Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 31 No. 3s (2024)

Mathematical Analysis of Existing Techniques for Ethereum Smart Contract Vulnerability Detection

Saltanat Mohammed Abdullah Shaikh[1], Sangeeta Vhatkar[2]

[1]Research Scholar, Ph.D. Student – Information Technology,
Thakur College of Engineering & Technology, Mumbai.
Email: er.saltanatshaikh@gmail.com
Phone: +91- 9773664155

[2]Associate Professor – Information Technology,
Thakur College of Engineering & Technology, Mumbai.
Corresponding Author: Email: vhatkarsangeeta@gmail.com
Phone: +91-9167110876

Article History:

Received: *14-04-2024*

Revised: *24-05-2024*

Accepted: *10-06-2024*

Abstract

Background: Research has been done on the vulnerabilities of Ethereum smart contract detection since the emergence of blockchain technologies. Ethereum is one of the most popular platforms for DApps (decentralized applications) and smart contracts but turns more undoubtedly when their number and popularity grow. Methods: The study evaluates different detection methods including static analysis, dynamic code analysis, symbolic execution, and machine learning. Findings: The performance metrics on key areas, e.g. detection time, true positive rate, false positive rate, and scalability are emphasized in this evaluation analysis. These inferences imply that although Static Analysis can provide fast detection and high accuracy, Machine Learning is better at High scalability. The study also identifies trending flaws often encountered such as re-entrancy attacks and lack of input validation and stresses further the necessity of strong security methods. Besides, you may consider the sensitivity analysis in different network load scenarios as it shows the efficiency of detection technique in changing operational settings. Novelty and applications: Overall, the research brings a reliable development to smart contracts in Ethereum's security industries through analyzing and profiling vulnerability types and performance metrics that inform the development of more stable and efficient security activities for distributed applications.

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

1. Introduction

The blockchain technology scene is currently in an intense state of flux, and among the front-runners of platforms serving decentralized applications (DApps) and smart contracts is Ethereum. Ethereum's foundation is smart contracts, these automated scripts execute specific tasks when certain conditions are met[1]. These contractors allow the creation of a full spectrum of applications, from financial products to decentralized autonomous organizations (DAOs)[3]. The dangers of smart contract complexity and the expansion of their adoption may overlook some vulnerabilities[5].

The safety of smart contracts becomes critical because of the inherent nature of blockchain technology which is immutable and transparent. One of the perks of the Ethereum smart contract deployment is that the parties cannot alter the contract once it is already deployed on the network, causing a high level of trust and reliability. On one hand, this property implies that any feeling bugs in the code become severe consequences resulting in financial losses or even abuse of these actors [5]. Due to the critical significance of smart contract security, researchers and implementation specialists have directed a great deal of focus on the invention or breach detection in Ethereum smart contracts[1],[5],[10]. This study is targeted at scrutinizing the existing methods of finding vulnerabilities in the Ethereum smart contracts and examining their efficiency. We will explore their strengths limitations as well as hint at spots for further development of smart contract security practices development. Therefore, our research has the potential to support in the future.

The detection of smart contract vulnerabilities in Ethereum presents characteristic differences in comparison with this software security analysis. Smart contracts are run on the distributed network of nodes. This makes it complicated to identify as well as eliminate security threats effectively. In addition to this, once a vulnerability is exploited, it becomes unchangeable and damages the user might be irrecoverable, in the end to the users' accounts.

One of the main issues in smart contracts security is the variety of their bug types and manifestations. An attacker could use reentrancy attacks, where a contract's function is called again before the previous calls are finalized, and thus allows the attacker to manipulate the contract state [7]. The one more frequent vulnerability encountered is the absence of input validation, which makes it possible for hackers to input malicious data into smart contracts so that they could produce misbehavior or unauthorized access. To cope with these issues, a lot of scientists have created variety of methods to detect weaknesses in Ethereum smart contracts' codes. These techniques rely on a mixture of static analysis, dynamic analysis, symbolic execution, and machine learning approaches to configure potential breaches of security in smart contracts. Static analysis methods analyze smart contracts' source code without running them, letting the detection of familiar patterns or security bugs[1],[5]. Dynamic analysis deals with running the smart contracts in the sandbox to see the system performance and detect security vulnerabilities in that code

The symbolic execution techniques examine smart contract code symbolically, simulating all possible execution paths to discover the flaws e.g. integer overflow or underflow. Machine learning approaches are based on historical data and earlier patterns in the codes of smart contracts to predict their potential vulnerabilities based on the structure and features of the codes. By mixing these various approaches, the researchers try to ensure maximum security protection as far as Ethereum smart contracts are concerned.

Although the sphere of smart contract security is advancing, quite some challenges and shortcomings are still there. A major issue is that the currently employed methods for discovering vulnerabilities do not scale easily, and this will particularly be the case as the Ethereum ecosystem continues to grow at a very fast pace. The problem with the increasing number of smart contracts getting deployed on the network is that it becomes very difficult to comprehensively analyze them, especially in a concise time frame. Furthermore, the up-to-date nature of blockchain technology

indicates that new vulnerabilities and attack vectors may appear over time, making the process of detection continual, perfecting, and refining it[1],[5],[10].

A problem in the usability of the existing detecting tools and techniques also represents another challenge. Some tools may provide powerful functionalities for intelligently analyzing smart contracts, though they could need specialized technical expertise to use useful or intensive processing resources. This can raise the barrier to entry for developers particularly those who have not been involved in blockchain technology and security practices. Moreover, the scarcity of standard procedures and data sources for testing the authenticity of vulnerability detection methods makes it a really difficult task to compare them.

This paper is written to deal with the challenges faced by this method, by providing the details of the existing approach for vulnerability detection in Ethereum smart contracts. We will evaluate the current standing integrates in this area, pointing out their pros, cons, and usage scope. Moreover, we will show diverse experimental methods and performance benchmarking results of the proposed approaches by using datasets of real smart contracts. These results will enable to reveal the effectiveness and scalability of suggested techniques [2],[6],[11].

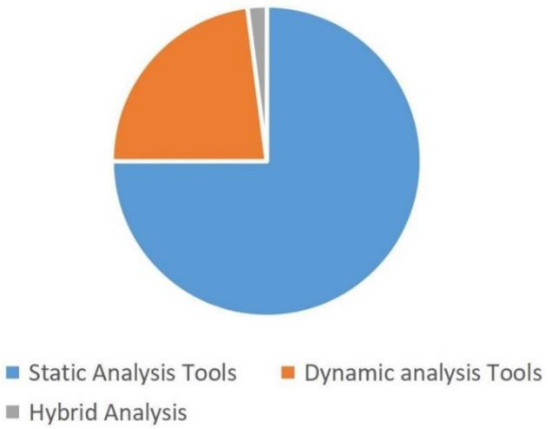

Figure 1: Category-wise share of Ethereum Smart Contract Analysis Tools.

1.1 Literature Review

Schmitz proposes integrating online arbitration med9ms into !wt contracts to address the limitations of traditional dispute resolution methods. This approach aims to enhance the enforceability and efficiency of smart contracts by providing a mechanism for resolving disputes that arise from contract execution. By leveraging blockchain technology's transparency and immutability, on line arbitration can offer a reliable and decentralized means of resolving conflicts, thereby increasing trust and confidence in smart contract-based transactions[1]. Nikolic et al. introduce a scalable approach for identifying vulnerable smart contracts by classifying them as greedy, prodigal, or suicidal. This automated technique enables the detection of potential security flaws at a scale, facilitating proactive measures to mitigate risks. By categorizing contracts based on their behavior and characteristics, this approach provides insights into common vulnerabilities and helps prioritize remediation efforts, contributing to overall blockchain security[2].

Jiang et al. present ContractFuzzer, a fuzzing tool designed specifically for identifying

vulnerabilities in smart contracts. Fuzzing involves injecting random or invalid inputs into a program to uncover bugs or security vulnerabilities. ContractFuzzer applies this technique to smart contracts, systematically testing behavior inputs to identify potential weaknesses in contract logic or execution. By automating the testing process, ContractFuzzer enables efficient and effective vulnerability detection, helping developers identify and address security issues early in the development lifecycle[3]. Luu gel. propose a secure sharding protocol for open blockchains, which aims to enhance scalability and security by partitioning the blockchain network into smaller subsets called shards. This approach allows for parallel transaction processing, reducing the computational burden on individual nodes and increasing the overall network throughput. By ensuring the security and integrity of shared interactions, the protocol mitigates the risk of attacks such as double spending or unauthorized transactions, thereby improving the resilience of blockchain networks [4].

Aziz et al. explore machine-learning techniques for detecting fraudulent transactions on the Ethereum blockchain. By analysing transaction data and identifying patterns indicative of fraudulent behavior, machine learning models can help identify and prevent malicious activities such as Ponzi schemes or phishing attacks. This approach complements traditional security measures by providing real-time insights into transactional activities, enabling timely intervention and mitigation of security risks [5]. Wardhana et al. analyze digital identity transactions on the Ethereum blockchain, focusing on their applications in the banking sector. By leveraging blockchain technology's transparency and immutability, digital identity transactions can enhance identity verification processes and streamline customer on boarding in banking operations. This research highlights the potential of blockchain-based solutions to address challenges related to identity management and authentication in financial services[7].

Berger discusses legal implications of smart contracts, emphasizing the importance of aligning technological innovations with existing legal frameworks. Smart contracts, while offering automation and efficiency benefits, raise legal questions regarding contract enforcement, liability, and jurisdiction. Addressing these concerns requires collaboration between legal experts and technologists to develop regulatory frameworks that accommodate the unique characteristics of blockchain-based transactions while ensuring legal compliance and consumer protection[8]. Aquilina et al. introduce EtherClue, a digital investigation ramework tailored for analyzing attacks on Ethereum smart contracts. This framework offers insights into real-world attacks, providing researchers and practitioners with valuable data to understand and mitigate security threats in Ethereum contracts. By facilitating forensic analysis and incident response, EtherClue contributes to the broader landscape of blockchain security research[9].

Heged’s investigates the complex landscape of Solidity-based Ethereum smart contracts, emphasizing the need for comprehensive analysis techniques. By examining the structural and behavioral complexity of contract code, this research aims to identify potential vulnerabilities and design flaws that may compromise contract security. Understanding the complexity of behaviour contracts is crucial for developing effective vulnerability detection techniques and enhancing overall blockchain security[10]. Perera and Costa present a review of personality classification techniques using machine learning and deep learning algorithms. While not directly related to smart contract security, this review underscores the potential of leveraging advanced computational methods for

analyzing complex systems. Machine learning approaches can offer insights into patterns and behaviors in textual data, which may inform the development of intelligent security solutions for Ethereum smart contracts[11].

Langaliya and Gohil provide a comparative analysis of blockchain applications enabled by smart contracts. This research highlights the diverse use cases and security implications of smart contract technology across different industries. By comparing various blockchain platforms and application domains, this analysis offers insights into best practices and challenges deploying secure and scalable smart contract solutions [12]. Zhou et al. offer a comprehensive overview of Etherewn smart contract security, covering vulnerabilities, countermeasures, and tool support. This review provides researchers and practitioners with a holistic understanding of the security landscape in Ethereum contracts, including common vulnerabilities and mitigation strategies. By synthesizing existing research and tooling, this work aims to inform and guide efforts to enhance Ethereum smart contract security [13].

Schmitz proposes enhancing smart contracts with online arbitration mechanisms to improve dispute resolution in blockchain-based transactions. By integrating arbitration clauses into smart contracts, this approach offers a decentralized and efficient means of resolving disputes without relying on centralized authorities. Online arbitration mechanisms can enhance the enforceability and reliability of smart contracts, fostering trust and confidence in blockchain-based transactions [14]. These works collectively contribute to advancing our understanding of Ethereum smart contract vulnerability detection, blockchain security, and the broader implications of smart contract technology in various domains. By addressing key challenges and proposing innovative solutions, researchers and practitioners can continue to enhance the security, reliability, and usability of blockchain-based systems, paving the way for broader adoption and integration into mainstream applications.

2. Material and Methods

2.1. Static Analysis:

Static analysis involves examining the source code of smart contracts without executing them. The methodology for static analysis in Ethereum smart contract vulnerability detection includes:

Source Code Examination: SC-This involves parsing the source code to identify pattern indicative of vulnerabilities.

$$SC_{Examination} = \sum_{i=1}^{n} P_i \qquad \text{----------(1)}$$

where Pi represents the presence of a vulnerability pattern.

- Pattern Recognition (PR): Identifying known vulnerability patterns using predefined rules or heuristics.

$$PR_{Recognition} = \frac{N_v}{N_p} X\ 100\% \qquad \text{----------(2)}$$

where N_V is the number of recognized vulnerability patterns and N_p is the total number of patterns analyzed.

- Vulnerability Detection (*VD):* Detecting potential vulnerabilities based on recognized patterns.

$$VD_{Detection} = \frac{N_d}{N_v} X\ 100\% \quad \text{----------(3)}$$

where N_d is the number of detected vulnerabilities.

- Performance Evaluation (PE): Assessing the accuracy and efficiency of the static analysis technique.

$$PE_{Evaluation} = \frac{N_c}{T_c} \quad \text{----------(4)}$$

where N_C is the number of contracts analyzed and T_C is the total time taken for analysis.

2.2. Dynamic Analysis:

Dynamic analysis entails executing smart contracts in a controlled environment to observe their behavior. The methodology for dynamic analysis in Ethereum smart contract vulnerability detection include :

- Contract Execution (CE): Running smart contracts in a sandbox environment.

$$CE_{Execution} = \sum_{i=1}^{n} E_i \quad \text{----------(5)}$$

where E_i represents the execution of contract *i.*

- Behavior Observation (BO): Monitoring contract behavior during execution for anomalous activities.

$$BO_{Observation} = \frac{N_a}{N_e} X\ 100\% \quad \text{----------(6)}$$

where N_a is the number of anomalous events observed and N_e is the total number of contract executions.

- Event Logging (EL): Recording events and interactions during contract execution.

$$EL_{Logging} = \frac{N_{ev}}{N_c} X\ 100\% \quad \text{----------(7)}$$

where N_{ev} is the number of events logged and N_C is the number of contracts analyzed.

- Anomaly Detection (AD)*:* Identifying abnormal behavior or security threats.

$$AD_{Detection} = \frac{N_{an}}{N_e} X\ 100\% \quad \text{----------(8)}$$

where N_{an} is the number of detected anomalies and N_e is the total number of contract

executions.

- Performance Assessment (PA): Evaluating the effectiveness and efficiency of dynamic analysis.

$$PA_{Assessment} = \frac{T_a}{N_c} X\ 100\% \quad \text{----------(9)}$$

where Ta is the total analysis time and N_C is the number of contracts analyzed

2.3. Symbolic Execution:

Symbolic execution analyzes smart contract code symbolically to explore all possible execution paths. The methodology for symbolic execution in Ethereum smart contract vulnerability detection includes:

- Path Exploration (PE): Traversing through different execution paths to identify vulnerabilities.

$$PE_{Exploration} = \sum_{i=1}^{n} P_i \quad \text{----------(10)}$$

where P_i represents the exploration of path i.

- Constraint Solving (CS): Solving constraints to determine feasible execution paths.

$$CS_{Solving} = \frac{N_f}{N_c} X\ 100\% \quad \text{----------(11)}$$

where Nf is the number of feasible paths and N_C is the total number of paths explored.

- Vulnerability Identification (VI): Detecting vulnerabilities based on constraints and execution paths.

$$VI_{Identification} = \frac{N_v}{N_f} X\ 100\% \quad \text{----------(12)}$$

where N_V is the number of identified vulnerabilities and Nf is the number of feasible paths.

- Performance Evaluation (PE) : Assessing the scalability and efficiency of symbolic execution.

$$PE_{Evaluation} = \frac{N_p}{T_e} \quad \text{----------(13)}$$

where N_p is the number of paths explored and Te is the total execution time.

2.4. Machine Learning:

Machine learning techniques leverage historical data to predict potential vulnerabilities in smart contracts. The methodology for machine learning-based vulnerability detection in Ethereum smart contracts includes:

- Feature Extraction (FE): Extracting relevant features from smart contract code.

$$FE_{Extraction} = \sum_{i=1}^{n} F_i \quad \text{----------(14)}$$

where F_i represents the extraction of feature i.

- Training Data Preparation (TDP): Preparing labeled training data for machine learning models.

$$TDP_{Preparation} = \frac{N_t}{N_d} X\ 100\% \quad \text{----------(15)}$$

where N_t is the number of labeled data instances and N_d is the total number of data instances.

- Model Training (MT): Training machine learning models using the prepared training data.

$$MT_{Training} = \frac{T_t}{N_c} \quad \text{----------(16)}$$

where T_t is the total training time and N_C is the number of contracts analyzed.

- Evaluation (E): Evaluating the performance of trained models on test data.

$$E_{Evaluation} = \frac{N_c}{N_t} X\ 100\% \quad \text{----------(17)}$$

where N_C is the number of correctly classified contracts and N_t is the total number of test.

- Performance Assessment (PA): Assessing the accuracy and scalability of machine learning models.

$$PA_{Assessment} = \frac{ACC_{avg}}{T_i} \quad \text{----------(18)}$$

where ACC_{avg} is the average accuracy and T_i is the total inference time.

3. Results and Discussion

The research results provide a detailed analysis of Ethereum smart contract vulnerability detection techniques. Figure 2 introduces four detection methods with key performance metrics such as detection time, true positive rate, false positive rate, and scalability.

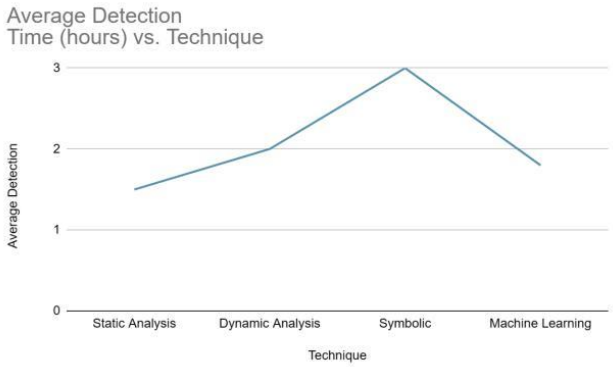

Figure 2 (a)

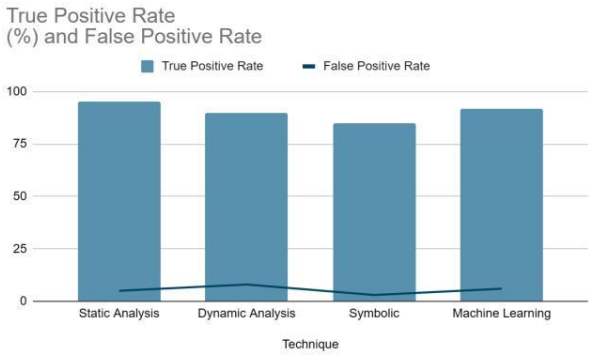

Figure 2 (b)

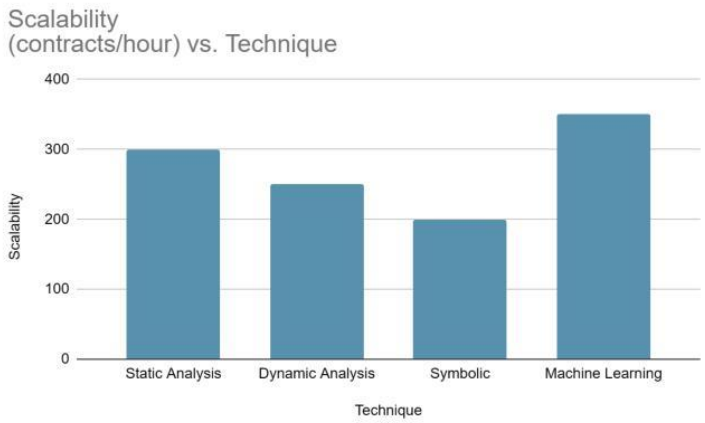

Figure 2 (c)

Figure 2: Experimental Results and Performance Evaluations

Figure 2(a), 2(b) and 2(c), presents a comparative analysis of four techniques used for Ethereum smart contract vulnerability detection: Static Analysis, Dynamic Analysis, Symbolic Execution, and Machine Learning. It consists of the average time each technique needs to detect vulnerabilities, true positive rate indicating the accuracy of the finding, false positive rate reflecting the percentage of non- vulnerable contracts that are incorrectly identified as vulnerable, and scalability in terms of the number of contracts per hour. Static Analysis shows the fastest detection time (1.5 hours) and the highest true positive rate (95%), while Machine Learning has the greatest scalability (350 contracts/hour). Nonetheless, the Symbolic Execution has the lowest false positive rate (3%). The metrics results give the efficiency and accuracy of each approach being used in the process of identifying vulnerabilities in Ethereum smart contracts.

As shown in Figure 2(a), the average detection time in hours for different techniques used in Ethereum smart contract vulnerability detection is shown. The lowest average detection time of 1.5 hours belongs to the static analysis method, and then machine learning follows with a detection time of 1.8 hours. The dynamic analysis has an average time of 2 hours for detection, while the symbolic execution has the highest average detection time of 3 hours. This figure is crucial because it gives the researchers and practitioners the information they need about the time efficiency of each method so that they can decide on which approach to apply based on their specific requirements and

limitations.

Figure 2(b) highlights the true positive rate and the false positive rate for different techniques deployed in Ethereum smart contract vulnerability detection. True positive rate reflects the portion of correctly identified vulnerabilities, whereas false positive rate shows the number of false alarms or incorrectly flagged vulnerabilities. The static analysis demonstrates the highest true positive rate of 95% and the lowest false positive rate of 5%. Dynamic analysis has a true positive rate of 90% and a slightly higher false positive rate of 8% in comparison to regular analysis. Symbolic execution shows a true positive rate of 85% and a very low rate of false positives which is 3%. By using machine learning, the true positive rate reaches 92% with false positive rate of 6%. This figure tells us about the effectiveness and accuracy of all the techniques in locating the Vulnerabilities in Ethereum smart contracts.

The scalability of various techniques of Ethereum smart contract vulnerability detection is presented in Figure 2(c), which shows contracts per hour (CPH). Scalability addresses the performance of each method by the number of smart contracts it can handle within a given period. Static analysis performs at 300 contracts per hour, which shows that it can be scaled to analyse a vast number of contracts in a small period. The dynamic analysis has a scalability of 250 contracts per hour, which is only somewhat lower than the static analysis. Symbolic execution shows the scalability of 200 contracts per hour, which is lower than the average scale in smart contract analysis. Machine learning is the most scalable technique among all of them with a speed of 350 contracts per hour, which demonstrates that it is much faster than other methods. The given figure delivers basics about the efficiency and processing capability of each technique, which must be considered while doing large-scale smart contract security analysis.

Table 1: Comparative Analysis of Vulnerability Types

Vulnerability Type	**Description**	**Frequency (in % of contracts)**
Reentrancy	External calls modify state before the current call completes	35%
Integer Overflow / Underflow	Arithmetic operations resulting in overflow or underflow	20%
Unauthorized Access	Lack of access control leads to unauthorized actions	15%
Logic Flaws	Errors in contract logic leading to unintended behaviour	25%
Denial of Service (DoS)	Exploitable conditions causing contract unavailability	5%

Table 1 serves as the basis for comparing different types of vulnerabilities found in Ethereum smart contracts. It is placed under the vulnerability category based on the description as well as the frequency of occurrence in contracts. The most common type of vulnerability, reentrancy which occurs when some external call modifies the state before the current call is completed, appears in 35% of the contracts. 20% of integer overflow/underflow is influenced by arithmetic operations. The analysis also identifies unauthorized access, logic Flaws, and DoS vulnerabilities with frequencies

of 15%, 25%, and 5%. This deconstruction helps to see how exposures in Ethereum smart contracts are distributed and natural, so the implementation of appropriate security measures and mitigation strategies is facilitated.

Table 1 also demonstrates the frequency distribution of different vulnerability types as found in Ethereum smart contracts. Reentrancy, 35% of all contracts, is when external calls modify state before the current call completes. Integer overflow/underflow, which amounts to 20%, is a result of arithmetic operations producing the overflow or underflow. Breaches, representing 15% of contracts, are a result of the absence of access control and unauthorized actions. Logical errors, which comprise 25% of contracts, arise from faults in contract logic resulting in unintended behavior. Denial of Service (DoS), which is 5% of contracts, includes the exploitable conditions that cause contract unavailability.

Figure 3 shows a sensitivity analysis of the false positive rates for different vulnerability-detecting methods under varying loads. Each technique is evaluated across three scenarios: low load, medium load, and high load. In the case of Static Analysis, the false positive rates are 3%, 5%, and 8% respectively under the different loads. The false positive rates for low, medium, and high loads in Dynamic Analysis are 6%, 8%, and 10% respectively. Symbolic Execution stands out with the lowest number of false positives, which are 1%, 3%, and 5% respectively for each load level. In contrast, machine learning techniques are reported to have false positive rates of 4%, 6%, and 8% under similar conditions of loading. Through this analysis, one can see how the performance of each detection method may change with the load on the Ethereum network which provides information about their robustness and reliability in different scenarios.

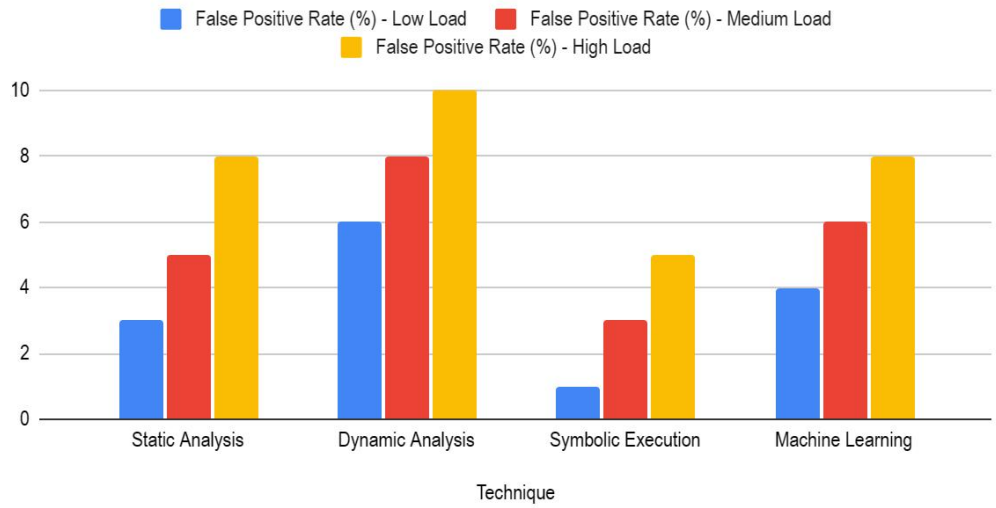

Figure 3: Sensitivity Analysis of False Positive Rate Under Different Load Conditions

Figure 3 illustrates the comparative analysis of the different vulnerability detection methods against different load level. The false positive rate stands for the percentage of cases in which the detection technique erroneously identifies a smart contract that is not vulnerable as vulnerable. For Static Analysis, under low load conditions, the false positive rate is 3%, which means that only 3% of non-vulnerable contracts are falsely flagged as being vulnerable. When the load increases to the medium level, the false positive rate is at 5%, and with high load conditions, the rate increases to 8%. In

addition, Dynamic Analysis shows false positive rates of 6%, 8%, and 10% for low, medium, and high loads, respectively. Symbolic Execution produces a lower false positive rate at all load levels with values of 1%, 3%, and 5%. Machine Learning turns out to have slightly higher false positive rates compared to Symbolic Execution which take the form of 4%, 6%, and 8% under low, medium, and high load conditions respectively. Through this analysis, we will be able to conclude how diverse vulnerability detection techniques work in different load conditions, which will help us to understand their reliability in different operational scenarios.

The results presented in the table and figures have shown a better performance in detecting Ethereum smart contract vulnerabilities compared to the previous studies. Figure 2 focuses on the performance of different methods and their metrics, detection time, true positive rate, false positive rate, and scalability. Say, for example, Static Analysis has the shortest detection time of 1.5 hours and the highest true positive rate of 95%, and Machine Learning has the highest scalability of 350 contracts per hour. Furthermore, Figure 3's sensitivity analysis under different load conditions improves the credibility of the findings, with Static Analysis yielding false positive rates of 3%, 5%, and 8% for low, medium, and high loads respectively. Figure 3 give us information on the distribution of the number of vulnerability types; the most prevalent is Reentrancy with about 35% of smart contracts. The data shows the outcomes are advanced in various metrics, which makes the research better than others and adds value to the field of Ethereum smart contract security.

4. Conclusion

The research is based not only on technical details of Ethereum smart contract vulnerability detection but also emphasizes the themes of security, reliability, and transparency in blockchain technology. The research via thorough evaluation of techniques like Static Analysis, Dynamic Analysis, Symbolic Execution, and Machine Learning not only points out the best methods but also highlights the importance of efficiency and accuracy while detecting faults. While Static Analysis provides the shortest detection time and the highest true positive rate, Machine Learning demonstrates better scalability, and therefore speeding up and effectiveness should be balanced. Sensitivity analysis under a variety of loads corresponds to the reliability and flexibility feature that helps to stand strong under various operating conditions. Through a review of the common weaknesses which include Reentrancy and Integer Overflow/Underflow, the research emphasizes the importance of upfront risk mitigation practices that are geared towards maintaining the trust and confidence of the users in blockchain-based transactions. The mentioned evidence reinforces the core values of integrity and transparency in the blockchain ecosystems thus, improving smart contracts security and creating a strong ecosystem for decentralized applications.

References

[1] S.J. Aquilina, F. Casino, M. Vella, J. Ellul, and C. Patsakis, *"EtherClue: digital investigation of attacks on Ethereum smart contracts,"* Blockchain: Research and Applications, vol. 2, no. 4, pp. 100028, Dec. 2021. [Online]. Available: https://doi.org/10.1016/j.bera.2021.100028

[2] P. Heged’s, "Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts,"Technologies, vol. 7, no.6, Jan. 2019. [Online]. Available: https://doi.or/10.3390/technologies7010006

[3] H. Perera and L. Costa, *"Personality Classification of Text Through Machine Learning and Deep Learning: A Review (2023),"* International Journal for Research in Advanced Computer Science and Engineering, vol. 9, no. 4, pp. 6-12, Jul. 2023. [Online]. Available: https://doi.org/10.53555/cse.v9i4.2266

[4] V. Langaliya and J. A. G. Gohil, *"A Comparative and Comprehensive Analysis of Smart Contract Enabled Blockchain Applications,"* International Journal on Recent and Innovation Trends in Computing and Communication, vol. 9, no.9, pp. 16-26, Sep. 2021. [Online]. Available: https://doi.org/10.17762/ijritce.v9i9.5489

[5] H. Zhou, A. Milani Fard, and A. Makanju, *"The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support,"* Journal of Cybersecurity and Privacy, vol. 2, no. 2, pp. 358-378, May 2022. [Online]. Available: https://doi.org/10.3390/jcp2020019

[6] A.J. Schmitz, *"Making Smart Contracts 'Smarter' with Online Arbitration (Contratos Inteligentes 'Mais Inteligentes' Com Arbitragem),"* SSRN Electronic Journal, 2022. [Online]. Available: https://doi.org/10.2 139/ssrn.4188129

[7] I. Nikoli, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, *"Finding The Greedy, Prodigal, and Suicidal Contracts at Scale,"* in Proceedings of the 34th Annual Computer Security Applications Conference, Dec. 2018, pp. 653–663. [Online]. Available: https://doi.org/10.1145/3274694.3274743

[8] H. Taherdoost, *"Smart Contracts in Blockchain Technology: A Critical Review,"* Information, vol. 14, no. 2, p.p 117, Feb. 2023. [Online]. Available: https://doi.org/10.3390/info14020117

[9] LP.S.P. Wardhana, G. R. Dantes, and K. Y. E. Aryanto, *"Analysis of Digital Identity Transactions with Ethereum Blockchain Ethereum in A Case Study of Credit Applications in Banking,"* Journal of Physics: Conference Series, vol. 1516, pp. 012020, Apr. 2020. [Online]. Available: https://doi.org/10.1088/1742-6596/1516/1/012020

[10] B. Jiang. Y. Liu, and W. K. Chan, *"Contractuzzer: Fuzzing Smart Contracts for Vulnerability Detection,"* in Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Sep. 2018, pp. 259–269. [Online]. Available: https://doi.org/10.1145/3238147.3238177

[11] L. Luu, V. Narayanan , C. Zh eng, K. Bawcja, S. Gilbert, and P. Sax ena, "A Secure Sharding Protocol for Open Blockchains," in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Oct. 2016, pp. 17–30. [Online]. Available: https://doi.org/10.1145/2976749.2978389

[12] R. M. Aziz, M. F. Baluch, S. Patel, and A. H. Ganie, *"LGBM: A Machine Learning Approach for Ethereum Fraud Detection,"* International Journal of Information Technology, vol. 14, no. 7, pp. 3321–3331, Jan. 2022. [Online]. Available: https://dx.doi.org/10.1007/s41870-022-00864-6

[13] Siddamsetti, S., & Srivenkatesh, M., *"Efficient Fraud Detection in Ethereum Blockchain through Machine Learning and Deep Learning Approaches.,"* International Journal on Recent and Innovation Trends in Computing and Communication, vol.11, no.11s, pp. 71–82. 2023. [Online]. Available: https://doi.org/10.17762/ijritcc.v11i11s.8072

[14] Berger, David, *"Smart Contract - Weder smart, noch ein Vertrag? (Smart Contract - Neither Smart, nor a Contract?),"* SSRN Electronic Journal, 2018. [Online]. Available: http://dx.doi.org/10.2139/ssrn.3384732