Journal of Information Systems Engineering and Management
2025, 10(24s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Examination and Performance Evaluation of Smart Contract Reentrancy Vulnerability using Deep Learning Detection Tools

Mrs. Saltanat Mohammed Abdullah Shaikh[1], Dr. Sangeeta Vhatkar[2]
[1]*Research Scholar, Ph.D. Student – Information Technology, Thakur College of Engineering & Technology*
[2]*Associate Professor – Information Technology, Thakur College of Engineering & Technology*

ARTICLE INFO

Received: 19 Dec 2024

Revised: 10 Feb 2025

Accepted: 25 Feb 2025

ABSTRACT

Introduction: This study proposes, from the perspective of optimizing the model feature space, a deep learning-based vulnerability detection technique for Ethereum smart contracts based on Raptor vision invasive Hunt optimization.

Objectives: To examine experimental results of available toools and novel model in detecting the re-entrancy vulnerability.

Methods: By combining the coordinated invasive hunting behaviour of Gallus domesticus with the intelligent vision-based in-depth driving behaviour of osprey, the RVIhO algorithm is achieved. They successfully adjust the model's hyperparameters with increased productivity for precise vulnerability detection based on these hybridising traits.

Results: A large-scale dataset with different vulnerabilities having the findings demonstrate that it suggested strategy performs exceptionally well in terms of detection 93.25% accuracy rates attained for detection of re-entrancy vulnerabilities of provided smart contract data set.

Conclusions: The results presented, and performance compared with the best available deep learning methods for vulnerability detection.

Keywords: Blockchain, Smart Contract, Smart Contract Vulnerabilities, Detection Technique**.**

INTRODUCTION

The blockchain has changed the game for digital asset trade and monetary transactions. This groundbreaking development has significantly altered the nature of online commerce. Smart contracts are the backbone of the blockchain because they allow for transactions to be made between the parties without requiring the contract to be mediated by a third party. SCs, or "smart contracts," are digital contracts that may be carried out mechanically. These contracts are kept on a distributed ledger called a blockchain. Smart contracts are being utilized more often to enable distributed commerce and program execution. Despite their useful features, because of their intricacy and lack of uniformity, they are vulnerable to attacks, like automation and transparency. Many security flaws exist in smart contracts because of their immutability and autonomy, as well as the wide range of programming languages that may be used to generate them. security. There are flaws and logical mistakes in every programming language. Flaws in the logic or code of smart contracts may result in a huge loss of digital assets and damage the image of blockchain technology as a whole, thus guaranteeing their security is of the utmost significance. The need to verify the safety and security of SCs smart contracts is growing as a result. Smart contracts may be protected against a variety of threats by using the numerous static tools available. However, a completely automated option is not yet available[1].

Smart Contracts:

This section presents an overview of the potential risks and hazards associated with smart contracts. The potential elimination of intermediaries such as banks or attorneys can be achieved by the automation of intricate financial transactions using blockchain based "smart contracts" (SCs). The breakthrough outlined above represents significant advancement in the area of blockchain technology and holds the potential to fundamentally transform the dynamics

of commercial transactions [1]. Prominent blockchain systems encompass Ethereum, Hyperledger Fabric, Corda, Hyperledger, and EOS [2]. Each of these platforms employs its own exclusive programming language for the development of SCs. Ethereum employs Solidity, a programming language that is specifically designed for contract-oriented programming. Similar to JavaScript [3,4], Solidity was developed with the objective of ensuring user-friendliness and accessibility. The underlying structure of Hyperledger Fabric consists of chain code, which may be implemented using programming languages such as Go, Java, or JavaScript [5-7]. The Corda blockchain technology was developed using the Kotlin programming language, which shares similarities with Java [8,9]. The programming languages employed in the development of SC, EOS, and Tron were C++ and Solidity, correspondingly. The primary objective of all blockchain systems is to facilitate the creation of secure, operational, and blockchain-compatible smart contracts (SCs). However, it is worth noting that various blockchain systems employ distinct programming languages for the production of smart contracts. Given the multitude of potential risks, such as the presence of defective code, fraudulent inputs and the few attacks on the blockchain network, it is imperative to prioritize the establishment of robust security measures for smart contracts (SCs). Blockchain systems are susceptible to financial losses and erosion of user confidence as a result of inherent security vulnerabilities. Therefore, it is imperative to prioritize the security of (SCs) smart contracts throughout the development of blockchain-based applications [10,11,12].

There are various types of attacks in the context of cybersecurity and blockchain technology. These attacks include DDOS attacks, routing attacks, Sybil attacks, phishing attacks, 51% majority attacks, reentrancy attacks, double spending attacks, smart contract overflow and underflow, short address attacks, transaction ordering dependence, exceptional handling, frozen funds attacks, infinite loop attacks, timestamp dependence, callstack depth, and unchecked send. Each of these attacks poses a unique threat to the security and the integrity of blockchain systems [13].

Ethereum Smart Contract (SCs) Vulnerability - Reentrancy

Reentrancy vulnerability is regarded as a call-to-action value that has the ability to call itself back over a series of calls. In addition to its other flaws, re-entrancy is a significant problem for Solidity smart contracts. A hostile actor may launch a re-entrancy attack if they are able to pause the process in progress and rejoin the contract with fresh instructions before the process has been fully executed. When several outcomes are intended to result from a single transaction, this flaw appears in the smart contract.

Therefore, the attacker may be able to take advantage of unexpected contract behaviour brought on by this phenomenon. This hole might be used by bad actors to repeatedly run the same procedure, which could lead to a breach of the contract and the theft of revenue. Developers may take precautions, including using the "check effects interaction" pattern or reducing the gas allocation for external calls. These precautions are taken to reduce the possibility of re-entrancy attacks by making sure all system state changes are made before any external calls are made. Tests and audits of the contract code should be thorough to find and fix any re-entrancy issues it may include [7].

OBJECTIVES

To Evaluate different vulnerabilities available with range of the respective tools available. To examine experimental results of BiLSTM, MEVD, Optimized CodeBERT and novel model in detecting the re-entrancy vulnerability.
To perform Comparative Analysis of BiLSTM, MEVD, Optimized CodeBERT and Triplet loss BiLSTM Under reentrancy vulnerability Conditions.

METHODS

Input data of vulnerability detection

The unknown vulnerabilities of smart contracts are detected in the research under the utilization of historical Ethereum transaction data, which contains classified opcode data sequences with or without vulnerabilities for performing the evaluation. The input is attained from a malicious smart contract dataset [26], which is represented as,

$$CV = \sum_{i=1}^{r} C_i, V_i \quad (1)$$

where, r denotes the total number of contracts with respective vulnerabilities, Further, the data organized as $C_i \in C$ and, $V_i \in V$, where C defines the set of contracts, and, V indicates the vulnerability that is affected in the contrast.

Pre-processing using Natural Language Techniques

The attained byte code includes various strings that contain symbols, characters, and words, together represent the text. In the pre-processing section, the raw sequence data are converted into a unique separator. To achieve, these unique separator processes, the pre-processing phase performs both tokenization and lemmatization processes effectively.

The input textual data contains various sets of characters that are converted into digital representation by tokenization to preserve the sensitive data. The stream of textual content is broken into terms, symbols, elements, and words called tokens, which are utilized in computer science and linguistics applications. These stacks of tokens are combined as input sources for performing text mining and parsing. In general, tokenizations mainly occurred in word-level documents to recognize the meaning of full keywords.

The input textual content is allowed into the lemmatization process to obtain the root word by finding the lemma to express the proper intended meaning for better understanding [28]. The attained pre-processed outcome D, after performing tokenization and lemmatization is mentioned as,

$$D = C_i^* V_i^* \tag{2}$$

Feature Extraction with Word Term Frequency Encoder Graph features

The obtained normalized word content is then subjected to the feature extraction phase that extracts the features and converts them into vectors. These text vectorizations are achieved by performing word2vec features, TF-IDF features, and AE-based graph features respectively.

Conceptually, word2vec features represent vector form notation for every word in the vocabulary, based on the semantic relationship of the word. Similar words garner similar vector values that are grouped in the same block for evaluation. The similarity value of the word is measured by a cosine similarity that ranges from -1 to 1. Moreover, word2vec features extract the vector value by vector dimensions and window size. All the word contents are converted into vectors and measure the distance among them. The outcome dimensions of word2vec features are (1×20), which is expressed as,

$$F(D) = \{\vec{u}_1, \vec{u}_2, \vec{u}_3, \ldots \vec{u}_r\} \tag{3}$$

where $\vec{u}_1, \vec{u}_2 \ldots \vec{u}_r$ denotes the vector representation of word content obtained from D.

TF-IDF is a statistical method algorithm that estimates the significance of the word in the document, which is the most often used feature extraction method to calculate the word weight function in the current vector space. Mainly, the function of TF-IDF is categorized into two such as word frequency and inverse text frequency. Word frequency observes the occurrence of certain words that are present in the entire document, whereas inverse text frequency evaluates the importance of the word. Furthermore, the inverse text frequency performs the quotient logarithm for the entire document along with the document containing the specific word. The mathematical representation of both frequency word and inverse text frequency are combined to form TF-IDF, which is depicted as,

$$tf(D) = \sum_{i=1}^{r} tf\left(D^c\right) \tag{4}$$

$$idf(D) = \log\left(\frac{g}{k+1}\right) \tag{5}$$

$$TF - IDF(D) = \log\left(\frac{g}{k+1}\right).\sum_{i=1}^{r} tf\left(D^c\right) \tag{6}$$

where, c defines the specific word, g indicates the total number of documents, and k expresses the total number of sample words in the document. The achieved outcome dimensions of TF-IDF are $^{(1\times 50)}$.

The graph embedding features are evaluated based on the distributed relation of diverse weighted matrices. The major contribution of the AE graph feature is to estimate the mapping function and convert the low-dimensional representation into high-dimensional based on graph vertices. Basically, the graph features are extracted as intrinsic and penalty-based graphs. In intrinsic-based graph that represents a similar data sample matrix along with its similar vector values. Similarly, in penalty-based graph that represents the relation among the specific characteristics of data. Based on these representations, the AE-based graph feature extracts the map function for diverse graph weights and analyzes the loss function for distinct feature representation effectively. Additionally, the graph feature maintains the local and global structure of the sample [31]. The basic AE-based graph feature representation is expressed as,

$$H = \{I, J\} \tag{7}$$

where, I refers to the sample data matrix and J represents the similarity matrix, both ^{I,J} where obtained from $_D$. The achieved outcome dimensions of the AE-based graph feature are $^{(1\times 50)}$. The attained outcome of feature extraction methods is arranged in a stack form with the dimension of $^{(1\times 120)}$, which is depicted as,

$$M = \{F, TF-IDF, H\} \tag{8}$$

Raptor Vision-Invasive Hunt Optimization

The RVIhO is utilized in the fourth layer of the model that effectively tunes the hyperparameter for achieving accurate vulnerability detection. The RVIhO algorithm accumulated from the behavior of vision-based drive character, intelligence character of Osprey [34], and invasive hunting and competitive search character of hen. From this perspective, the hierarchical hunting character and coordinate attempt character obtained from the hen effectively improved the detection accuracy of the model. Meanwhile, the intelligent and vision-based drive character obtained from Osprey enhanced the accurate vulnerability detection in the Ethereum network. In this context, the RVIhO-LGAtt-G2SN model achieves effective performance with better quality detection results, which also reduces redundancy and increases the availability of the model.

RESULTS

The major goal of all the given tools is to do a source-level security analysis of Solidity programs. There are nine different tools available, but only four of them can detect vulnerabilities in bytecode. On closer inspection, many of these resources have been updated within the previous several months, as evidenced by the data supplied in the latest update column. All the software is publicly available on GitHub and is free to use. Python, Rus, Java, and Solidity are the four languages used to create platforms that aid in vulnerability detection. Python, among the other possible programming languages, was found to have the most widespread use. The evaluation indicators used in experiment are recall, precision, F1-score, and accuracy [1].

Arithmetic, Re-Entrancy, Inconsistent Access Control, Unchecked Calls, and Security-Contract with Amount 20,044, 42,573, 39,098, 28,171, and 35,130 contracts, respectively, are among the vulnerabilities found in the statistics on the quantity of each type of contract. There are 165,000 smart contracts in the dataset, which is split 8:2 into training and test sets elaborate in Table 1.

Average time as on Table 1 for detection of re-entrtancy vulnerability in smart contract by novel method is 0.09s compared with Mythril and Oyente are 4.53s and 4.41s respectively. Models based on DL are evaluated based on metrics such as Accuracy, Precision, F1 score, and Recall. BiLSTM achieved an accuracy of 81.01%, precision of 78.93%, recall of 80.06%, and an F1 score of 79.71%. MEVD performed better with an accuracy of 92.13%, precision of 89.49%, recall of 90.15%, and an F1 score of 89.28%. Optimized CodeBERT showed promising results presented in Table 1 with an accuracy of 93%, precision of 85.45%, recall of 80.45%, and an F1 score of 87.29%. The model presented as "Ours" outperformed the others with an accuracy of 93.25%, precision of 96.20%, recall of 86.13%, and an F1 score of 90.89%.

The results presented and discussion compared with the best deep learning methods like BiLSTM, MEVD, Optimized CodeBERT and techniques available and researched till date. To achieve descent accuracy, precision, Recall and f1 score with the novel method based on triple loss and BiLSTM detection methods for vulnerabilities using deep learning depicted in Table 1.

Models based on DL	Accuracy	Precision	Recall	F1 score
BiLSTM	81.01%	78.93%	80.06%	79.71%
MEVD	92.13%	89.49%	90.15%	89.28%
Optimized CodeBERT	93%	85.45%	80.45%	87.29%
RVIhO	93.25%	96.20%	86.13%	90.89%

Table 1. Experimental results of BiLSTM, MEVD, Optimized CodeBERT and novel model in detecting the re-entrancy vulnerability.

The BiLSTM framework [13] This approach uses BiLSTM in conjunction with an attention strategy to identify several opcode vulnerabilities in smart contracts. First, we preprocessed the data to turn the opcode into a feature matrix that could be fed into the neural network. We then classified multiple-label smart contracts using the BiLSTM model, which is based on the attention mechanism. The technique suggested in this research for multiple vulnerability detection tasks in smart contracts is effective, as demonstrated by the testing findings, which demonstrate that the model can detect several vulnerabilities simultaneously and that all evaluation indicators surpassed roughly 80%.

MEVD based model [14]: To find known, high-risk vulnerabilities in smart contracts, apply the multi-scale encoder vulnerability detection (MEVD) technique. First, we build a new surface feature encoder (SFE) that improves the semantic content of code features by leveraging the gating mechanism. The global structure and local detail features of the smart contract code are then recorded by a dual-branch encoder that we create by combining a base transformer encoder (BTE) with a detail CNN encoder (DCE). Lastly, we use the deep residual shrinkage network (DRSN) to identify model features that are associated with susceptibility. With an average detection accuracy of 90%, experimental findings on three different kinds of high-risk vulnerability datasets demonstrate how well the MEVD method works in comparison to state-of-the-art techniques.

Optimized CodeBERT [1]- Through the use of deep learning techniques, optimised CodeBERT creates Lightning Cat, a tool for detecting the SCs vulnerabilities. Three models of deep learning are optimized by the solution. We present an efficient technique for preparing data that identifies the semantic characteristics of vulnerabilities in smart contracts. We obtain code snippets of functions that include vulnerabilities during the data preprocessing step in order to extract vulnerability features. With the primary objective of enhancing the model's semantic analysis skills, we additionally use the pre-trained CodeBERT model for data preprocessing. This paper's Lightning Cat vulnerability detection tool outperforms other tools in terms of detection performance, according to the findings of the experimental evaluation. The lightning cat optimised CodeBERT model , which is superior to the Optimized-LSTM models and Optimized-CNN models.

DISCUSSION

We propose a unique method for vulnerability identification from the perspective of feature representation space optimization. Unlike previous approaches, we do not combine multiple models to enhance the model's overall performance in feature learning. To maximize the model's ability to represent features, however, construct the metric learning triplet loss function on top of the conventional binary cross-entropy loss function. Contracts belonging to the same category are closer together while contracts belonging to different categories are farther apart because of feature space optimization, which also improves the model's detection accuracy and helps to somewhat mitigate the high false-positive rate of the previous method. The suggested plan improves vulnerability detection's interpretability. Using the smart contract's source code as the input data, we can identify the root cause of the vulnerability and filter out its salient characteristics by applying word vectorization and attention mechanisms. To find vulnerabilities in contract source code, we built a sizable dataset. From the Etherscan website, we gathered 165,000 validated contracts, which we then classified using Slither and Mythril. The detection performance of the

technique is outstanding. We contrasted this model's performance with that of other deep learning-based techniques and conventional approaches based on symbolic analysis in order to accurately assess its performance.

REFRENCES

[1] S. Vani, M. Doshi, A. Nanavati, A. Kundu. Vulnerability Analysis of Smart Contracts. Cornell University, arXiv:2212.07387 [cs.CR] Cryptography and Security. 2022, December, 15. Available from: https://doi.org/10.48550/arXiv.2212.07387

[2] Dawei Yuan, Xiaohui Wang, Yao Li, Tao Zhang. Optimizing smart contract vulnerability detection via multi-modality code and entropy embedding. Journal of Systems and Software 2023, August; Volume 202, 111699. Available from: https://doi.org/10.1016/j.jss.2023.111699

[3] Mojtaba Eshghie, Cyrille Artho. Dynamic Vulnerability Detection on Smart Contracts Using Machine Learning. EASE '21: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, June 2021 Pages 305–312. Available from: https://doi.org/10.1145/3463274.3463348

[4] Peng Qian, Zhenguang Liu1, Qinming He, Roger Zimmermann, Xun Wang1. Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models. IEEE Access. 2020, January 31; PP(99):1-1. Available from: https://doi.org/10.1109/ACCESS.2020.2969429

[5] Junjun Guo, Jingkui Li. Smart Contract Vulnerability Detection Based on Multi-Scale Encoders. MDPI Electronics 2024. 2024, January 18; 13(3), 489. Available from: https://doi.org/10.3390/electronics13030489.

[6] Tharaka Mawanane Hewa, Yining Hu , Madhusanka Liyanage,Salil S. Kanhar. Survey on Blockchain-Based Smart Contracts: Technical Aspects and Future Research. IEEE Access. 2021, June 24. Available from: http://dx.doi.org/10.1109/ACCESS.2021.3068178

[7] Satpal Singh Kushwaha 1, Sandeep Joshi. Ethereum Smart Contract Analysis Tools: A Systematic Review. IEEE Access. 2022, June 3. Available: http://dx.doi.org/10.1109/ACCESS.2022.3169902

[8] Kushwaha , Sandeep Josh, Dilbag Singh, Manjit Kaur. Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. IEEE Access. 2022, January 20; Vol. 10 pp. 6605 – 6621. Available from: https://doi.org/10.1109/ACCESS.2021.3140091

[9] Zhongju Yang , Weixing Zhu, And Minggang Yuzhenpeng Liu,Mingxiao Jiang, Shengcong Zhang2, Jialiang Zhang, And Yi Liu. Improvement and Optimization of Vulnerability Detection Methods for Ethernet Smart Contracts.IEEEAccess.2023,August1;PP(99):1-1.Availablefrom: http://dx.doi.org/10.1109/ACCESS.2023.3298672

[10] Seon-Jin Hwang, Seok-Hwan Choi , Jinmyeong Shin. CodeNet: Code-Targeted Convolutional Neural Network Architecture for Smart Contract Vulnerability Detection. IEEE Access. 2022, March 29; Volume 10. Available from: http://dx.doi.org/10.1109/ACCESS.2022.3162065

[11] Zhenpeng Liu, Mingxiao Jian, Shengcong Zhang, Jialiang Zhang. A Smart Contract Vulnerability Detection Mechanism Based on Deep Learning and Expert Rules. IEEE Access 2023; PP(99):1. Available from: http://dx.doi.org/10.1109/ACCESS.2023.3298048

[12] XueyanTang, Yuying Du, Alan Lai, Ze Zhang & Lingzhi Shi. Deep learning-based solution for smart contract vulnerabilities detection. IEEE Access. 2023, August 1. Available from: http://dx.doi.org/10.1109/ACCESS.2023.3298048

[13] Chenyi Qian, Haohan Ning, Yaqiong He, Mengqi Chen. Multi-Label Vulnerability Detection of Smart Contracts Based on BiLSTM and Attention Mechanism. MDPI Electronics. 2022, October 10; 11(19): 3260. Available from: https://doi.org/10.3390/electronics11193260

[14] Peng Qian, Zhenguang Liu. Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models. IEEE Access. 2020; PP(99):1-1. Available from: https://doi.org/10.1109/ACCESS.2020.2969429