Vulnerability Analysis of Smart Contract on Blockchain Platforms

Karan Tejwani
Department of Computing Security
Rochester Institute of Technology
NY, USA
kt9576@rit.edu

Pranjali Thakur
Department of Computing Security
Rochester Institute of Technology
NY, USA
pt4202@rit.edu

***Abstract*—Traditional financial systems that are centralized need a trusted third party to manage and validate the transactions from one party to another. These transactions may have a processing time that may vary from hours to days. On the other hand, blockchain technology and its implementation of smart contracts, provide a decentralized and trustless alternative. As smart contracts gain traction in a variety of businesses, their security and durability have become significant considerations. However, the complexity of blockchain-based systems, as well as their inherent vulnerabilities, pose significant risks to their security and integrity. This survey study provides a complete examination of smart contract vulnerabilities and investigates the tools and approaches available for detecting them on blockchain systems.**

I. Introduction

Smart contracts have emerged as a groundbreaking technology in the realm of decentralized systems, revolutionizing the way agreements are executed and enforced. Smart contracts are self-executing digital contracts that run on blockchain networks. They are designed to automate and enforce the terms and conditions of agreements between parties without the need for intermediaries. They have predefined rules and conditions written in code. As these Smart contracts are deployed on a Blockchain platform they have properties such as Immutability, Transparency and are Distributed and Secure. However, it is important to note that while Smart contracts offer many advantages they are not immune to vulnerabilities or flaws in design, implementation, or the blockchain platform. The complex nature of Smart contracts has introduced many vulnerabilities and risks that can have severe consequences.

A. Importance of the Topic

As Smart contracts are being used in a wide range of applications, such as finance, supply chain, and governance, it becomes necessary to address the security and reliability of these contracts. Vulnerabilities in smart contracts can lead to a huge financial loss, unauthorized access, or manipulation of contract logic, undermining the trust and benefits offered by blockchain technology. Hence, vulnerability analysis and detection tools are essential for the identification and mitigation of potential risks.

B. Motivation

This paper provides a comprehensive overview of the current research on vulnerability analysis of smart contracts. We aim to understand the various vulnerabilities and risks associated with smart contracts and the tools available for detecting and mitigating these vulnerabilities.

C. Goal of the Survey

We aim to identify the different types of vulnerabilities that can occur in smart contracts and have caused severe attacks in the past. We have categorized these as design flaws, cryptographic weaknesses, time-related risks, external interactions, and integration risks, Arithmetic and Numeric issues, and Tooling and Platform risks. We also compare the tools and methods used to detect these vulnerabilities. This survey also gives examples of past attacks that occurred due to these vulnerabilities.

D. Our Solution

Along with surveying the existing literature about different vulnerabilities and detection tools, this paper also proposes to classify the vulnerabilities according to their Characteristics, unlike the early study of the classification of these vulnerabilities. We compare the vulnerability detection tools with parameters such as analysis types and methodologies used.

II. Background Knowledge

Blockchain is a digital ledger technology that is used to store and track data in a secure, decentralized, and tamper-proof manner. A blockchain is essentially a database that consists of a series of blocks, each containing a set of transactions. These blocks are linked together in a chronological sequence, forming a chain of blocks - hence the name ”blockchain”. These blocks are immutable as they contain the hashed value of the previous block. And if a malicious actor tries to change anything in the block the entire hash value changes drastically. The most commonly used cryptographic hash function is SHA-256. Hash algorithms provide a high level of security and efficiency, which is important for decentralized applications.

One of the most significant applications of blockchain technology is smart contracts, which automate the execution of contractual agreements on a blockchain network. Smart

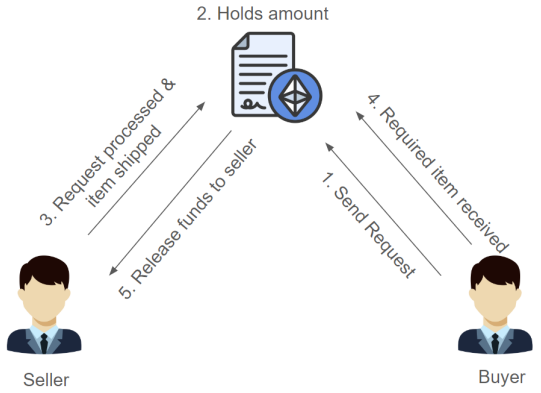

Fig. 1. Real-world example of Smart contract execution

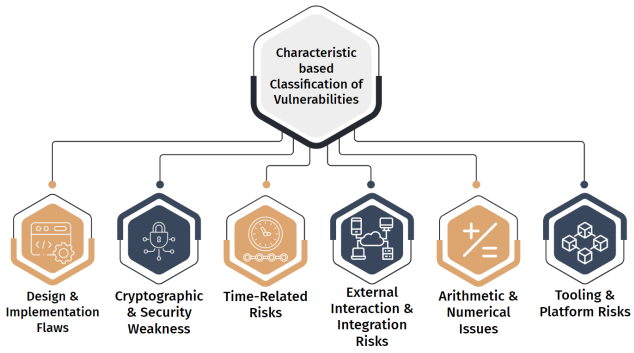

Fig. 2. Characteristic-based Classification of Vulnerabilities

contracts are self-executing programs that are stored on a blockchain and are designed to execute automatically when certain conditions are met. Any rules and functionalities can be written in a compatible programming language and encoded as a smart contract to be invoked whenever an action is required by a user or other smart contracts. They are designed to automate and enforce the terms and conditions of agreements between parties without the need for a middleman, like banks or escrow agents.

Figure 1 [6] shows a real-world example of smart contract execution, where there are two users, a seller and a buyer, who are engaging in business with the help of a smart contract. Step 1 is where the buyer sends the required amount of funds to the smart contract's address, where the contract acts as an escrow account. In Step 2, the smart contract notifies the seller of the buyer's request. In Step 3, the seller verifies the request, if there is the correct amount of funds, the seller ships the item and informs the smart contract. After the buyer receives the item, the smart contract updates the delivery status, and in the final step, the contract transfers the funds from the address of the Smart contract to the seller's address.

Smart contracts can hold and manage a large amount of virtual currencies which could be worth thousands of dollars. Therefore, the adversaries keep attempting to manipulate the execution of smart contracts in favor of their activities. Vulnerabilities pose significant risks to the integrity, security, and functionality of smart contracts, making vulnerability analysis and detection crucial for ensuring their reliability and resilience.

[1] Classifies vulnerabilities into three groups according to where they were introduced, that is, in the Programming language Solidity, the Ethereum Virtual Machine (EVM) or the blockchain. [2] [3] Classify these vulnerabilities based on properties like trace, safety, liveness, Hyperproperties. [4] Here the Decentralized Application Security Project (DASP) Top 10, classifies ten groups of smart contract vulnerabilities but the ranking and selection methodology of this is not provided. Various research works and tools have been developed to address the challenges associated with vulnerability analysis in smart contracts. These works employ a range of techniques, including static analysis, dynamic analysis, symbolic execution, and formal verification, to identify vulnerabilities and enhance the security of smart contracts.

The importance of vulnerability analysis in smart contracts is underscored by the growing adoption of blockchain technology across industries. High-profile incidents, such as the DAO attack and the Parity wallet vulnerability, have highlighted the need for robust security measures in smart contract development and deployment.

III. Systematic Review

A. Categorization based on Vulnerability types

Categorizing vulnerabilities helps in understanding the nature of the security risks associated with smart contracts and guides developers in implementing appropriate security measures and best practices to mitigate these risks. This paper classifies Smart contract vulnerabilities into different categories based on common characteristics or underlying causes. Figure 2 shows the classification based on the Characteristics of the vulnerabilities. Here are groups for classifying smart contract vulnerabilities:

a) Design and Implementation Flaws: Design and Implementation Flaws consist of vulnerabilities that arise from errors or flaws in the design or implementation of the contract's logic. These include vulnerabilities such as:

- Logic Bugs: These occur when there are errors in the programming logic of the smart contract. Because logic bugs can be subtle and difficult to detect, thorough code review and testing are essential for detecting and addressing them.
- Input Validation Issues: These occur when the smart contract fails to validate user input in a proper manner. Proper input validation is required to verify that the contract works as intended and to avoid potential security issues.
- Access Control Vulnerabilities: These arise when the smart contract does not effectively enforce access restrictions or permissions. Proper access control measures should be put in place to ensure that only authorized

parties can perform particular actions and gain access to sensitive information.

b) Time-Related Risks: Time-Related Risks refer to vulnerabilities that are connected with the contract's manipulation or exploitation of time-related functions or dependencies. These include vulnerabilities such as:

- Timestamp Manipulation: These arise when smart contracts depend on time-sensitive functions that can be manipulated by attackers. Attacks can alter the order of transactions which can lead to unfair advantages and data inconsistencies.
- Front-Running Attacks: These occur when an attacker finds that a transaction is pending in the mempool of the network and then they immediately submit their own transaction to disrupt fairness and manipulate the outcome.

c) Cryptographic and Security Weaknesses: Cryptographic and Security Weaknesses are vulnerabilities that arise from weaknesses in cryptographic mechanisms used in the smart contract. These include vulnerabilities such as:

- Key Management Issues: These are caused due to inadequate techniques for generating, storing, and using cryptographic keys within the smart contract. This can jeopardize private keys, exposing the contract to unauthorized access or manipulation.
- Cryptographic Vulnerabilities: These arise from flaws in cryptographic algorithms or incorrect use of cryptographic mechanisms within the smart contract. These can include outdated or insecure algorithms, incorrect key generation or management, or flawed encryption or signature schemes.

d) Arithmetic and Numerical Issues: Arithmetic and Numerical Issues are related to vulnerabilities caused by inappropriate handling of arithmetic operations and numerical data within the contract. These include vulnerabilities such as:

- Integer Overflow/Underflow: When arithmetic operations within the smart contract surpass the boundaries of the data type used to record numerical values, integer overflow/underflow issues emerge. Overflow happens when the operation result exceeds the maximum value and underflow happens when the operation result falls below the minimum value. [6]

e) External Interaction and Integration Risks: External Interaction and Integration Risks relate to the vulnerabilities associated with interactions between the smart contract and other entities such as other contracts, systems, or data sources. [20] These include vulnerabilities such as:

- External Call Risks: When the smart contract interacts with other contracts or systems via external function calls, external call vulnerabilities occur. These include reentrancy attacks, where an external contract can maliciously call back into the current contract before the prior call completes, potentially leading to unexpected behaviors or unauthorized financial transfers.
- Dependency Risks: This vulnerability arises when there are flaws in the external libraries or other dependencies used by the smart contract.

f) Tooling and Platform Risks: Tooling and Platform Risks relate to vulnerabilities and risks caused by defects, flaws, or vulnerabilities in the tools, compilers, or underlying blockchain platforms used for smart contract development and implementation. These include vulnerabilities such as:

- Compiler Bugs: Compiler bugs are faults or vulnerabilities in the software compilers that are used to convert smart contract code into executable bytecode.
- Platform Vulnerabilities: Platform Vulnerabilities are flaws or vulnerabilities in the underlying blockchain platform or infrastructure on which smart contracts are deployed. They include consensus algorithm flaws, security weaknesses in the platform's architecture, and protocol vulnerabilities.

B. Smart Contract Security Analysis Tools

After identifying the various types of vulnerabilities present in smart contracts, the next step is to use security analysis tools to detect them. In this survey paper, we have reviewed several popular smart contract security analysis tools, which include Oyente [12], Zeus [19], Gasper [16], Vandal [14], EthIR [17], Securify [18], Maian [15], F* [13], FEther, Mythril, Slither, VeriSol, and Manticore. A basic introduction to these tools is provided in Table I.

The selection of these tools was based on their popularity, availability, and ability to detect a wide range of vulnerabilities in smart contracts. Some of these tools are more suitable for certain types of vulnerabilities than others, while some are capable of detecting a broad range of vulnerabilities. It is worth noting that some tools use multiple analysis techniques, such as combining static and dynamic analysis, to improve their accuracy in detecting vulnerabilities.

Overall, the selection of a tool depends on various factors such as the type of vulnerability to be detected, the complexity of the smart contract, and the level of accuracy required. It is also important to keep in mind that no tool can guarantee 100% security, and manual code review and testing should always be performed alongside automated analysis to ensure the highest level of security for smart contracts.

C. Security Analysis Methods of Smart Contracts

a) Static Analysis: Static Analysis is a method of analyzing software without executing it. This technique inspects source code or binary code to identify potential vulnerabilities, coding errors, or non-compliant coding practices. It can be performed manually or with the help of automated tools.

b) Dynamic Analysis: Dynamic Analysis involves analyzing a software program while it is being executed. This technique is used to detect potential runtime errors, memory leaks, and other issues that may not be visible through static analysis. Dynamic analysis can be performed using various techniques such as debugging, profiling, and monitoring.

TABLE I
SMART CONTRACT SECURITY ANALYSIS TOOLS

Tool	Description
Oyente	Symbolic execution tool that detects reentrancy, transaction ordering, and timestamp issues.
Zeus	Symbolic analysis and SMT solvers for security checks, including reentrancy and arithmetic.
Gasper	Analyzes gas consumption in Ethereum smart contracts to identify and mitigate gas issues.
Vandal	Constructs control flow graphs of Ethereum smart contracts to identify security issues.
EthIR	Generates an intermediate representation of Ethereum smart contracts for analysis.
Securify	Examines smart contracts for vulnerabilities by checking compliance with security patterns.
Maian	Symbolic execution and static analysis tool to detect trace vulnerabilities in smart contracts.
F*	A verification-oriented programming language used for formally verifying smart contracts.
FEther	Uses F* framework to create formal specifications for Ethereum smart contracts.
Mythril	Symbolic execution, taint analysis, and control flow analysis for detecting vulnerabilities.
Slither	Static analysis framework to detect vulnerabilities, code quality issues, and optimizations.
VeriSol	Formal verification tool for Solidity smart contracts by Microsoft Research.
Manticore	Symbolic execution tool for smart contracts and binaries, supporting Ethereum and EVM bytecode.

c) Formal Verification: Formal Verification is a technique that mathematically proves the correctness of a system with respect to a formal specification. It involves creating a mathematical model of the system and using logic-based techniques to prove that the system meets its requirements. Formal verification is particularly useful for verifying critical systems, where failure is not an option.

TABLE II
SECURITY ANALYSIS METHODS OF SMART CONTRACTS

Types of Analysis Methodologies	Analysis Technique
Static Analysis	Symbolic Execution
	Control Flow Graph Construction
	Pattern Recognition
	Rule-based Analysis
	Compilation
	Decompilation
Dynamic Analysis	Execution Trace at Run-time
	Transaction Graph Construction
	Symbolic Analysis
	Validation of True/False Positives
Formal Verification	Using Theorem Provers
	Translation of Formal Language
	Construction of Program Logics

Source: Adapted from [6]

IV. EVALUATION

In evaluating smart contract security analysis tools, it is important to consider their methodology and the type of analysis they perform. As shown in Table III, we have reviewed various types of analysis methods including static analysis, dynamic analysis, and formal verification, as well as the specific tools that employ these methods. For static analysis, the tools such as Oyente, Zeus, Vandal, EthIR, Securify, Slither, and Manticore use techniques such as symbolic execution, control flow graph construction, and pattern recognition to detect vulnerabilities. For dynamic analysis, tools like Gasper, Mythril, and Maian analyze smart contract execution traces at run-time to identify potential issues. Additionally, the formal verification tools like F*, FEther, and VeriSol use theorem provers, formal languages, and program logic to formally verify the correctness of smart contracts. Overall, choosing the appropriate tool and methodology will depend on the specific requirements and goals of the security analysis.

TABLE III
SMART CONTRACT SECURITY ANALYSIS TOOLS AND METHODOLOGIES

Analysis Type	Tool	Methodology
Static Analysis	Oyente	Symbolic Execution
Static Analysis	Zeus	Symbolic Analysis
Static Analysis	Vandal	Control Flow Graph Construction
Static Analysis	EthIR	Decompilation
Static Analysis	Securify	Rule-based Analysis
Static Analysis	Slither	Pattern Recognition
Static Analysis	Manticore	Symbolic Execution
Static Analysis	Gasper	Execution Trace at Run-time
Static, Dynamic Analysis	Mythril	Symbolic Execution
Dynamic Analysis	Maian	Execution Trace at Run-time
Formal Verification	F*	Using Theorem Provers
Formal Verification	FEther	Translation of Formal Language
Formal Verification	VeriSol	Construction of Program Logics

TABLE IV
OPEN-SOURCE STATUS AND REPOSITORY LINKS OF SMART CONTRACT SECURITY ANALYSIS TOOLS

Tool	Open-Source	Repository Link
Oyente	Y	https://github.com/enzymefinance/oyente
Zeus	N	N/A
Gasper	N	N/A
Vandal	Y	https://github.com/usyd-blockchain/vandal
EthIR	Y	https://github.com/costa-group/ethir
Securify	Y	https://github.com/eth-sri/securify
Maian	Y	https://github.com/ivicanikolicsg/MAIAN
F*	Y	https://secpriv.tuwien.ac.at/tools/ethsemantics
FEther	N	N/A
Mythril	Y	https://github.com/ConsenSys/mythril
Slither	Y	https://github.com/crytic/slither
VeriSol	Y	https://github.com/microsoft/verisol
Manticore	Y	https://github.com/trailofbits/manticore

Table IV presents the open-source status and repository links of the smart contract security analysis tools reviewed in this survey paper. As evident from the table, many of these tools,

including Oyente, Vandal, EthIR, Securify, Maian, F*, Mythril, Slither, VeriSol, and Manticore, are open-source and freely available on Github. The use of open-source tools is beneficial as it allows researchers and developers to collaborate, share knowledge, and make improvements, leading to more reliable and effective tools. Moreover, the use of open-source tools promotes transparency, allowing anyone to audit and verify the code for potential vulnerabilities. However, some tools, such as Zeus and FEther, are not open-source, which may limit their adoption and hinder transparency.

To avoid the vulnerabilities mentioned in smart contracts, it is important to follow a set of best practices throughout the development and deployment process. Table V gives all the best practices that one should follow to avoid or mitigate these vulnerabilities. These are given according to the categorization given above. By adhering to these best practices, developers can significantly reduce the likelihood of encountering vulnerabilities in smart contracts and enhance the overall security and reliability of their applications.

TABLE V
MITIGATION MEASURES FOR SMART CONTRACT VULNERABILITIES

Vulnerability Type	**Mitigation Measures**
Design and Implementation Flaws	Reviewing the code, performing extensive testing, secure coding practices
Cryptographic and Security Weaknesses	Employ up-to-date and secure cryptographic algorithms, proper key management practices, established security guidelines
Time-Related Risks	Commit-reveal schemes, randomness sources, or cryptographic techniques
External Interactions and Integration Risks	Secure coding practices (input validation, output sanitization, and proper access control)
Arithmetic and Numerical Issues	Using larger data types, safe mathematical libraries, and conducting thorough testing and auditing
Tooling and Platform Risks	Regular updates and patches applied to the tools and platforms

V. CONCLUSION

In conclusion, as smart contracts on Ethereum become increasingly important in the realm of distributed applications, ensuring their security is crucial to prevent losses and malicious attacks. Our survey has provided a comprehensive understanding of the security risks associated with smart contracts by classifying smart contract vulnerabilities into different categories based on their common characteristics or underlying causes. We have identified six major groups of smart contract vulnerabilities: Design and Implementation Flaws, Time-Related Risks, Cryptographic and Security Weaknesses, Arithmetic and Numerical Issues, External Interaction and Integration Risks, and Tooling and Platform Risks. These classifications help developers implement appropriate security measures and best practices to mitigate risks.

Additionally, our survey has categorized the security analysis methods into three approaches: static, dynamic, and formal verification. Although static and dynamic analysis methods offer automation tools that are easy to use, they are limited in scope, only detecting vulnerabilities based on predefined patterns. In contrast, formal verification methods leverage theorem provers to validate the correctness properties of smart contracts using interpreted proofs. However, these methods also have their own limitations, such as scalability and support for various blockchain platforms.

Future research should focus on developing new tools and techniques, improving the efficiency and effectiveness of existing tools, and investigating the security implications of emerging blockchain platforms. There is a growing interest in utilizing machine learning techniques and dynamic analysis methods to enhance vulnerability detection. Collaboration among researchers, developers, and industry practitioners is essential for addressing these challenges and maintaining trust in decentralized systems as blockchain adoption continues to grow.

REFERENCES

[1] Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). Lecture Notes in Computer Science, 164–186. https://doi.org/10.1007/978-3-662-54455-6_8

[2] Bowen Alpern and Fred B Schneider. "Defining Liveness". In: Information Processing Letters 21.4 (1985-10), pp. 181–185. issn: 0020-0190. doi:10.1016/0020-0190(85)90056-0

[3] Groce, A., Feist, J., Grieco, G., & Colburn, M. (2020). What are the Actual Flaws in Important Smart Contracts (And How Can We Find Them)? Financial Cryptography and Data Security, 634–653. https://doi.org/10.1007/978-3-030-51280-4_34

[4] DASP - TOP 10. (n.d.). DASP - TOP 10. https://dasp.co

[5] Rameder, H. (2021). Systematic review of ethereum smart contract security vulnerabilities, analysis methods and tools [Diploma Thesis, Technische Universität Wien]. reposiTUm. https://doi.org/10.34726/hss.2021.86784

[6] Praitheeshan, P., Pan, L., Yu, J., Liu, J.K., & Doss, R.R. (2019). Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey. ArXiv, abs/1908.08605.

[7] Praitheeshan, P., Pan, L., Yu, J., Liu, J.K., & Doss, R.R. (2019). Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey. ArXiv, abs/1908.08605.

[8] N. A. Noor Aidee, M. G. M. Johar, M. H. Alkawaz, A. Iqbal Hajamydeen and M. S. Hamoud Al-Tamimi, "Vulnerability Assessment on Ethereum Based Smart Contract Applications," 2021 IEEE International Conference on Automatic Control & Intelligent Systems (I2CACIS), Shah Alam, Malaysia, 2021, pp. 13-18, doi: 10.1109/I2CACIS52118.2021.9495892.

[9] Marchesi, L., Marchesi, M., Pompianu, L., & Tonelli, R. (2020). Security checklists for ethereum smart contract development: patterns and best practices. arXiv preprint arXiv:2008.04761.

[10] He, D., Deng, Z., Zhang, Y., Chan, S., Cheng, Y., & Guizani, N. (2020). Smart contract vulnerability analysis and security audit. IEEE Network, 34(5), 276-282.

[11] Rameder, H., Di Angelo, M., & Salzer, G. (2022). Review of automated vulnerability analysis of smart contracts on Ethereum. Front. Blockchain, 5.

[12] Luu, L., Chu, D.H., Olickel, H., Saxena, P. and Hobor, A., 2016, October. Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 254-269).

[13] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N. and Zanella-Béguelin, S., 2016, October. Formal verification of smart contracts: Short paper. In Proceedings of the 2016 ACM workshop on programming languages and analysis for security (pp. 91-96)

[14] Brent, L., Jurisevic, A., Kong, M., Liu, E., Gauthier, F., Gramoli, V., Holz, R. and Scholz, B., 2018. Vandal: A scalable security analysis framework for smart contracts. arXiv preprint arXiv:1809.03981.

[15] Nikolić, I., Kolluri, A., Sergey, I., Saxena, P. and Hobor, A., 2018, December. Finding the greedy, prodigal, and suicidal contracts at scale. In Proceedings of the 34th annual computer security applications conference (pp. 653-663).

[16] Chen, T., Li, X., Luo, X. and Zhang, X., 2017, February. Under-optimized smart contracts devour your money. In 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER) (pp. 442-446). IEEE.

[17] Albert, E., Gordillo, P., Livshits, B., Rubio, A. and Sergey, I., 2018, September. Ethir: A framework for high-level analysis of ethereum bytecode. In Automated Technology for Verification and Analysis: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings (pp. 513-520). Cham: Springer International Publishing.

[18] Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F. and Vechev, M., 2018, October. Securify: Practical security analysis of smart contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 67-82).

[19] Kalra, S., Goel, S., Dhawan, M. and Sharma, S., 2018, February. Zeus: analyzing safety of smart contracts. In Ndss (pp. 1-12).

[20] Delmolino, K., Arnett, M., Kosba, A., Miller, A. and Shi, E., 2016. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers 20 (pp. 79-94). Springer Berlin Heidelberg.