Peertechz

ENGINEERING GROUP

TRENDS IN Computer Science and Information Technology
OPEN ACCESS JOURNAL

ISSN: 2641-3086 DOI: https://dx.doi.org/10.17352/tcsit

Research Article

GRIFFIN: Enhancing the security of smart contracts

Franciscu SY*, Ruggahakotuwa RK, Samarawickrama SWYS and Lahiru JAD

Department of Computer, Systems Engineering, Sri Lanka Institute of Information Technology, Colombo, Sri Lanka

Received: 28 October, 2023
Accepted: 16 November, 2023
Published: 17 November, 2023

***Corresponding author**: Franciscu SY, Department of Computer, Systems Engineering, Sri Lanka Institute of Information Technology, Colombo, Sri Lanka, Tel: +94 76 3093803; E-mail: it20017910@my.sliit.lk ; zyberzone.info@gmail.com

ORCiD: https://orcid.org/0009-0008-8700-6785

https://www.peertechzpublications.org

Check for updates

Abstract

In the rapidly evolving landscape of decentralized systems, ensuring the integrity and trustworthiness of smart contracts is paramount for developers. This paper presents a comprehensive strategy for enhancing smart contract security by focusing on specific high-risk areas, including Integer Overflow, Dangerous Delegate Calls, Timestamp Dependency, Reentrancy Vulnerabilities, Race Conditions, and Sybil attacks. Despite the growing significance of smart contracts in blockchain ecosystems, a notable research gap exists in the development of specialized tools capable of providing real-time vulnerability detection and mitigation guidance. To bridge this gap, our research introduces the 'GRIFFIN' - Smart Contracts.

Vulnerability Detector is a powerful tool that has been rigorously tested and validated. Our study has yielded significant results, demonstrating the efficacy of the GRIFFIN in proactively identifying and mitigating critical vulnerabilities within a diverse dataset of 12,000 real-world solidity smart contracts. The tool leverages state-of-the-art static analysis techniques and machine learning algorithms, achieving superior accuracy rates when compared to existing solutions. This heightened accuracy not only empowers developers but also boosts the overall robustness and dependability of smart contract ecosystems. The cornerstone of our research is the development and validation of a practical, user-centric solution. By providing actionable insights, code snippets, and real-time feedback to developers, GRIFFIN equips them with the knowledge and tools needed to address vulnerabilities swiftly and effectively. This innovative approach is not merely an academic endeavor but a significant stride towards cultivating resilient and dependable smart contract environments. It instills a culture of security-conscious development practices, ensuring that the smart contracts crucial to decentralized systems can operate with the highest level of trust and reliability.

Index Terms— Smart Contracts; Integer overflow; Dangerous

Delegate call; Timestamp Dependence; Reentrancy Attack; Race

Condition; Sybil Attack; Static Analysis; Detection

Introduction

Smart contracts, which are autonomous agreements recorded on a blockchain [1], have emerged as a transformative innovation with diverse applications in finance, supply chain management, and decentralized applications [2]. As the popularity of smart contracts continues to grow, ensuring their security becomes paramount to prevent financial losses, legal disputes, and disruptions within decentralized ecosystems. This research study addresses the pressing need for robust smart contract security by presenting a comprehensive strategy for detecting vulnerabilities. We focus on detecting critical vulnerabilities such as Integer Overflow, Dangerous Delegate Call and Timestamp Dependency, Reentrancy Vulnerability, and Race Conditions. These vulnerabilities pose significant threats to the reliability, availability, and trustworthiness of smart contracts, necessitating their identification and prevention.

The increasing complexity and programmability of smart contracts render them susceptible to a range of security flaws. For instance, Integer Overflow occurs when

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

mathematical operations on unsigned numbers exceed their maximum value, resulting in unexpected and potentially hazardous behavior. Dangerous Timestamp and Delegate Call Dependency vulnerabilities expose contracts to tampering and unauthorized access, jeopardizing their intended functionality. Reentrancy Vulnerability enables attackers to repeatedly enter a contract before previous actions conclude, potentially leading to unauthorized fund transfers or data alterations. Race Conditions arise when the execution sequence of contract operations depends on unpredictable external factors, creating opportunities for malicious actors to exploit timing discrepancies.

We introduce 'GRIFFIN' - The Smart Contracts Vulnerability Detector, an advanced tool that integrates static analysis methodologies to address these challenges. Our tool scrutinizes smart contract codes to identify potential risks associated with these vulnerabilities. It provides developers with actionable insights and solutions to mitigate these vulnerabilities by analyzing control flow, data dependencies, and external interactions. Our approach aims to assist developers in crafting more secure and reliable smart contracts by leveraging a knowledge base of known vulnerability patterns and cutting-edge analytical techniques.

The remainder of this research study is organized as follows: Section 2 provides a summary of pertinent literature on smart contract security. Section 3 delves into the specific vulnerabilities we target and elucidates their impact on smart contract ecosystems. Section 4 details the design and implementation of our Smart Contracts Vulnerability Detector, including an explanation of the analytical methodologies and algorithms employed. Finally, Section 5 outlines our contributions and explores potential future avenues for enhancing smart contract security.

Related work

In recent years, much effort has been devoted to the creation of tools and frameworks for discovering vulnerabilities in smart contracts. These initiatives are part of a coordinated effort to improve the security of blockchain-based systems by proactively recognizing possible risks and vulnerabilities. Several prominent tools and frameworks have arisen, each applying its own methodology to assess and identify vulnerabilities in smart contracts. This collaborative endeavor underscores the industry's dedication to bolstering blockchain technology's integrity and resilience, eventually promoting a more secure and dependable environment for decentralized apps and transactions.

Among these trailblazing technologies, Mythril [3] stands out as an open-source symbolic execution tool that has been methodically engineered to expose vulnerabilities like reentrancy attacks, integer overflow, and unhandled exceptions. Because of its user-friendly design and rich analytical capabilities, Mythril has garnered traction in the blockchain world. It provides a deeper knowledge of the complexities of smart contract vulnerabilities by giving extensive data and actionable insights. This allows developers to make educated judgments and make required changes to ensure the robustness and security of their apps. Furthermore, Mythril's capacity to carry out symbolic execution successfully assists in the discovery of complicated problems, such as possible reentrancy assaults, which are notoriously difficult to identify using traditional techniques. Its ability to do extensive inspections of all alternative execution routes considerably contributes to the reduction of security risks and the prevention of unanticipated vulnerabilities, encouraging a more secure and dependable environment for Ethereum-based applications [4].

Oyente [5] is another famous tool in this space, renowned for its unique methodology utilizing symbolic execution and an intermediate representation to detect vulnerabilities. Oyente has made major contributions to smart contract analysis in the blockchain realm, owing to its novel usage of symbolic execution and an intermediate representation for vulnerability discovery. Its emphasis on transaction sequencing and call stack depth has been critical in identifying nuanced vulnerabilities that would otherwise go undetected in traditional testing. Nonetheless, its limitations in dealing with specific control flow categories paved the way for the development of more advanced tools such as Manticore. Manticore has emerged as a more complete solution by addressing these constraints, providing increased capabilities for deconstructing complicated smart contract vulnerabilities and improving the overall security infrastructure of blockchain applications.

Manticore [6] positioned as a high-performance symbolic execution tool, rectifies several limitations prevalent in its predecessors. Manticore hailed as a high-performance symbolic execution tool, has effectively overcome several problems that plagued previous tools. This progress is largely due to the addition of support for complicated instructions and a detailed examination of Ethereum's Yellow Paper specs. Its adaptability enables a thorough investigation of Ethereum smart contracts, allowing for the detection of vulnerabilities such as those caused by timestamp dependencies and complex gas-related behaviors. However, the ongoing argument about the balance of accuracy and scalability in Manticore execution remains an important academic study and conversation issue. This reflects the tool's importance in the field and the ongoing effort to improve its capabilities to reach a more ideal mix of precision and scalability for thorough smart contract analysis [7].

The surge in interest in machine learning has influenced the development of vulnerability analysis tools, and Security has successfully capitalized on this trend [8]. Security examines smart contracts to discover possible vulnerabilities using a rule-based architecture reinforced by machine learning. Securify significantly increases its precision and flexibility by using information obtained from a dataset including identified vulnerabilities, resulting in more effective and precise vulnerability identification. This technique not only improves the tool's capacity to detect existing vulnerabilities but also allows it to adapt to developing threat environments, making it a strong and proactive solution for enhancing smart contract security [9].

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

Furthermore, the emergence of tools like Slither [10] underscores the amalgamation of diverse analytical techniques. Slither integrates both static and dynamic analyses, culminating in a comprehensive identification of vulnerabilities encompassing uninitialized storage pointers and incorrect function access controls. This fusion of methodologies effectively harnesses the intrinsic strengths of both analysis types, thereby enabling a more comprehensive identification of vulnerabilities.

The importance of smart contract vulnerability research tools cannot be emphasized, especially with the growing integration of blockchain technology across several businesses. Mythril, Oyente, Manticore, Securify, and Slither are just a few of the tools available, each concentrating on a different area of smart contract security. Their different techniques, which range from symbolic execution to machine learning integration, represent continuous attempts to improve the accuracy, scalability, and adaptability of vulnerability analysis tools. As the blockchain ecosystem evolves, these tools will be critical in ensuring the integrity and security of smart contracts, permitting the widespread acceptance and deployment of blockchain solutions across multiple industries.

Methodology

The technique described here seeks to give a complete approach to the study and identification of vulnerabilities in smart contracts by using a strong mix of data preprocessing, machine learning model training, and sophisticated security mechanisms. The importance of guaranteeing the integrity and security of smart contracts cannot be emphasized in the ever-expanding realm of blockchain technology. The potential dangers associated with vulnerabilities like reentrancy attacks, token misuse, and Sybil attacks [11] have been focal points for developers and stakeholders as blockchain is increasingly being integrated across multiple businesses. This technique addresses these issues by executing a systematic procedure that begins with the collection and analysis of a large dataset and ends with the building of a sophisticated web application interface capable of delivering thorough vulnerability reports (Figure 1).

The first phase is acquiring a comprehensive dataset comprised of a varied variety of smart contracts and accompanying transaction data, allowing for a more nuanced understanding of the prevalent vulnerabilities within the blockchain ecosystem. Following that, a thorough data preprocessing step is carried out, which includes feature extraction techniques, data balancing tactics, and multiple classification approaches. This makes it easier to create a clean and structured dataset, which is required for the successful training of different machine learning models such as Deep Neural Networks [12], SVM classifiers [13], K-NN classifiers [14], and ensemble approaches such as Adaboost [15], Random Forest [16], and Extreme Boost classifiers [17].

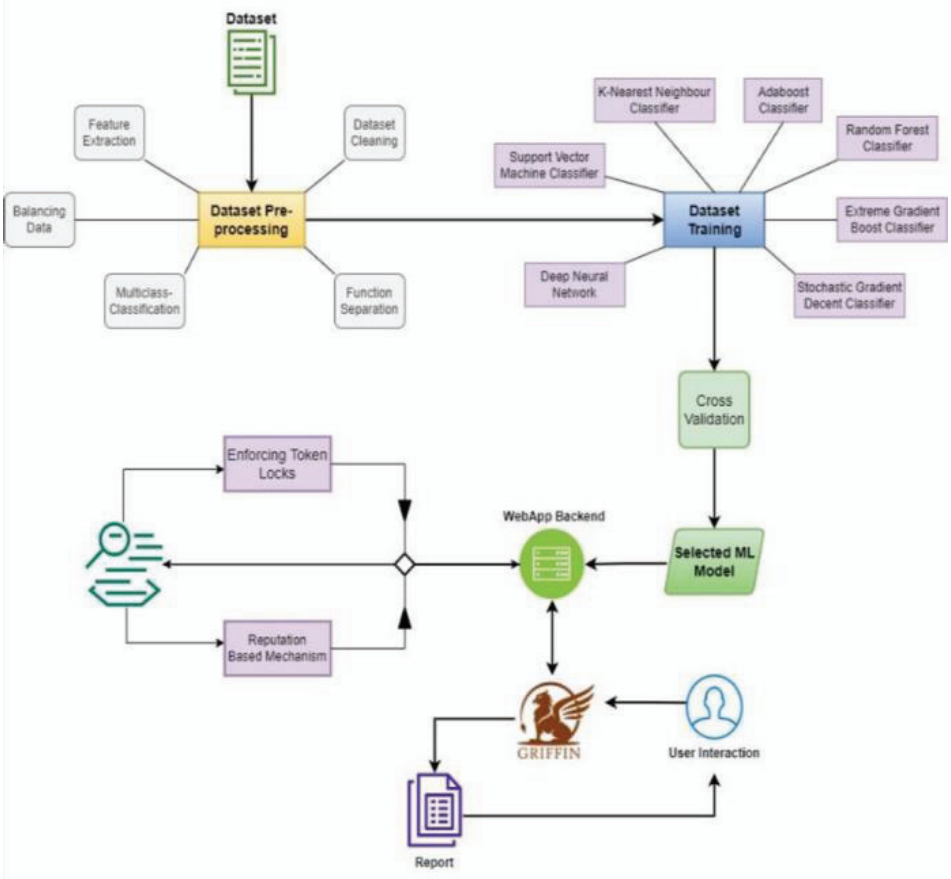

Figure 1: Technical Overview Diagram.

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

The trained models are then subjected to cross-validation techniques to assess their resilience and dependability, allowing the discovery and selection of the best machine-learning algorithm for smart contract vulnerability detection. Furthermore, sophisticated security features, like a token locking mechanism and a reputation-based system to resist Sybil attacks, are included in the technique to strengthen the smart contract system's security infrastructure. This strategy culminates in a user-friendly web application interface with a robust reporting system, allowing users to proactively reduce vulnerabilities and assure the resilience and trustworthiness of their blockchain-based apps.

Dataset preparation

To effectively harness machine learning models for the purpose of learning from sequential data, it is imperative to meticulously curate a well-structured dataset. This section elucidates the multifaceted steps involved in the meticulous preparation of datasets pertaining to both vulnerable and nonvulnerable smart contracts.

In the context of this research paper, we employed the Smart Contract Dataset sourced from the Messi-Q GitHub repository [18]. This dataset has been generously shared on GitHub by Peng Qian, a PhD candidate specializing in Computer Science at Zhejiang University [19].

The dataset acquisition process involved the retrieval of data from Etherscan's verified contracts, which represent genuine smart contracts deployed on the Ethereum mainnet [20]. The resultant dataset comprised a total of 12,515 smart contracts, all of which possessed corresponding source code. In accordance with Table I, focus was placed on eight distinct vulnerability types within this dataset, specifically Timestamp Dependency (TP), Block Number Dependency (BN), Dangerous Delegate Call (DG), Ether Frozen (EF), Unchecked External Call (UC), Reentrancy (RE), Integer Overflow (OF), and dangerous Ether Strict Equality (SE).

For the definitive labeling of the smart contracts in question, a comprehensive two-step methodology was adopted. Initially, vulnerability-specific patterns, including the utilization of keyword matching, were employed to provisionally assign labels to these contracts. Subsequently, a thorough manual assessment was conducted to validate the presence of specific vulnerabilities within each smart contract. This approach was instrumental in optimizing the labeling process by prioritizing the identification of potentially vulnerable contracts, while simultaneously excluding those determined to be secure. Notably, all files were systematically organized based on their respective vulnerabilities, and each dataset was accompanied by a function-wise comprehensive ground truth table with binary classification.

Table 1: Result of ML algorithms.

Model name	Accuracy	F1 Score	Precision	Recall	Cross-validation
KNN	0.88	0.99	0.98	0.99	0.88
Random Forrest	0.90	0.99	0.99	0.99	0.90
SVM	0.90	0.99	0.99	0.99	0.90
SGDC	0.89	0.99	0.99	0.99	0.89
ADB	0.39	0.97	0.98	0.94	0.39
XGB	0.88	0.99	0.99	0.98	0.88

Data pre-processing

The inherent imbalance within the dataset was an important concern that necessitated addressing throughout the initial stages of our data preparation procedure. It became clear that the distribution of files across the various vulnerability categories was noticeably unequal. To correct this imbalance, we used the 'learn.over_sampling' methodology [21], a robust method that produced random samples for each vulnerability class methodically. This methodical approach guaranteed that every vulnerability category in our dataset had a consistent representation and that all categories reached the stated maximum dataset sample size. As a result, we were able to standardize our dataset to include a total of 100K data points, successfully minimizing any inherent discrepancies and promoting equitable representation.

```
ros = RandomOverSampler(sampling_strategy='auto', random_state=42)

X_resampled, y_resampled = ros.fit_resample(X, y)

X_resampled = X_resampled['Functions']

processed_data = pd.DataFrame({

Function': X_resampled,

'Label': y_resampled

})
```

Random over sampling

Following the harmonization of our dataset, a critical step was performed to combine the many data sources into a single, cohesive entity. This combination of data sources permitted the production of a complete ground truth table that included all various vulnerability kinds that had been discovered. This table served as a cornerstone for our subsequent analyses, enabling a comprehensive and systematic examination of vulnerabilities across the entire dataset. With these preparatory steps completed, we now possess a meticulously curated dataset that is distinguished by its function-wise labeling and multi-class classification scheme, setting the stage for the advanced stages of our research endeavor.

```
vulnerability_mapping = {

1: "Block Number Dependency",

2: "Reentrancy",

3: "Timestamp Dependency Vulnerability",

4: "Dangerous Delegatecall Vulnerability",
```


Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

5: "Integer Overflow Vulnerability",

}

Multi-class classification

In our data preparation, we partitioned our balanced 100,000data-point dataset into training (80%) and testing (20%) subsets, adhering to standard practice. This segmentation enables robust model training as well as independent model assessment.

X_train, X_test, Y_train, Y_test = train_test_split(features, labels, test_size=0.2)

feature_extraction = TfidfVectorizer(min_df = 1, lowercase=True)

TFIDF Vectorizer [22] was used for feature extraction to aid machine learning. This method translates textual smart contract source code to numerical representation by assigning values to terms based on their relevance in documents and throughout the dataset. This step was critical in preparing our smart contract data for machine learning analysis later.

Implementation

In our research, we have harnessed a suite of seven distinct machine learning models, each thoughtfully selected for its suitability in addressing the complexities of multi-class classification to train the above data set. We have analyzed 7 major models such as K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest Classifier, Extreme Gradient Boost (XGBoost) Classifier, Stochastic Gradient Descent (SGD) classifier, AdaBoost Classifier and Deep Neural Network (DNN) which give better accuracy results.

In our analysis, K-Nearest Neighbors (KNN) excels for its simplicity and effectiveness in multi-class classification. Random Forest, an ensemble method, is notable for its capacity to combine decision trees for accuracy. Support Vector Machine (SVM) handles multi-class tasks efficiently, while the Stochastic Gradient Descent Classifier (SGD) [23] is pragmatic when combined with a one-vs-all (OvA) strategy. XGBoost is known for its speed and multi-class support. Deep Neural Networks (DNNs) are adept at capturing complex patterns. However, AdaBoost (ADB), effective in binary classification, may not be ideal for multi-class scenarios. In such cases, alternative models may offer more straightforward solutions.

To determine the optimal model among the candidates, we employed a comprehensive assessment framework encompassing four fundamental evaluation metrics: Accuracy, F1 Score, Precision, and Recall. These metrics, when used in conjunction with cross-validation, serve as indispensable tools for a thorough evaluation of classification models. Accuracy offers a holistic gauge of predictive correctness, demonstrating its utility in scenarios characterized by balanced datasets. The F1 Score strikes a crucial balance between precision and recall, making it particularly adept at handling imbalanced class distributions. Precision meticulously assesses the accuracy of positive predictions, proving invaluable in situations where false positives carry substantial costs. Meanwhile, Recall quantifies a model's prowess in capturing relevant instances, a critical measure in contexts where missing positive instances entails significant consequences. Complementing these metrics, cross-validation bolsters the robustness of model assessment by systematically partitioning data, mitigating the risks of overfitting, and ensuring reliable generalization.

The selection of an appropriate metric and cross-validation strategy is contingent upon the specific machine-learning task at hand (Table 1).

Epochs are a crucial aspect of training Deep Neural Networks (DNNs) [24]. They denote the number of times the entire training dataset is processed forward and backward through the network. In our analysis, we employed 5 epochs. This iterative learning approach serves several essential purposes. It enables the gradual refinement of the network's internal parameters, ensuring convergence toward an optimal configuration for the given task. Multiple epochs are necessary to avoid overfitting, promote generalization, and facilitate learning rate adjustments (Figure 2).

After all parts of the training phases and finetuning, we got a clear and reliable accuracy rate of 0.9002 accuracy value.

The introduction of a strong token-locking mechanism and a reputation-based system inside the smart contract ecosystem acts as a vital line of protection against possible weaknesses, notably Sybil attacks. As a preventative precaution, token locking entails the implementation of a secure protocol that restricts the transfer or use of tokens for a set period, therefore limiting the risks associated with illegal token transfers and potential manipulation.

This protocol is built within the smart contract architecture, and it makes use of cryptographic methods and consensus procedures to assure the immutability and transparency of token transactions. The token locking mechanism functions as a precaution against fraudulent operations by imposing time-based limits and preset access rights, boosting the overall security and trustworthiness of the blockchain-based system.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Interface for the ERC-20 token contract interface IToken {

function transfer(address recipient, uint256 amount) external returns (bool);

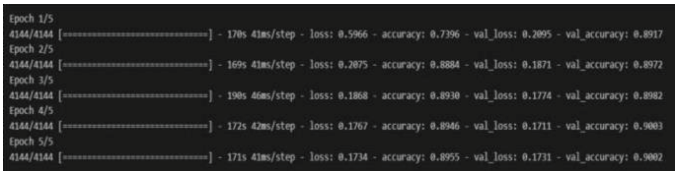

Figure 2: Avoid overfitting using Epochs.

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

```
function balanceOf(address account) external view returns
(uint256);
function allowance(address owner, address spender) external
view returns (uint256);
function transferFrom(address sender, address recipient,
uint256 amount) external returns (bool);
}
// TokenLockContract is a contract that enforces a token lock
requirement
contract TokenLockContract {
// Contract owner
address public owner;
// ERC-20 token interface
IToken public token;
// Amount of tokens required to be locked
uint256 public requiredTokens;
// Mapping of user addresses to their locked token amounts
mapping(address => uint256) public lockedTokens;
// Constructor initializes the contract
constructor(address _tokenAddress, uint256 _requiredTokens) {
owner = msg.sender;
token = IToken(_tokenAddress);
requiredTokens = _requiredTokens;
}
// Modifier that allows only the contract owner to call a function
modifier onlyOwner() {
require(msg.sender == owner, "Only the owner can call this
function");
_;
}
// Modifier that checks if the user has the required locked tokens
modifier hasRequiredTokens() {
require(lockedTokens[msg.sender] >= requiredTokens,
"Insufficient locked tokens");
_;
}
// Lock tokens by transferring them from the user to this contract
function lockTokens(uint256 amount) external {
require(amount > 0, "Amount must be greater than 0");
// Transfer tokens from user to contract
require(token.transferFrom(msg.sender, address(this),
amount), "Token transfer failed");
// Increase the user's locked token balance lockedTokens[msg.
sender] += amount;
}
// Unlock tokens and transfer them back to the user
function unlockTokens(uint256 amount) external
hasRequiredTokens {
require(amount <= lockedTokens[msg.sender], "Insufficient
locked tokens");
// Decrease the user's locked token balance
lockedTokens[msg.sender] -= amount;
// Transfer tokens back to user
require(token.transfer(msg.sender, amount), "Token transfer
failed");
}
// Update the required locked tokens by the contract owner
function updateRequiredTokens(uint256 newRequiredTokens)
external onlyOwner {
requiredTokens = newRequiredTokens;
}
// Get the amount of locked tokens for a specific user
function getLockedTokens(address account) external view
returns (uint256) {
return lockedTokens[account];
}
// Example function that requires the user to have the required
locked tokens
function interactWithLockedTokensFunction() external
hasRequiredTokens {
// Your function logic here
}
}
```


Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

Token locking mechanism

Simultaneously, the adoption of a reputation-based mechanism necessitates the construction of a complete system that assesses and monitors network participants' behavioral patterns and transaction histories. This approach distributes reputation ratings to individual users based on their previous interactions and adherence to set protocol requirements using advanced algorithms and data analytics techniques.

By incorporating this reputation score system into the smart contract ecosystem, it is feasible to differentiate between legitimate users and potential malevolent actors conducting Sybil assaults. Users with high reputation scores have expanded privileges and access to capabilities, whilst those with poor reputation scores face onerous verification processes and limited access rights. This strategy provides a trustworthy and safe environment, establishing a feeling of accountability and integrity among network users and discouraging malevolent acts that might jeopardize the blockchain system's overall integrity and reliability.

```
// SPDX-License-Identifier: MIT

pragma solidity ^o.8.o;

contract WhitelistedFunction {

address public owner;

mapping(address => bool) public whitelist;

constructor() {

owner = msg.sender;

}

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can call this

function");

_;

}

modifier onlyWhitelisted() {

require(whitelist[msg.sender], "You are not whitelisted");

_;

}

function addToWhitelist(address account) external onlyOwner {

whitelist[account] = true;

}

function removeFromWhitelist(address account)
external

onlyOwner {

whitelist[account] = false;

}

function whitelistedFunction() external onlyWhitelisted {

// Your function logic here

}

}
```

Reputation-based mechanism

Results & discussion

Research findings

The research identified a range of common vulnerabilities in smart contracts, including reentrancy attacks, race conditions, integer overflows, and logical errors. These vulnerabilities pose significant risks to the security and functionality of smart contracts. Each vulnerability was assessed for its potential impact. Findings showed that certain vulnerabilities, such as reentrancy attacks, can result in substantial financial losses, while others, like logical errors, may lead to unintended contract behaviors.

A comprehensive report was created to educate users about smart contract vulnerabilities. The report explained each vulnerability in user-friendly language, offering clear and actionable recommendations for evaluating and engaging with smart contracts securely.

Future research may explore the integration of machine learning models to enhance vulnerability detection in smart contracts. This could automate the identification of new vulnerabilities and improve the overall security landscape.

There is a need for user-friendly tools that guide smart contract developers in writing secure code. Such tools can help prevent vulnerabilities during the development phase and empower developers to write secure contracts.

The research recognized the growing importance of ethical considerations, including responsible disclosure of vulnerabilities and ethical user engagement. It emphasized the need for an ethical approach within the blockchain community.

Limitations and future work

Several difficulties and possibilities emerge in the field of smart contract security. Because of the computational intensity needed, the scalability of contract analysis is a significant challenge, particularly in large blockchain networks. The current tools and infrastructure may struggle to keep up with the examination of many contracts in an acceptable amount of time. Furthermore, the complexity of vulnerabilities is a continuing problem, since developing sophisticated attacks may outstrip existing research on common flaws. Encouraging user adoption of security practices is a continuous problem since behavioral change is slow and resistant to rapid transformation. The study emphasizes the relevance of legal

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

concerns without giving navigational assistance within the shifting legal environment, which adds another degree of complexity.

Furthermore, the study briefly mentions privacy risks related to smart contracts, implying that future research should dive deeper into their influence on privacy, especially when sensitive data is involved. In the future, researchers may investigate the use of formal verification techniques for smart contracts, providing mathematical proof of correctness to prevent vulnerabilities. The creation of user-friendly tools to assist programmers in designing safe smart contracts is an area ripe for innovation, aligning with a proactive approach to vulnerability avoidance throughout the development stage.

Conclusion

In conclusion, this work has proposed a thorough method for strengthening smart contract security in the context of dynamic decentralized systems. Our study has highlighted the need to emphasize integrity and trustworthiness in smart contract creation by tackling crucial high-risk areas such as Integer Overflow, Dangerous Delegate Calls, Timestamp Dependency, Reentrancy Vulnerabilities, Race Conditions, and Sybil attacks.

According to this comprehensive study, Machine Learning is the most suitable method to analyze smart contracts. To counter those vulnerabilities, we have built a web application using Python Flask that has smart contracts analyzing capability. The developer may upload a solidity file or copy the solidity code to our portal. Then the web application will give detailed results on whether that contract is vulnerable or not and the user has the capability to get those results in pdf format, if it is vulnerable; what the vulnerability is, which code section it placed and it recommends online sources to resolve the weakness of the contract.

Recognizing a gap in the market for specialized tools for real-time vulnerability identification and mitigation advice, our team developed the 'GRIFFIN' - Smart Contracts Vulnerability Detector. We have proved the efficiency of GRIFFIN in proactively finding and mitigating vulnerabilities across a varied dataset of 12,000 real-world solidity smart contracts through rigorous testing and validation. GRIFFIN has demonstrated improved accuracy rates by leveraging cutting-edge static analysis techniques and machine learning algorithms, enabling developers, and increasing the overall robustness of smart contract ecosystems.

The emphasis on a practical, user-centric solution is central to our study. GRIFFIN facilitates rapid and effective vulnerability remediation by providing developers with actionable insights, code snippets, and real-time feedback. This method not only makes a substantial contribution to the creation of strong smart contract ecosystems, but it also develops a culture of security-conscious development practices. We are certain that by using these techniques, we will be able to assure the smooth and dependable operation of smart contracts inside decentralized systems, hence increasing overall trust and reliability.

By adopting these mitigation strategies, developers can significantly improve the security, integrity, and resilience of smart contract systems. These measures proactively address risks, instilling confidence in decentralized applications and blockchain ecosystems. Our study contributes to fostering safer and more reliable blockchain networks and decentralized applications.

Acknowledgment

We extend our heartfelt appreciation to all those who contributed to the completion of this research study. Our gratitude goes to the individuals and organizations that provided valuable insights, guidance, and resources throughout the research process. We are thankful for the support from our mentors and advisors, whose expertise and feedback were instrumental in shaping this study.

We would also like to acknowledge the participants who willingly shared their perspectives and experiences, contributing to a deeper understanding of the subject matter. Additionally, we are grateful for the research facilities and technological resources that enabled the successful execution of our experiments and analyses.

Finally, we express our gratitude to the academic community for fostering an environment of learning and discovery. Through this research, we hope to make a meaningful contribution to the field and inspire further advancements in the realm of smart contract security.

References

1. Wood G. Google Scholar. 2014. https://cryptodeep.ru/doc/paper.pdf.
2. IBM. IBM. https://www.ibm.com/topics/smart-contracts.2023.
3. ConsenSys. GitHub. https://github.com/ConsenSys/mythril.
4. Cai W, Wang Z, Ernst JB, Hong Z, Feng C, Leung VCM. Decentralized Applications: The Blockchain-Empowered Software System. in IEEE. 2018.
5. Olickel H. Oyente: Making Smart Contracts Smarter. Academia. https://www.academia.edu/28735174/Oyente_Making _Smart_Contracts_Smarter.
6. Mossberg M, Manzano F, Hennenfent E, Groce A, Grieco G, Feist J. Manticore: A User-Friendly Symbolic ExecutionFramework for Binaries and Smart Contracts. in IEEE/ACM. 2019.
7. Trailofbits. trailofbits/manticore. https://github.com/trailofbits/manticore/.
8. Tsankov P, Dan A, Drachsler-Cohen D, Gervais A, Bünzli F, Vechev M. Securify: Practical Security Analysis of Smart Contracts. in ACM. 2018.
9. eth-sri/securify2. https://github.com/eth-sri/securify2.
10. Feist J, Grieco G, Groce A. Slither: A Static Analysis Framework for Smart Contracts. arXiv. 2019.
11. Shubhani Aggarwal NK. Chapter Twenty - Attacks on blockchain. Advances in Computers. 2021; 121: 399-410.
12. Alzubaidi L, Zhang J, Humaidi AJ, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data. 2021; 8: 53.
13. Zhang Y. Support Vector Machine Classification Algorithm and Its Application. in ICICA 2012. 2012.

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071

14. Guo G. KNN Model- Based Approach in Classification. in OTM 2003. 2003.

15. Chengsheng T, Huacheng L, Bing Xu. AdaBoost typical Algorithm and its application research. in MATEC Web of Conferences. 2017.

16. Breiman L. Random Forests. Machine Learning. 2001; 45: 5-32.

17. Dhieb N, Ghazzai H, Besbes H, Massoud Y. Extreme Gradient Boosting Machine Learning Algorithm For Safe Auto Insurance Operations, in 2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES). 2019.

18. Qian P. Messi-Q. https://github.com/Messi-Q/Smart-Contract- Dataset/blob/master/README.md#smart-contractdataset.

19. Qian P. https://scholar.google.com/citations?user=ic5pZxEAA AAJ&hl=en.

20. Ethereum Whitepaper. https://ethereum.org/en/whitepaper/.

21. imbalanced-learn. https://imbalanced-learn.org/stable/.

22. scikit-learn. https://scikitlearn.org/stable/modules/generated/sklearn.feature_ext raction.text.TfidfVectorizer.html.

23. Amari Si. Backpropagation and stochastic gradient descent method. Neurocomputing. 1993; 5:185-196.

24. Yi H. Shiyu S, Xiusheng D, Zhigang C. A study on Deep Neural Networks framework, in 2016 IEEE Advanced Information Management. Communicates, Electronic and Automation Control Conference (IMCEC). 2016.

Citation: Franciscu SY, Ruggahakotuwa RK, Samarawickrama SWYS, Lahiru JAD (2023) GRIFFIN: Enhancing the security of smart contracts. Trends Comput Sci Inf Technol 8(3): 073-081. DOI: https://dx.doi.org/10.17352/tcsit.000071