[1]Janhavi Satam

[2]Sangeeta Vhatkar

[3]Kavita Singh

Examination of Approaches for Identifying Vulnerabilities in Smart Contracts

Abstract: - Objective: By reviewing various previous works, this paper collects the multiple of approaches, strategies used to identify vulnerabilities in smart contracts. Blockchain is a decentralized technology that securely and immutably, records transactions across numerous computers in a visible manner. On a blockchain, smart contracts are self-executing agreements that independently execute and verify contract conditions. This reduces the need for middlemen and increases transparency. Smart contract vulnerabilities are problems in the code that could allow other parties to gain access to, alter, or steal assets as a result of mistakes, faults or imperfections made during development, thereby causing financial and operational harm. In this paper we have algorithms, techniques to detect vulnerabilities in smart contract using deep learning found in literature surveys. Methods: We have found some techniques using opcode, bytecode, Skip-Gram-Word2Vec to convert the smart contract file. Findings: We have found that LSTM, Vanilla-RNN, GRU have very less accuracy 49.64,53.68,54.54. Novelty & Applications: We will come with some different algorithms that will understand different vulnerability with more accuracy. We have come with CNN, Xception, EfficientNet-B2 which has accuracy high then LSTM, Vanilla-RNN, GRU i.e.71,69,75 percent.

Keywords: Smart Contracts, Vulnerability Detection, Deep Learning, Reentrancy Vulnerability.

1.Introduction

Ethereum is the second rising technology in blockchain. It is a platform where decentralized apps can be created as well as smart contracts can be created. In decentralized blockchain networks, there are multiple nodes, and each node has a copy of the entire blockchain ledger. If a block is added in one node, its copy is made in all other nodes. The blocks record are verified after some interval for all the nodes. If one change in block 1 is seen in node A, as the blocks of all nodes are verified after some interval, the change in block 1 of node A is found out, and an update is carried in block 1 with respect to the record compared with other blocks of other nodes, since they all have copies of it. All other nodes in the network check the record of the block and, if they are valid, add the same block to their own copy of the blockchain. This ensures that all nodes have an identical and up-to-date copy of the blockchain ledger.The blockchain is a chain of blocks, and each block in the chain contains a reference to the previous block through its hash value[1]. Blockchain technology is used in various fields such as: 1) Supply Chain Management 2) Smart Contracts 3) Healthcare 4) Voting Systems 5) Financial Services 6) Real estate Supply Chain Finance 7) Charity and Donations 8) Insurance 9) Internet of Things (IoT), etc.Smart contracts are a basic concept in blockchain technology that was presented by a computer scientist named Nick Szabo. The idea of smart contracts included self-executing agreements with contract conditions clearly encoded into code. Here, the Ethereum platform has a language to create the smart contract that is solidity language. There are more such languages to create the smart contract on the Ethereum platform, but the most preferred language is solidity language[2]. Smart contracts in the Ethereum network are deployed and executed on the Ethereum Virtual Machine (EVM). Once the smart contract is deployed on the Ethereum platform, it is immutable. Smart contracts are used by various industries, including Supply Chain Management, Digital Identity, Real Estate, Voting Governance, Healthcare, Insurance, and more. If any vulnerability is found in them, harmful actions can take place by an attacker. Existing tools detect vulnerabilities in smart contracts but do not focus on all the vulnerabilities. Several types of vulnerabilities detected in smart contracts, including Reentrancy Attacks, Block

[1 ,2]Department of Information Technology,

Thakur College of Engineering and Technology, Mumbai, Maharashtra, India

[1]Email: janhavisatam1999@gmail.com; Ph: +91- 8433515266

*[2]Corresponding author: Email: vhatkarsangeeta@gmail.com; Ph: +91-9167110876

[3]Department of Electronics and Telecommunication, Thakur Polytechnic, Mumbai, Maharashtra, India

Email: kavitassingh82@gmail.com; Ph: +91-9769118547

Gas Limit Vulnerability, Transaction Order Dependence Attacks, Time Stamp Dependence, Denial of Service, Integer Overflows And Underflows, Front-Running, Simple Logic Error, etc. These are occurrence of vulnerabilities in smart contract:1)Reentrancy 2) Integer Overflow/Underflow 3) Unchecked External Call 4) Timestamp Dependence 5) Uninitialized Storage Pointer 6) Denial of Service (DoS) 7) Unprotected Ether Withdrawal 8) Front-Running 9) Logic Errors: 10) Access Control Issues , etc. We studied various methods aiming at improving the detection of vulnerabilities through a survey of literatures. Our study includes research articles that show many strategies used with machine learning and deep learning in the field of smart contract security[3].

1.1 Types of Extensions in Ethereum Development and Usage

There are various kinds of extensions in Ethereum. Extensions for Solidity Contracts: On Ethereum, Solidity is the most widely used language for creating smart contracts. The.sol extension is usually used for Solidity contract files. Bytecode Extensions: Bytecode is generated from compiled Solidity code and is published to the Ethereum network. The.bin extension is frequently found in bytecode files.Extensions for the Application Binary Interface (ABI): An ABI is a JSON file that provides instructions on how to communicate with a deployed smart contract. The.abi extension is commonly used with ABI files. Wallet Extensions: Different file extensions are used by wallets, like MetaMask, for their wallet backups[4]. For instance, MetaMask stores its wallet backups in.json format. Keystore Extensions: Encrypted private keys are kept in Keystore files. These files frequently end in.json or.keystore.

1.2 Occurrence of Vulnerabilities

a. A smart contract has a reentrancy vulnerability if an external call is placed to another contract before the current call has finished executing. Users can withdraw their balance in this example by using the withdrawBalance function. Nevertheless, it sets the balance to zero after sending the caller the balance first. A reentrancy attack can occur when a malicious contract calls withdrawBalance and then returns to the function before the balance is updated[5].
b. Smart contracts are vulnerable to timestamp dependence issues if they depend on block timestamps for essential functionality or decision-making. Because miners have some control over the timestamp of a block they mine, several vulnerabilities may result from the nature of timestamps in blockchain systems.
c. When smart contracts depend on the current block number to make important choices or perform necessary actions, they have a block number dependency vulnerability. The vulnerability stems from the inherent characteristics of Ethereum's blockchain, which allow miners to alter the block number to some degree, particularly in the event of a "block reorganization. "Block reorganization can result in the original block being replaced in the blockchain when a miner produces a new block at the same block height as an existing block. This may result in a modification to the block number for a specific contract call or transaction[6].
d. The use of the delegatecall opcode, which enables a contract to execute code from another contract while preserving the context of the calling contract, is the reason behind the delegatecall vulnerability in smart contracts. If not utilized appropriately, this could be harmful since it could result in unexpected behavior and security flaws. In Ethereum, a low-level action called the delegatecall opcode enables a contract to call another contract and run its code inside the calling contract's context. This implies that the called contract has access to the caller's message data and value, as well as the calling contract's storage, balance, and address.
e. A particular kind of vulnerability that can arise in smart contracts because of the way Ethereum performs type conversions and arithmetic operations is known as the "Ether Strict Equality Vulnerability". This weakness may result in unforeseen actions and even financial loss.

1.3 Prevention of vulnerabilities

a. It's critical to adhere to best practices, such as the "Checks-Effects-Interactions" pattern, which requires state changes to be made prior to any external calls, and making sure that external calls are made to trustworthy contracts, in order to prevent reentrancy vulnerabilities.
b. It's crucial to refrain from depending entirely on block timestamps for crucial choices or actions in order to reduce timestamp dependency issues[7].

c. Developers should refrain from depending on the current block number when making important decisions in order to reduce block number dependency vulnerabilities. For time-sensitive tasks, they can instead make use of alternative mechanisms like external time oracles or block timestamps.
d. The delegatecall vulnerability emphasizes how crucial it is to comprehend how delegatecall functions and use it sparingly in order to prevent unexpected outcomes and security issues in smart contracts.
e. To stop the "Ether Strict Equality Vulnerability" in smart contracts, which can happen when unsigned and signed integers are compared and cause unexpected behavior. Employ the SafeMath Library:Use the SafeMath library when doing arithmetic operations, especially with unsigned integers, to avoid overflow and underflow issues. SafeMath offers functions that safely handle certain situations, such as add, sub, mul, and div[8][9].

2. Methodology

The technique SmartPol, a machine learning-based technique for smart contract vulnerability detection, in this study. The data is cleaned up using a data pre-processor before features are extracted and categorized. Secondly, provide a technique based on TSVM for semi-supervised learning classification models to automatically detect smart contract vulnerabilities, as well as an approach based on PSOGSA for data optimization and function extraction.49512 real-world smart contracts were tested on EtherScan using SmartPol. Suggest a combinatorial approach to static analysis. A word embedding data pre-processor was employed for the dataset's feature extraction and classification. Ethereum provides the Ethereum Virtual Machine (EVM) for the execution and invocation of smart contracts. Word2Vec and FastText are the most commonly employed word embeddings in the NLP group. The dataset contains textual parts, thus after the raw data has been processed, the real sense needs to be represented in a vector format. The continuous skip-gram model, which is thought to be related to its ability to pick up knowledge from a dense, low-dimensional term vector that can accurately anticipate the phrases that surround a core word. The Skip-Gram-Word2Vec model architecture, which predicts the terms that surround a word in a text or phrase by first generating word representations. Provide a method based on TSVM for the automatic labeling of smart contract files. In order to enhance the effectiveness of automatic smart contract vulnerability detection, are motivated to incorporate a text labeling algorithm that leverages TSVM as an underlying semi-supervised labeling process. This algorithm can then be used to support more sophisticated supervised text classification techniques. Transactions for 49512 smart contracts that were verified before September 2020 were retrieved from Etherscan and used in this work. PSOGSA approach for optimizing and feature extraction. The TSVM was used to classify vulnerabilities. The algorithms xgboost, adaboost, knn, svm, etc. are compared with PSOGSATSVM[10].

Introduce a smart contract vulnerability detection mechanism. Also describe expert knowledge to identify vulnerabilities like code injection, reentrancy, and calls with hardcoded gas amounts. The entire work is the combination of GNN and expert knowledge. How to express the contract code as a graph, how to extract the graph's features using what method or algorithm, and selecting the appropriate model for label classification (i.e., vulnerability detection) are the three key elements of graph-based vulnerability detection. The edges of the graph denote changes in activity, while the nodes of the graph denote significant function calls or crucial variables. Provide a set of rules for eliminating nodes that are unnecessary to normalize the resulting contract graph to a standard graph. The normalized graph is analyzed using the temporal message propagation network model (TMP), which computes the global label of the graph and verifies the existence of vulnerabilities based on the label. Their main method for extracting graph features and performing label categorization is the TMP network, which has two phases: message propagation and readout. Before and after the deployment of a smart contract are the two phases of their strategy. Before deployment, a component with the GNN model is used to check a contract for flaws. The deployment of the contract is stopped and a bug report is produced if it is discovered to be unqualified. During the runtime phase, the user must define a value for the contract. Expert rules are used to examine the delicate opcodes involved in dangerous operations for high-value smart contracts. This contract transaction is stopped, and an error report is generated for submission once indications of a violation (i.e., a violation of the specified constraint rules) are discovered. Even after the smart contract has been deployed, their rules can stop contract transactions at the EVM level if there are contract problems. The checking mechanism can also create error reports and block contract transactions containing hazardous actions at the Ethereum Virtual Machine (EVM) level. In order to identify and stop the execution of dangerous transactions at the EVM level, they take into account

the use of graph neural networks (GNNs) in deep learning in addition to the traditional expert models provided for vulnerabilities. Comparison done between algorithms like Vanilla-RNN, LSTM, GRU, GCN, GNN+expert knowledge. The accuracy of GNN+expert knowledge for reentrancy is 89.74, timestamp dependency is 88.52, code injection is 88.62, and call with hardcoded gas amount is 90.62, respectively. GNN+expert knowledge has high performance[11].

Method's general architecture is divided into three phases:

1. A graph generation phase that models the fallback mechanism directly and extracts the control flow and data flow semantics from the source code.
2. A graph normalization step that draws inspiration from the k-partite graph.
3. New message propagation networks for modeling and identifying vulnerabilities.

Three types of nodes—major, secondary, and fallback—are extracted by them. In order to represent the complex semantic relationships among nodes, they create four different kinds of edges: data flow, forward, fallback, and control.Each subsidiary node Si is eliminated, but its feature is transferred to the major node that is closest to it.To handle the normalized graphs, they extend GCN to a degree-free GCN (DR-GCN). To automatically identify smart contract vulnerabilities, they present a unique temporal message propagation network (TMP) and a degree-free GCN (DR-GCN). When function A calls function B with the wrong parameters, this contract's fallback function will automatically start running. There are three components to the feature of Vi: i) Self-feature, namely major node Mi's feature ii) in-features, that is, characteristics of the merged secondary nodes {Pj} |P | j=1 with a path pointing from Pj to Mi iii) Out-feature, or features of the merged secondary nodes {Qk} |Q| k=1 with a path directing from Mi to Qk. After that, they remove matrix Dˆ from the equation, keeping in mind that the graph is already well-normalized in our setting. When compared to our DR-GCN and TMP networks, Vanilla-RNN, LSTM, GRU, and GCN exhibit the lowest accuracy[12].

Upon first observation, they realize that various program components inside a function are not equally crucial for identifying vulnerabilities. Consequently, they take out three different kinds of nodes: backup, normal, and core. Control flow, data flow, and backup edges are the three types of edges they create in order to capture rich semantic dependencies between the nodes. To be more precise, the feature of an edge is retrieved as a tuple (Vstart, Vend, Order, Type), where Vstart and Vend stand for the edge's start and end nodes, Order signifies its temporal order, and Type indicates the edge type. To integrate the pattern feature Pr and the graph feature Gr and produce the detection results, they employ a fusion network. Following the extraction of the security pattern feature Pr, they employ our suggested temporal-message-propagation network—which consists of a message propagation phase and a readout phase—to further retrieve the semantic feature of the contract graph. Vanilla-RNN: A two-layer recurrent neural network that gathers input in the form of the code sequence and uses recurrent evolution of its hidden states to extract the sequential pattern contained inside. LSTM: The recurrent neural network that processes sequential data the most frequently utilized. Long short term memory, or LSTM for short, reads the code sequence a series of times and then periodically updates the cell state. GRU: The gated recurrent unit, which manages the code sequence through gating methods.

GCN: Graph Convolutional Network: This network uses the contract graph as input and applies graph Laplacian to perform layer-wise convolution on the graph. TMP: The temporal message propagation network, which gathers information by progressively moving along the edges in the sequence in which they occur to understand the contract graph feature. The last feature of the graph is utilized to predict vulnerabilities. Here CGE has highest accuracy by 89.15 in retrancy vulnerability[13][14].

3. Results and Discussion

The results are gathered based on reentrancy vulnerability only.

Table 1: Accuracy for reentrancy vulnerability [8,13,4]

Algo/Parameter	Accuracy			Recall			Precision			F1		
Vanilla_RNN	49.64	49.64	49.64	58.78	58.78	58.78	49.82	49.82	49.82	50.71	50.71	50.71
LSTM	53.68	53.68	53.68	67.82	67.82	67.82	51.65	51.65	51.65	58.64	58.64	58.64
GRU	54.54	54.54	54.54	71.30	71.30	71.30	53.10	53.10	53.10	60.87	60.87	60.87
TMP	-	84.48	84.48	-	82.63	82.63	-	74.06	74.06	-	78.11	78.11

Table 2: Accuracy for reentrancy vulnerability ratings

Model	Accuracy	Recall	Precision	F1-score
CNN	0.71	0.756	0.733	0.745
XCEPTION	0.69	0.729	0.840	0.781
EfficientNet B2	0.75	0.740	0.750	0.740

Table 1 is considering their ratings where table 2 considers our ratings. The three distinct Deep Learning models whose performance is summed up in the comparison table. CNN obtains a reasonable accuracy of 0.71, while XCEPTION has a slightly lower accuracy of 0.69 but a better precision of 0.840. Notably, EfficientNet B2 performs better than both models, with balanced precision, recall, and F1-score values of approximately 0.74 and the maximum accuracy of 0.75. According to these findings, EfficientNet B2 might perform better in classification tasks than the other models, highlighting the significance of choosing the right model architecture for a given set of application needs. Here the accuracy of Vanilla-RNN, LSTM, GRU is low where TMP method is high i.e 84.48 percent. Methodology used for our model is first take .sol file and convert it into bytecode using compile_standard library. Then create a image from that bytecode by converting the bytecode into array and by padding it, as using Python Imaging Library final converts the bytecode into image. Finally use our deeplearning models on it.

4. Conclusion

As a result of our examination, we have found some algorithms and approaches for efficiently identifying vulnerabilities in smart contracts. We are prepared to create our model by utilizing this valuable information set, giving it a different identity. The major goal of this research study is to undertake an analysis of the literature in order to guide for beginners who are looking to identify the techniques in this field. We come with new deep learning algorithms like CNN, Xception, EfficientNet-B2 model that have higher accuracy then LSTM, Vanilla-RNN,GRU..

REFERENCE

[1] Mojtaba Eshghie, Cyrille Artho, and Dilian Gurov. 2021. Dynamic Vulnerability Detection on Smart Contracts Using Machine Learning. In Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering (EASE '21). Association for Computing Machinery, New York, NY, USA, 305–312. Available from: https://doi.org/10.1145/3463274.3463348

[2] Mezina, A.; Ometov, A. Detecting Smart Contract Vulnerabilities with Combined Binary and Multiclass Classification. Cryptography 2023, 7, 34. Available from: **https://doi.org/10.3390/cryptography7030034**

[3] Lakshminarayana, K. & Sathiyamurthy, K.. (2022). Towards Auto Contract Generation and Ensemble-based Smart Contract Vulnerability Detection. International journal of electrical and computer engineering systems. Available from: https://doi.org/10.32985/ijeces.13.9.3

[4] Hasan, Qusai Omar Mustafa, "Machine Learning Based Framework for Smart Contract Vulnerability Detection in Ethereum Blockchain" (2023). Thesis. Rochester Institute of Technology. Available from: https://repository.rit.edu/theses/11469

[5] Sosu RNA, Chen J, Brown-Acquaye W, Owusu E, Boahen E. A Vulnerability Detection Approach for Automated Smart Contract Using Enhanced Machine Learning Techniques. Research Square; 2022. Available from: https://doi.org/10.21203/rs.3.rs-1961251/v1

[6] K. Lakshmi Narayana, K. Sathiyamurthy, Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning, Materials Today: Proceedings, Volume 81, Part 2, 2023, Pages 653-659, ISSN 2214-7853. Available from: https://doi.org/10.1016/j.matpr.2021.04.125

[7] P. Momeni, Y. Wang and R. Samavi, "Machine Learning Model for Smart Contracts Security Analysis," 2019 17th International Conference on Privacy, Security and Trust (PST), Fredericton, NB, Canada, 2019, pp. 1-6. Available from: DOI: 10.1109/PST47121.2019.8949045

[8] Z. Liu, M. Jiang, S. Zhang, J. Zhang and Y. Liu, "A Smart Contract Vulnerability Detection Mechanism Based on Deep Learning and Expert Rules," in IEEE Access, vol. 11, pp. 77990-77999, 2023. Available from: Digital Object Identifier 10.1109/ACCESS.2023.3298048

[9] S. -J. Hwang, S. -H. Choi, J. Shin and Y. -H. Choi, "CodeNet: Code-Targeted Convolutional Neural Network Architecture for Smart Contract Vulnerability Detection," in IEEE Access, vol. 10, pp. 32595-32607, 2022. Available from: DOI:10.1109/ACCESS.2022.3162065

[10] Wang, W., Song, J., Xu, G., Li, Y., Wang, H., & Su, C. (2021). ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts. IEEE Transactions on Network Science and Engineering, 8(2), 1133–1144. Available from: DOI:10.1109/TNSE.2020.2968505

[11] P. Qian, Z. Liu, Q. He, R. Zimmermann and X. Wang, "Towards Automated Reentrancy Detection for Smart Contracts Based on Sequential Models," in IEEE Access, vol. 8, pp. 19685-19695, 2020. Available from: DOI:10.1109/ACCESS.2020.2969429

[12] Deng, W.; Wei, H.; Huang, T.; Cao, C.; Peng, Y.; Hu, X. Smart Contract Vulnerability Detection Based on Deep Learning and Multimodal Decision Fusion. Sensors 2023, 23, 7246. Available from: **https://doi.org/10.3390/s23167246**

[13] Zhuang, Yuan & Liu, Zhenguang & Qian, Peng & Liu, Qi & Wang, Xiang & He, Qinming. (2020). Smart Contract Vulnerability Detection using Graph Neural Network. 3255-3262. Available from: https://doi.org/10.24963/ijcai.2020/454

[14] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu and X. Wang, "Combining Graph Neural Networks With Expert Knowledge for Smart Contract Vulnerability Detection," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 2, pp. 1296-1310, 1 Feb. 2023. Available from: 10.1109/TKDE.2021.3095196