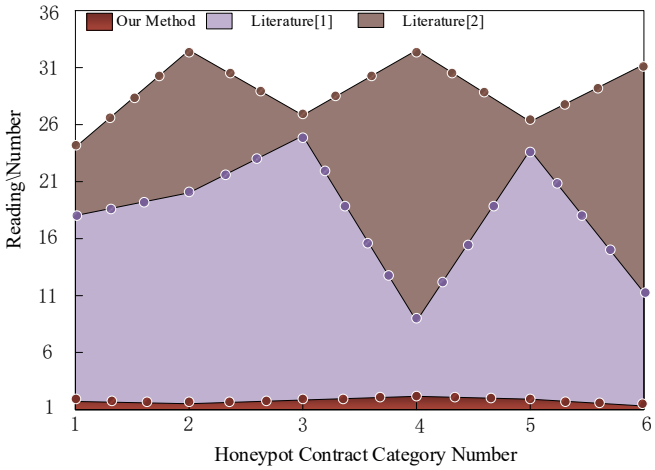

Figure 3. Comparison of honeypot contract test results

5. Conclusion

This study develops an intelligent solution for fine-grained honeypot contract detection through deep integration of LSTM temporal modeling and Fuzzing mutation testing techniques. The approach employs LSTM networks to filter critical operation codes within contracts, while Fuzzing test cases are utilized to identify specific vulnerability types. Experimental validation demonstrates the method's reliability in honeypot contract detection. This achievement provides a low-false-positive and highly interpretable detection tool for smart contract development, facilitating the transition from passive response to proactive defense in smart contract security technology. The research holds significant theoretical and practical value.

Future work can be further extended to honeypot detection in a multi-chain environment, exploring collaborative attack patterns of cross-chain contracts, and investigating a hybrid detection framework combining symbolic execution and deep learning to enhance the discovery capability of zero-day honeypot logic. Additionally, consideration can be given to building an open-source honeypot contract detection platform to promote the co-construction and sharing of the industry's security ecosystem.

References

[1] He, D., Wu, R., Li, X., Chan, S., & Guizani, M. (2023). Detection of vulnerabilities of blockchain smart contracts. *IEEE Internet of Things Journal*, *10*(14), 12178-12185.

[2] Zhang Renlou, Wu Sheng, Zhang Hao, & Liu Fangyu. (2025). Slice-GCN: 基于程序切片与图神经网络的智能合约漏洞检测方法[Slice-GCN: An Intelligent Contract Vulnerability Detection Method Based on Program Slicing and Graph Neural Networks]. *Journal of Cyber Security*, *10*(1), 105-118.[in Chinese]

[3] Zhang, L., Li, Y., Guo, R., Wang, G., Qiu, J., Su, S., ... & Tian, Z. (2024). A novel smart

contract reentrancy vulnerability detection model based on BiGAS. *Journal of Signal Processing Systems*, *96*(3), 215-237.

[4] Zhang, J., Lu, G., & Yu, J. (2024). A Smart Contract Vulnerability Detection Method Based on Heterogeneous Contract Semantic Graphs and Pre-Training Techniques. *Electronics*, *13*(18), 3786.

[5] Gu, M., Feng, H., Sun, H., Liu, P., Yue, Q., Hu, J., ... & Zhang, Y. (2022, May). Hierarchical attention network for interpretable and fine-grained vulnerability detection. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 1-6). IEEE.

[6] Li, L., Liu, Y., Sun, G., & Li, N. (2023). Smart contract vulnerability detection based on automated feature extraction and feature interaction. *IEEE Transactions on Knowledge and Data Engineering*, *36*(9), 4916-4929.

[7] Liu Fangqing, Huang Han, Xiang Yi & Hao Zhifeng.(2023). 基于流形鸽群优化的智能合约重入性漏洞检测方法研究 [Research on Intelligent Contract Reentrancy Vulnerability Detection Based on Manifold Pigeon Swarm Optimization]. *Scientia Sinica(Technologica)*, 53(11),1922-1938. [in Chinese]