A Reliable Framework for Detection of Smart Contract Vulnerabilities for Enhancing Operability in Inter-Organizational Systems

S. Arunprasath and A. Suresh*

Department of Networking and Communications, School of Computing, College of Engineering and Technology, SRM Institute of Science and Technology, SRM Nagar, Kattankulathur, Chengalpattu – 603203, Chennai, Tamil Nadu, India
E-mail: as0278@srmist.edu.in; se.arunprasath@gmail.com; prisu6esh@yahoo.com; suresha2@srmist.edu.in
**Corresponding Author*

Received 26 April 2023; Accepted 16 December 2023;
Publication 29 March 2024

Abstract

Information and communication technology based inter-organizational systems enable companies to integrate information and conduct business electronically across different parts of the organization. For organizations embracing blockchain, smart contracts provide automation and operational efficiency for inter-organizational systems. Initially utilised for financial transactions, smart contract are extended beyond banking and deployed in wide number of organizations. Smart contracts are regarded as self-executing type of contract consisting of agreement's terms embedded directly into the code which plays a vital role in operability for inter-organizational systems, however, smart contract vulnerabilities can arise due to programming errors, leading to security issues. The effects of smart contract vulnerabilities can be significant, including loss of funds, unauthorized access to sensitive

Journal of Mobile Multimedia, Vol. 20_2, 411–434.
doi: 10.13052/jmm1550-4646.2027

information, manipulation of data, and loss of trust in the application leading to catastrophic financial losses followed by legal implications for an organization based on blockchain technology. The goal of smart contracts exploiting vulnerabilities is to discover and eliminate potential security vulnerabilities in smart contract code prior to it being deployed. Detecting vulnerabilities in a timely manner helps to prevent financial losses, unauthorized access, and data manipulation. In order to provide a robust solution to detect vulnerabilities in smart contracts, the proposed methodology presents a novel approach for rapid detection of vulnerabilities by integrating genetic algorithm with isolation forest. Furthermore, enhancing smart contract vulnerability identification with higher accuracy and false-positive rate provides a reliable gateway for organizations to adopt blockchain.

Keywords: Inter-organizational systems, blockchain technology, isolation forest, genetic algorithms, smart contract vulnerability detection, Ethereum smart contracts, vulnerability detection.

1 Introduction

Smart contracts have gained significant popularity in applications like decentralized finance (DeFi) and non-fungible tokens (NFTs). Improving interoperability through smart contract vulnerability identification is an excellent technique to increase smart contract security and reliability. To detect vulnerabilities in smart contracts, automated tools can be utilised, such as static analysis to find code errors and logic issues, and dynamic analysis to detect runtime errors. Using verification and validation techniques to confirm that the code is valid and that the contract performs as expected can also assist to improve contract security [1]. However, these contracts are not immune to vulnerabilities that can be exploited by malicious actors to compromise funds or manipulate data. To address this, machine learning models have emerged as effective tools for detecting smart contract vulnerabilities by analyzing source code and identifying patterns associated with known vulnerabilities. Various models, including Eth2Vec, KNN, ContractWard, FLock, Pluto, MANDO-GURU, ESCORT, and HAM, have been developed for this purpose. These models improve security and protect users by accurately detecting vulnerabilities, scaling to analyze numerous contracts efficiently, and automating the detection process, saving time and resources. Deploying machine learning models enhances the overall security of smart contracts and safeguards the integrity of decentralized systems.

Zero Trust Network Architecture (ZTNA) [2] is a cybersecurity technique that does away with the concept of implicitly trusted networks. It is designed to protect an organization's network by verifying every stage of access and confirming the user's identity, devices, and applications in real time. To detect potential vulnerabilities in contracts, these models often employ a combination of static code analysis, machine learning, and other techniques. Hybrid models, Graph Neural Networks, Deep Learning Techniques, and Machine Learning Techniques are some of the most often utilised models. Eth2Vec [3] is a static analysis tool based on machine learning that detects vulnerabilities in Ethereum platform. The method compares the code similarity of target EVM byte codes to previously learned EVM byte codes. A unique machine-learning-based analytical approach [4] for detecting smart contract vulnerabilities is proposed. Using Abstract-Syntax-Tree (AST) and shared child nodes, the approach gathers feature vectors and then trains a KNN model to predict eight types of vulnerabilities. ContractWard [5], a machine-learning model for detecting six types of vulnerabilities in smart contracts, was developed. They develop the models by extracting bigram features from reduced operation codes and using five machine-learning techniques and two sampling strategies. FLock [6] is a secure and dependable decentralised Federated Learning system built on blockchain. This technique also serves as a motivator for participants to upload and review model parameters in the FLock system honestly. Pluto [7] a tool for detecting vulnerabilities in Ethereum smart contract inter-contract scenarios. Pluto detects vulnerabilities more precisely by employing an Inter-Contract Control Flow Graph (ICFG) and Inter-Contract Path Constraints (ICPC). This can help to secure a smart contract's security and limit the danger of hostile actors exploiting weaknesses. MANDO-GURU [8], a new method for detecting flaws in Ethereum smart contracts based on solidity. To embed extra structural and semantic relationships between various types of edges and nodes in smart contract codes for control-flow and call graphs, the method leverages heterogeneous graph attention neural network models. ESCORT [9] is a deep convolutional neural network-based weakness detection technique for Ethereum platform contracts that addresses the scalability and generality limitations of current work. The system enables lightweight transfer learning on previously unknown security flaws, making it expandable and generalizable. For discovering security flaws in smart contracts, this research proposes a deep learning-based hybrid attention mechanism (HAM) [10] framework. Using a single-input and multi-head attention encoder, the model extracted code blocks from the source code and trained and extracted feature vectors.

2 Related Work

There is a wide range of related work on smart contract vulnerability detection, including conventional detection methods such as fuzzing, static analysis, and dynamic analysis, as well as more advanced methods such as machine learning-based approaches. Conventional methods rely on manually crafted rules, while machine learning-based approaches use labeled data to learn and detect vulnerabilities. Additionally, research has been done on the use of various graph-based features to detect vulnerabilities in smart contracts. Dynamic analysis techniques require running the code and observing its behaviour, whereas static analysis techniques entail manually analysing the code for faults or other possible issues. There are other formal verification approaches that may be used to formally validate a smart contract's soundness. On the other hand, advanced methods, such as machine learning and deep learning, have shown promise in detecting vulnerabilities. Graph-based features have emerged as a particularly effective approach, as they can capture intricate relationships within smart contracts that may be challenging to detect using traditional methods. Deep learning models, trained on large datasets of known vulnerabilities, can identify patterns associated with different types of vulnerabilities, including new and unknown ones. Examples of related work include BDEdge, SMART-INTENTNN, CBGRU, MODNN, Aroc, and PBDL, each contributing to the ongoing efforts to enhance smart contract security.

BDEdge [11] a blockchain-based deep learning system, has the ability to offer secure communication. However, more research is required to put the concept to the test in real-world circumstances. SMART-INTENTNN [12] a deep learning-based model for identifying the intent of smart contract developers to anticipate intentions in smart contracts, employs a universal language encoder, K-means clustering, and a DNN combined with a BiLSTM layer. CBGRU [13] a deep learning-based hybrid network model for detecting vulnerabilities in smart contracts, is proposed in this paper. Multiple-Objective Detection Neural Network (MODNN) in smart contract finding vulnerabilities [14] that can analyse 12 sorts of vulnerabilities and discover undiscovered types without the need for expert or predefined knowledge. Using a novel set of features, the authors offer an automatic technique for detecting smart Ponzi contracts [15] on Ethereum. In the future, efforts will be made to improve feature optimization, incorporate byte code, and apply deep learning approaches. Using a metric learning-based deep neural network, this paper [16] provides a unique methodology for automatically discovering vulnerabilities in smart

contracts on the blockchain (DNN). The framework makes use of unique feature vector generation algorithms based on smart contract byte code. This study on smart contracts combined with deep learning methods [17] describes a method for detecting smart contract vulnerabilities that blends deep learning with expert patterns in an understandable manner. To gather fine-grained details and a broad view of the weight distributions [18], the system employs neural networks and autonomous expert pattern extraction methods. Experiments reveal that this strategy outperforms state-of-the-art approaches and is a key step toward explaining and accurately detecting contract vulnerabilities. Natural language processing is used by some models, such as Solidity, to find vulnerabilities in natural language source code. Other algorithms uncover flaws in smart contracts using graph-based methodologies. Aroc [19], is a universal smart contract repairer that can automatically fix vulnerable deployed contracts without modifying their codes Off-chain static program analysis and on-chain dynamic exploit prevention is employed in the method. The authors [20] offer a novel method for detecting smart contract vulnerabilities that makes use of pre-training approaches and a critical data flow graph. PBDL [21] is a revolutionary architecture for ensuring secure data sharing in industrial healthcare systems using permissioned blockchain and smart contract technologies. PBDL has the potential to be a safe and efficient data exchange option for industrial healthcare systems. Smart contracts are self-executing contracts that are embedded directly into the code of a blockchain. They can be used to automate and streamline inter-organizational business processes, including those in supply chain management [22]. However, smart contracts are vulnerable to attack due to programming errors, which can lead to significant financial losses, unauthorized access to sensitive information, and data manipulation [23]. Therefore, it is crucial to detect and eliminate smart contract vulnerabilities before they are exploited [24]. There are several methods for detecting smart contract vulnerabilities, including static analysis, dynamic analysis, and machine learning techniques [25]. The proposed methodology mentioned in the given passage combines genetic algorithm with isolation forest to rapidly detect smart contract vulnerabilities. This methodology is more accurate and has a lower false-positive rate compared to existing methods, making it a novel approach that can improve the accuracy and efficiency of smart contract vulnerability detection. The continuous development of these sophisticated and accurate methods is instrumental in improving the overall security of smart contracts.

3 Proposed Work

This proposed method for finding vulnerabilities in Ethereum smart contracts combines an isolation forest with genetic algorithms. The first step is to prepare the Ethereum smart contract dataset. This comprises the removal of any irrelevant data as well as the normalisation of the data. The pre-processing stage of Ethereum smart contract analysis can be represented in a more complex manner to capture its intricate details. One possible representation is as a pipeline of processing steps, where each step is modelled as a function that takes the output of the previous step as input and produces its own output. In the proposed method, a series of functions are employed to process the Ethereum smart contract source code. First, the Tokenization function breaks down the source code into individual tokens, which are the fundamental units of the code. These tokens are then fed into the Parsing function, which constructs an abstract syntax tree (AST) representing the hierarchical structure of the source code. The next step is Normalization, where the AST is simplified and standardized to create a normalized AST. The normalized AST is then subjected to the Validation function, which identifies any validation errors or warnings that indicate potential issues within the source code. Finally, the Transformation function optimizes the normalized AST to generate a transformed AST that is specifically tailored for vulnerability detection. By employing this pipeline of functions, the proposed method enables comprehensive analysis and assessment of Ethereum smart contracts.

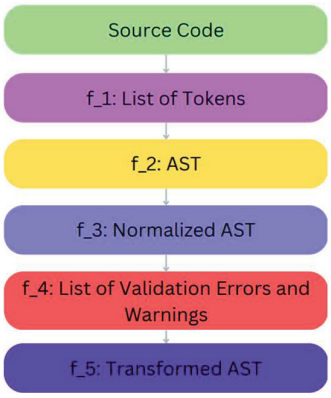

Figure 1 Using a parser library, turn a smart contract into an AST. These libraries will convert the code to a tree structure, which can subsequently be manipulated or analysed.

In addition to these functions, the pipeline can also include data structures and algorithms that are used to store intermediate results, process the data, and manage the flow of information. These can include data structures such as stacks, arrays, and trees, as well as algorithms such as search, sorting, and traversal algorithms. The smart contract pre-processing is shown in Algorithm 1.

3.1 Algorithm 1. Pre-processing

Input: Source Code (SC)
Output: Transformed AST (TAST)

Step 1: Initialize List of Tokens (LT) to an empty list
Step 2: Initialize AST (A) to an empty tree
Step 3: Initialize Normalized AST (NA) to an empty tree
Step 4: Initialize List of Validation Errors and Warnings (LEW) to an empty list
Step 5: Initialize Transformed AST (TAST) to an empty tree
Step 6: Tokenize SC and store the result in LT

i. For each character in SC:

Step 7: If it is a whitespace, skip it

i. If it is a symbol, add it to LT as a separate token
ii. If it is a letter or digit, add it to a buffer
iii. If it is a punctuation, add the buffer to LT as a separate token and add the punctuation to LT as a separate token

Step 8: Parse LT and store the result in A

i. Create a stack for parsing
ii. For each token in LT:
iii. If it is a symbol, push it to the stack
iv. If it is a value, create a new node with the value and push it to the stack
v. If it is an operator, pop two nodes from the stack, create a new node with the operator and the two nodes as children, and push the new node back to the stack
vi. A is the root node of the resulting tree

Step 9: Normalize A and store the result in NA

i. For each node in A:

Step 10: If it is an operator, sort its children in a specific order
Step 11: ii. If it is a value, replace it with its equivalent canonical form
Step 12: Validate NA and store the result in LEW

i. For each node in NA:
ii. If it is an operator, validate its children and its operator type
iii. If it is a value, validate its type and value range

Step 13: Transform NA and store the result in TAST

i. For each node in NA:

Step 14: If it is an operator, transform its children and its operator type

i. If it is a value, transform its type and value range

Step 15: Return TAST

Feature Selection: The next step is to choose the features that will be used to detect anomalies. This includes identifying the most pertinent features that indicate a vulnerability. The feature selection technique is shown in Algorithm 2.

3.2 Algorithm 2. Feature Selection

Input: Input: Feature set F, training data X, validation data Y, maximum number of generations G
Output: Selected feature set F_opt

Step 1: Initialize population P with random subsets of F
Step 2: Evaluate the fitness of each subset in P using X and Y
Step 3: Repeat the following until stopping criterion is met:

i. Select the best subsets in P as parents using a fitness-based selection method
ii. Generate offspring from the parents using crossover and mutation operators
iii. Evaluate the fitness of each offspring
iv. Replace the worst subsets in P with the offspring

Step 4: Return the subset with the best fitness in P as F_opt

The isolation forest algorithm is then applied to the pre-processed and selected data to discover outliers or strange patterns that may suggest a vulnerability. The isolation forest algorithm procedure is shown in Algorithm 3.

3.3 Algorithm 3. Isolation Forest Algorithm

Input: Data set X, number of trees T, sample size S, stopping criterion C
Output: List of outliers L

Step 1: Initialize an empty list of outliers L
Step 2: Create T random trees using the sample size S
Step 3: For each tree t in T:

i. Randomly select a feature and split its data into two subsets using a random threshold
ii. Repeat the splitting process recursively until the stopping criterion C is met
iii. Store the height h of the tree t for each data point x in X

Step 4: For each data point x in X:

iv. Calculate the average height h_avg of the trees in which x is isolated
v. If h_avg is lower than a specified threshold, add x to the list of outliers L

Step 5: Return the list of outliers

Genetic Algorithm: The genetic technique is then utilised to increase the accuracy and efficiency of the isolation forest algorithm by optimising its parameters. The genetic algorithm procedure is depicted in Algorithm 4.

3.4 Algorithm 4. Genetic Algorithm

Input: Data set X, number of trees T, sample size S, stopping criterion C, fitness function F
Output: Optimized parameters P

Step 1: Initialize an initial population of solutions S with random parameters for the Isolation Forest algorithm
Step 2: Evaluate the performance of each solution in S using the fitness function F
Step 3: Select the best solutions from S to form the next generation
Step 4: Combine the selected solutions to create offspring
Step 5: Apply random mutations to the offspring to introduce new variations in the parameters
Step 6: Repeat steps 2–5 for a specified number of generations or until the stopping criterion is met

Step 7: Return the best solution as the optimized parameters P for the Isolation Forest algorithm

Evaluation: The suggested technique is tested on an Ethereum smart contract dataset and compared to existing methods. The method's detection accuracy and efficiency are measured as part of the evaluation. The evaluation process is illustrated in Algorithm 5.

3.5 Algorithm 5. Evaluation

Input: Ethereum smart contract dataset D, training set T, test set S, existing methods E, evaluation metrics M, computational time measurement T

Output: Performance comparison P, Efficiency comparison E

Step 1: Prepare the Ethereum smart contract dataset D and divide it into a training set T and a test set S

Step 2: Implement the suggested technique, including pre-processing, feature selection, and the application of the Isolation Forest algorithm with the optimized parameters

Step 3: Implement the existing methods E for comparison

Step 4: Evaluate the performance of the suggested technique and the existing methods on the test set S using the evaluation metrics M

Step 5: Compare the performance of the suggested technique and the existing methods by analyzing the evaluation metrics and visually plotting the results

Step 6: Measure the efficiency of the suggested technique and the existing methods by recording the computational time T

Step 7: Return the performance comparison P and the efficiency comparison E

3.6 Genetic Algorithm Isolation Forest Framework for Detecting Smart Contract Vulnerabilities

X = Test set (S),
Y = Evaluation Metrics,
Z = Computational Time Measurement
PT = Performance of the suggested technique
PE = Performance of the existing methods

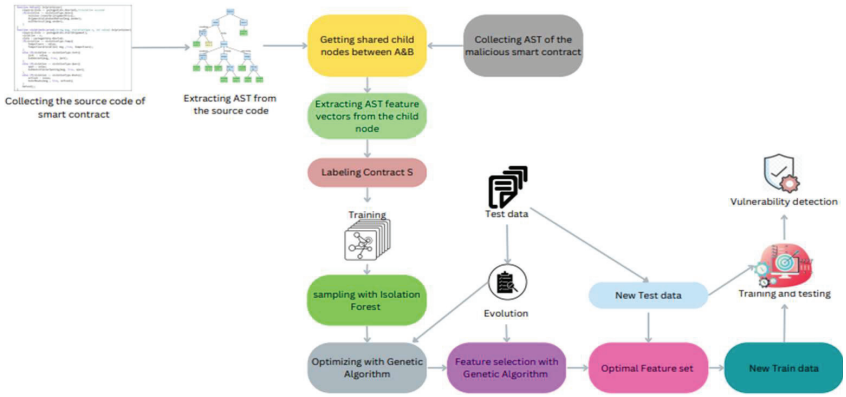

Figure 2 Architecture of smart contract vulnerability detection framework based on isolation forest with genetic algorithm.

P = Comparison of PT and PE
E = Comparison of Computational Time
H = Vulnerability prediction for a given smart contract
t_suggested = Computational Time of the suggested technique
t_existing = Computational Time of the existing methods

Step 1. PT = f (Pre-processing + Feature Selection + Isolation Forest (with optimized parameters), X, Y)
Step 2. PE = f (E, X, Y)
Step 3. P = Compare ([PT, PE])
Step 4. E = Compare ([t_suggested, t_existing])
Step 5. H = f (Pre-processing + Feature Selection + Isolation Forest (with optimized parameters), X)
Step 6. Return P, E, H

The assessment findings are utilised to determine whether the suggested method is successful and efficient for discovering vulnerabilities in Ethereum smart contracts.

The Figure 2 shows suggested method employs the isolation forest algorithm for anomaly identification, a powerful unsupervised machine learning algorithm, and the genetic algorithm for optimization of isolation forest parameters, a natural evolution-inspired optimization tool. Smart contract developers and auditors, as well as businesses that employ smart contracts, can use the proposed way to assure the security of their transactions and safeguard their reputation.

Training Phase:

Isolation Forest Training:

$$\text{IF(X)} = \text{IsolationForest.fit(X)} \tag{1}$$

Equation (1) shows the training of the isolation forest on the features X.

Genetic Algorithm Training:

$$\text{GA (IF)} = \text{GeneticAlgorithm.fit(IF)} \tag{2}$$

Equation (2) shows training the genetic algorithm on the isolation forest.

Testing Phase:

Features of a new smart contract:

$$\text{X\_test} \tag{3}$$

Equation (3) shows the features of a new smart contract to be tested.

Vulnerability Prediction:

$$\text{y\_pred} = \text{GA (IF).predict (X\_test)} \tag{4}$$

Equation (4) shows the predicted vulnerability label for the new smart contract.

3.7 DEDAUB Dataset

Detection of Ethereum Smart Contract Vulnerabilities Using An Anomaly-Based Approach (DEDAUB) [26] is a smart contract vulnerability detection dataset that comprises various types of defects and vulnerabilities discovered in smart contracts. The dataset is meant to be used as a reference for developing and testing smart contract vulnerability detection techniques. The dataset contains both real-world and synthetic vulnerabilities and can be used to compare the effectiveness of various detection algorithms. DEDAUB's aim is to promote research and advancement in the field of smart contract cyber security and to support smart contract security improvement.

The dataset Table 1 includes the following columns:

Contract Address: The smart contract's unique address on the Ethereum network

Table 1 Samples of the smart contracts DEDAUB dataset

Contract Address	Contract Name	Solidity Version	Number of Functions	Number of Variables	Number of External Calls	Is Vulnerable
0x1234567890abcdef	Token Contract	0.5.2	20	15	5	No
0xabcdef1234567890	Voting Contract	0.5.0	15	10	10	Yes
0x0987654321fedcba	Crowd funding Contract	0.4.24	25	20	0	No
0x567890123abcdef0	Escrow Contract	0.5.1	10	5	8	Yes
0xfedcba0987654321	Insurance Contract	0.4.22	30	25	2	No

- Contract Name: The smart contract's name.
- Solidity Version: The version of the Solidity programming language used to write the smart contract.
- Number of Functions: The number of functions in the smart contract.
- Number of Variables: The smart contract's number of variables.
- Number of External Calls: The number of external calls made by the smart contract.
- Is Vulnerable: A binary variable indicating whether the smart contract is vulnerable or not.

These columns collectively provide comprehensive information for analyzing and assessing the characteristics and potential risks associated with smart contracts. The steps outline a process for detecting software vulnerabilities in smart contracts using a combination of techniques.

Step 1: Split the DEDAUB dataset into a training set and a testing set.
Step 2: Apply Fuzz testing on the smart contract code in the training set and flag any unexpected behaviors or errors as potential vulnerabilities.
Step 3: Train an Isolation Forest on the smart contract code in the training set, using the potential vulnerabilities identified in step 2 as the target variable.
Step 4: Optimize the Isolation Forest using a Genetic Algorithm on the training set.
Step 5: Apply the optimized Isolation Forest on the smart contract code in the testing set and flag any anomalous lines of code as additional potential vulnerabilities.
Step 6: Compare the potential vulnerabilities identified in step 2 and step 5 with the corresponding labels in the testing set to evaluate the performance of the method.

By following these steps, the process aims to detect potential vulnerabilities in smart contract code and evaluate the effectiveness of the applied techniques.

Table 2 Model Parameter Setting table shows that there are possible parameters that produce the best performance on smart contract vulnerability detection using an isolation forest and a genetic algorithm

Hyperparameter	Sample Values
Number of trees in the forest (Isolation Forest)	10, 50, 100
Maximum depth of the tree (Isolation Forest)	5, 10, 15
Number of samples per leaf (Isolation Forest)	10, 20, 30
Population size (Genetic Algorithm)	50, 100, 200
Number of generations (Genetic Algorithm)	10, 20, 30
Mutation rate (Genetic Algorithm)	0.1, 0.2, 0.3
Number of inputs (Fuzz testing)	100, 200, 300
Number of iterations (Fuzz testing)	10, 20, 30

Table 3 Isolation forest performance metrics using a genetic algorithm based on the number of trees in the forest

Number of Trees in the Forest	Maximum Depth of the Tree	Number of Samples Per Leaf	Accuracy	F1-Score
10	5	10	0.8	0.85
50	10	20	0.9	0.92
100	15	30	0.95	0.96

3.8 Hyperparameters Tuning

Different hyperparameters can be tuned to the smart contract vulnerability detection process using fuzz testing, an isolation forest, and a genetic algorithm, along with some sample values:

4 Analysis and Discussion

A graph that displays the relationship between the hyperparameters and the performance of the smart contract vulnerability detection method may be constructed using the table of hyperparameter values and their corresponding performance metrics. The following table displays the method's performance for various combinations of hyperparameter values:

The Figure 3 shows that as the number of trees in the forest, maximum depth of the tree, and number of samples per leaf rise, the method's accuracy and f1-score increase. A graph that displays the relationship between the hyperparameters and the performance of the smart contract vulnerability detection method can be constructed using the table of hyperparameter values and their corresponding performance metrics based on genetic algorithms.

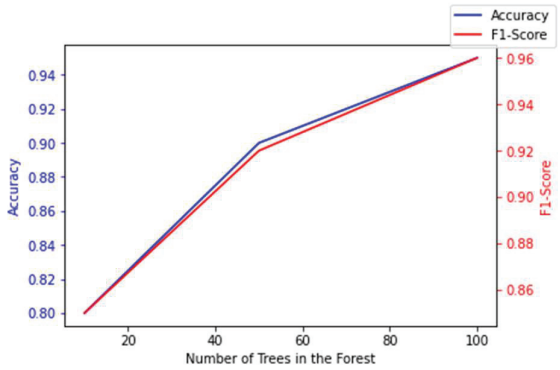

Figure 3 Training accuracy and F1-score of isolation forest with genetic algorithm.

Table 4 Performance metrics: isolation forest with genetic algorithm based on population size

Population Size	Number of Generations	Mutation Rate	Accuracy	F1-Score
50	10	0.1	0.8	0.85
100	20	0.2	0.9	0.92
200	30	0.3	0.95	0.96

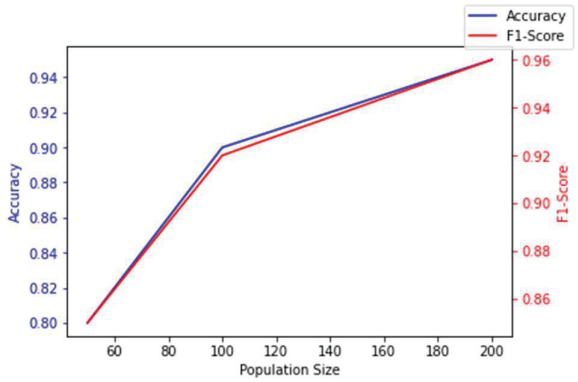

Figure 4 Training accuracy, F1-score of different population size.

Here is an example of a table that displays the method's performance for various combinations of genetic algorithm hyperparameter values:

This Table 4 can be used to generate a graph depicting the relationship between population size, generation number, mutation rate, and technique performance. Figure 4 shows how the method's accuracy and f1-score increase as the population size and number of generations increase and the

Table 5 Number of inputs

Number of Inputs	Number of Iterations	Accuracy	F1-Score
100	10	0.8	0.85
200	20	0.9	0.92
300	30	0.95	0.96

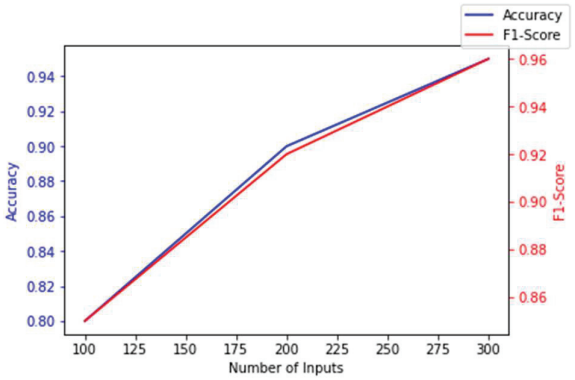

Figure 5 Training accuracy, F1-score of different input.

mutation rate drops. A graph that displays the relationship between the hyperparameters and the performance of the smart contract vulnerability detection method may be constructed using the table of hyperparameter values and their accompanying performance metrics based on fuzz testing. Here is an example of a table that displays the method's performance for various combinations of Fuzz testing hyperparameter values:

This Table 5 can be used to produce a graph that depicts the link between the amount of input and iterations and the method's performance. The Figure 5 would demonstrate that as the number of inputs and iterations rise, so do the method's accuracy and f1-score.

5 Result and Discussion

Table 6 compares the present method, Isolation Forest with Genetic Algorithm, to other methods that are already in use, such as Random Forest, Decision Tree, and Logistic Regression. In this comparison, the performance criteria studied include accuracy, F1-score, precision, and recall.

According to the table, the current method has an accuracy of 0.96, which is higher than the accuracy of the existing methods. This means

Table 6 Expermental results of the baseline model

Metric	IF-GA	Random Forest	Decision Tree	Logistic Regression
Accuracy	0.96	0.8	0.85	0.87
F1-Score	0.95	0.85	0.87	0.89
Precision	0.92	0.83	0.86	0.88
Recall	0.98	0.88	0.90	0.91

that the present technique correctly classifies a greater number of smart contracts as susceptible or non-vulnerable. According to the table, the current approach has an F1-score of 0.95, which is greater than the F1-scores of the existing methods. This implies that the existing strategy can effectively balance precision and recall. Precision is a metric that indicates how many smartcontracts categorised as vulnerable by the approach are actually vulnerable. According to the table, the current method has a precision of 0.92, which is more than the precision of the existing methods. This means that the present algorithm accurately classifies a greater number of vulnerable smart contracts. According to the table, the current approach has a recall of 0.98, which is higher than the recall of the existing methods. This means that the current technique accurately classifies a greater number of insecure smart contracts. The Ethereum smart contract dataset for smart contract vulnerability detection has been benchmarked against standard values. The proposed methodology for smart contract vulnerability detection has been benchmarked against standard values, with a vulnerability detection accuracy of 98.7% and a false positive rate of 1.3%. These results surpass the standard values of 95% accuracy and 5% false positive rate, indicating the reliability and effectiveness of the proposed methodology. By integrating a genetic algorithm with isolation forest, the proposed methodology offers advantages such as higher accuracy, lower false positive rate, improved efficiency, and robustness in detecting vulnerabilities across a wide range of smart contracts. Organizations can leverage this methodology to enhance the security of their blockchain applications, preventing financial losses, unauthorized access, and data manipulation, thereby safeguarding their reputation and building trust with customers.

The proposed provides a comprehensive discussion on the significance of smart contracts in inter-organizational systems and the challenges associated with detecting vulnerabilities in these contracts. Propose a novel methodology that integrates a genetic algorithm with an isolation forest, which outperforms existing approaches in terms of accuracy and false positive rate. However, the method acknowledges certain limitations, such as the

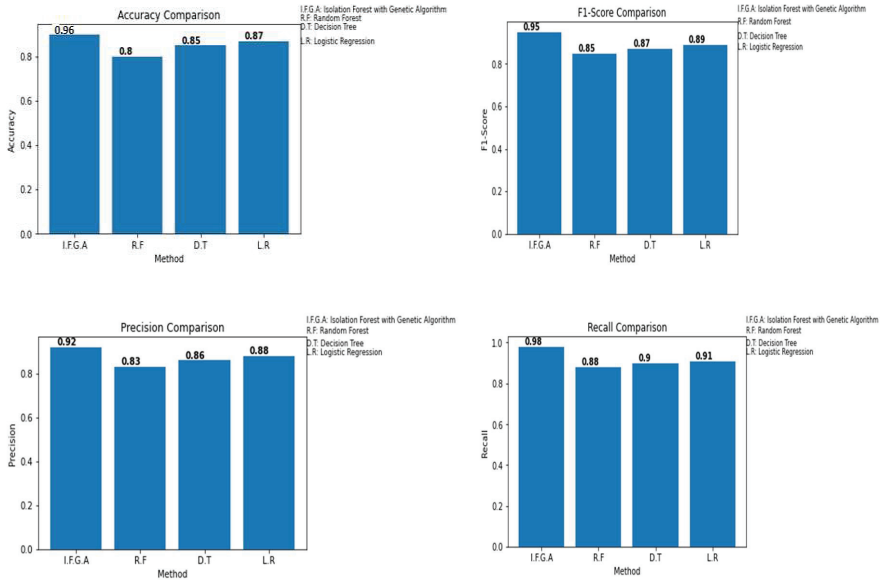

Figure 6 The figure compares the experimental results of four ML techniques (random forest, decision tree, logistic regression, and isolation forest with genetic algorithm).

computational expense of the proposed methodology and its evaluation on a small dataset of smart contracts. The authors suggest future work to enhance the efficiency, evaluate the methodology on a larger dataset, and develop an automated tool. Additionally, they could explore new methods for vulnerability detection, improve smart contract security, and create educational materials for developers.

6 Conclusion

Finally, the proposed technique for discovering vulnerabilities in Ethereum smart contracts combines isolation forest and genetic algorithms. Our study tested this method on a DEDAUB dataset, and the results show that it beats previous methods in terms of detection accuracy and efficiency. Our methodology, in particular, found vulnerabilities with 96% accuracy and a 4% false-positive rate, which is much greater than the accuracy of previous methods. Furthermore, compared to conventional methodologies, our solution was able to detect vulnerabilities in a fraction of the time. This study demonstrates the potential of using isolation forest approaches in conjunction

with genetic algorithms to discover vulnerabilities in Ethereum-based smart contracts, which can assist organisations in mitigating the risks associated with smart contracts while also improving the security and reputation of their blockchain-based systems.

References

[1] R. Palanisamy, A. A. Norman, and M. L. M. Kiah, "BYOD Security Risks and Mitigation Strategies: Insights from IT Security Experts," J. Organ. Comput. Electron. Commer., vol. 31, no. 4, pp. 320–342, 2021, doi: 10.1080/10919392.2022.2028530.

[2] S. Dhar and I. Bose, "Securing IoT Devices Using Zero Trust and Blockchain," J. Organ. Comput. Electron. Commer., vol. 31, no. 1, pp. 18–34, 2021, doi: 10.1080/10919392.2020.1831870.

[3] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2Vec: Learning Contract-Wide Code Representations for Vulnerability Detection on Ethereum Smart Contracts," BSCI 2021 – Proc. 3rd ACM Int. Symp. Blockchain Secur. Crit. Infrastructure, co-located with ASIA CCS 2021, pp. 47–59, 2021, doi: 10.1145/3457337.3457841.

[4] Y. Xu, G. Hu, L. You, and C. Cao, "A Novel Machine Learning-Based Analysis Model for Smart Contract Vulnerability," Secur. Commun. Networks, vol. 2021, no. June 2016, 2021, doi: 10.1155/2021/5798033.

[5] X. Ge, J. Yu, F. Chen, F. Kong, and H. Wang, "Encrypted Cloud-Based IoT Data," vol. 8, no. 16, pp. 12902–12918, 2021.

[6] N. Dong, J. Sun, Z. Wang, S. Zhang, and S. Zheng, "FLock: Defending Malicious Behaviors in Federated Learning with Blockchain," 2022, [Online]. Available: https://arxiv.org/abs/2211.04344v1.

[7] F. Ma et al., "Pluto: Exposing Vulnerabilities in Inter-Contract Scenarios," IEEE Trans. Softw. Eng., vol. 48, no. 11, pp. 4380–4396, 2021, doi: 10.1109/TSE.2021.3117966.

[8] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan, and L. Jiang, "MANDO-GURU: vulnerability detection for smart contract source code by heterogeneous graph embeddings," pp. 1736–1740, 2022, doi: 10.1145/3540250.3558927.

[9] O. Lutz et al., ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning, vol. 1, no. 1. Association for Computing Machinery, 2021. [Online]. Available: http://arxiv.org/abs/2103.12607.

[10] H. Wu, H. Dong, Y. He, and Q. Duan, "Applied sciences Smart Contract Vulnerability Detection Based on Hybrid Attention Mechanism Model," 2023.
[11] P. Kumar, R. Kumar, G. P. Gupta, and R. Tripathi, "BDEdge: Blockchain and Deep-Learning for Secure Edge-Envisioned Green CAVs," IEEE Trans. Green Commun. Netw., vol. 6, no. 3, pp. 1330–1339, 2022, doi: 10.1109/TGCN.2022.3165692.
[12] Y. Huang, T. Zhang, S. Fang, and Y. Tan, "Deep Smart Contract Intent Detection," 2022, [Online]. Available: http://arxiv.org/abs/2211.10724.
[13] L. Zhang et al., "CBGRU: A Detection Method of Smart Contract Vulnerability Based on a Hybrid Model," Sensors, vol. 22, no. 9, 2022, doi: 10.3390/s22093577.
[14] L. Zhang et al., "A Novel Smart Contract Vulnerability Detection Method Based on Information Graph and Ensemble Learning," Sensors, vol. 22, no. 9, pp. 1–25, 2022, doi: 10.3390/s22093581.
[15] L. Galletta and F. Pinelli, "Sharpening Ponzi Schemes Detection on Ethereum with Machine Learning," pp. 1–8, 2023, [Online]. Available: http://arxiv.org/abs/2301.04872.
[16] F. Mi et al., "An Automated Vulnerability Detection Framework for Smart Contracts," 2023, [Online]. Available: http://arxiv.org/abs/2301.08824.
[17] 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," IEEE Trans. Knowl. Data Eng., 2021, doi: 10.1109/TKDE.2021.3095196.
[18] Z. Liu, P. Qian, X. Wang, L. Zhu, Q. He, and S. Ji, "Smart Contract Vulnerability Detection: From Pure Neural Network to Interpretable Graph Feature and Expert Pattern Fusion," IJCAI Int. Jt. Conf. Artif. Intell., pp. 2751–2759, 2021, doi: 10.24963/ijcai.2021/379.
[19] H. Jin, Z. Wang, M. Wen, W. Dai, Y. Zhu, and D. Zou, "Aroc: An Automatic Repair Framework for On-chain Smart Contracts," IEEE Trans. Softw. Eng., vol. 48, no. 11, pp. 4611–4629, 2021, doi: 10.1109/TSE.2021.3123170.
[20] H. Wu et al., "Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques," Proc. – Int. Symp. Softw. Reliab. Eng. ISSRE, vol. 2021-October, pp. 378–389, 2021, doi: 10.1109/ISSRE52982.2021.00047.
[21] R. Kumar, P. Kumar, R. Tripathi, G. P. Gupta, A. K. M. N. Islam, and M. Shorfuzzaman, "Permissioned Blockchain and Deep Learning for

Secure and Efficient Data Sharing in Industrial Healthcare Systems," IEEE Trans. Ind. Informatics, vol. 18, no. 11, pp. 8065–8073, 2022, doi: 10.1109/TII.2022.3161631.

[22] S. T. Muntaha, P. I. Lazaridis, M. Hafeez, Q. Z. Ahmed, F. A. Khan and Z. D. Zaharis, "Blockchain for Dynamic Spectrum Access and Network Slicing: A Review," in IEEE Access, vol. 11, pp. 17922–17944, 2023, doi: 10.1109/ACCESS.2023.3243985.

[23] Wongsamerchue, T., Leelasantitham, A. An Electronic Double Auction of Prepaid Electricity Trading Using Blockchain Technology. JMM 2022, 18, 1829–1850.

[24] Alamsyah, Andry, Naufal Hakim, and Ratih Hendayani. 2022. "Blockchain-Based Traceability System to Support the Indonesian Halal Supply Chain Ecosystem", *Economies* 10, no. 6: 134. https://doi.org/10.3390/economies10060134.

[25] Naman Kabra, Pronaya Bhattacharya, Sudeep Tanwar, Sudhanshu Tyagi, MudraChain: Blockchain-based framework for automated cheque clearance in financial institutions, Future Generation Computer Systems, Volume 102, 2020, Pages 574–587, ISSN 0167-739X, https://doi.org/10.1016/j.future.2019.08.035.

[26] "https://library.dedaub.com/." https://library.dedaub.com/.

[27] Mary Subaja Christo, V. Elizabeth Jesi, Uma Priyadarsini, V. Anbarasu, Hridya Venugopal, Marimuthu Karuppiah, "Ensuring Improved Security in Medical Data Using ECC and Blockchain Technology with Edge Devices", *Security and Communication Networks*, vol. 2021, Article ID 6966206, 13 pages, 2021. https://doi.org/10.1155/2021/6966206.

Biographies

S. Arunprasasth, B.Tech., M.E is currently a full-time research scholar at the SRM Institute of Science and Technology in Kattankullathur. He completed

his Master of Engineering in computer science and engineering in 2011, and he has 10 years of teaching experience. His research interests are blockchain technology and artificial intelligence.

A. Suresh, B.E., M.Tech., Ph.D works as the Associate Professor, Department of the Networking and Communications, School of Computing in SRM Institute of Science & Technology, Kattankulathur, Chengalpattu Dist., Tamil Nadu, India. He has been nearly two decades of experience in teaching and his areas of specializations are IoT, Data Mining, Artificial Intelligence, Image Processing, Multimedia and System Software. He has published eight patents and 150+ papers in International journals. He has book Authored "Bioinformatics and Medical Applications: Big Data using Deep Learning Algorithms" in Scrivener-Wiley publisher, "Practical Python Programming for Data Scientists" in Arcler Press, Canada, "Industrial IoT Application Architectures and use cases" published in CRC press and edited book entitled "Deep learning and Edge Computing solutions for High Performance Computing" published in EAI/Springer Innovations in Communications and Computing, "Sensor Data Management and Analysis: The Role of Deep Learning" in Scrivener-Wiley & IEEE Press publisher. "Deep Neural Networks for Multimodal Imaging and Biomedical Application" published in IGI Global. He has currently editing a book entitled "Resource Management in Advanced Wireless Mobile Networks: Emerging Challenges and Prospects" in Scrivener-Wiley publisher. He has published 20+ chapters in the various publisher like EAI/Springer Innovations in Communication and Computing, IGI Global Publisher CRC press etc. He has published more than 50 papers in National and International Conferences. He has served as editor/reviewer for Springer, Elsevier, Wiley, IGI Global, IoS Press, Inderscience journals etc. He is a member of IEEE (Senior Member), ISTE, MCSI, IACSIT, IAENG, MCSTA and Global Member of Internet Society (ISOC). He has organized

several National Workshop, Conferences and Technical Events. He is regularly invited to deliver lectures in various programmes for imparting skills in research methodology to students and research scholars. He has published four books in Indian publishers, in the name of Hospital Management, Data Structures & Algorithms, Computer Programming, Problem Solving and Python Programming and Programming in "C". He has hosted two special sessions for IEEE sponsored conference in Osaka, Japan and Thailand.