Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
DOI : 10.47974/JIOS-2207

A comprehensive security framework for Ethereum smart contracts : Integrating static analysis and dynamic testing

Satpal Singh Kushwaha †
Department of Computer Science and Engineering
Manipal University Jaipur
Jaipur 303007
Rajasthan
India

Anubhav Saxena *
Department of IoT and Intelligent Systems
Manipal University Jaipur
Jaipur 303007
Rajasthan
India

Abstract

Ethereum smart contracts, pivotal components of decentralized applications, are increasingly susceptible to various security vulnerabilities. To address these challenges, this paper presents a unified security approach that combines static analysis and dynamic testing techniques. Static analysis, with its capability to scrutinize code without execution, offers a comprehensive detection of vulnerabilities by analyzing the smart contract's source code. Dynamic testing, on the other hand, involves executing the code in a controlled environment to uncover runtime issues that static analysis might overlook. Our approach leverages the strengths of both techniques to enhance the overall security assessment of Ethereum smart contracts. Static analysis is employed to identify potential security flaws in the codebase, such as reentrancy, integer overflow, and access control issues. Subsequently, dynamic testing, through fuzzing and symbolic execution, validates these findings and detects additional vulnerabilities by observing the contract's behavior during execution. The integration of these methodologies provides a more robust and comprehensive security analysis framework. We demonstrate the effectiveness of our hybrid approach through highlighting its superiority in identifying and mitigating security threats compared to traditional single-method analyses. This unified security approach not only enhances the detection of vulnerabilities but also

† *E-mail:* satpal.singh@jaipur.manipal.edu
* *E-mail:* anubhav1cse@gmail.com (Corresponding Author)

© Taru Publications

contributes to the development of more secure and reliable smart contracts in the Ethereum ecosystem.

Subject Classification: *68M25, 68N30.*

Keywords: *Static analysis, Dynamic analysis, Smart contract, Ethereum, Vulnerability detection.*

1. Introduction

Blockchain technology enables decentralized, immutable transaction systems, with Ethereum as a leading smart contract platform. Smart contracts are self-executing programs that enforce agreements when predefined conditions are met without intermediaries. However, security flaws in smart contracts can lead to financial losses and harm trust in blockchain systems. Since blockchain is immutable and contracts cannot be modified after deployment, thorough security analysis is essential beforehand. Traditional Ethereum security analysis uses static and dynamic methods. Static analysis examines code for vulnerabilities without execution, while dynamic analysis detects runtime issues during execution. Each approach has limitations: static analysis may miss runtime flaws, while dynamic analysis may overlook vulnerabilities or produce false positives due to incomplete execution coverage. This work proposes a unified security approach that combines static analysis and dynamic testing to overcome these limitations, improving vulnerability detection, reducing false positives, and ensuring safer smart contract deployment.

1.1 *Motivation*

Ethereum smart contracts play a critical role in financial transactions and blockchain agreements, making security essential. Traditional methods such as static analysis and dynamic testing have limitations: static analysis detects code vulnerabilities but may miss runtime issues, while dynamic testing identifies execution faults but may not cover all paths. Therefore, a combined approach using both methods is needed. This hybrid technique improves vulnerability detection, reduces false positives, and strengthens smart contract security before deployment, ensuring safer blockchain applications.

1.2 *Contributions*

Through a unified security methodology that incorporates static analysis and dynamic testing, this study advances Ethereum smart contract security:

- **Proposed and implemented a hybrid framework:** Combining static code analysis and dynamic testing approaches. This integrated approach uses both approaches to examine Ethereum smart contract security more thoroughly.
- **Improved Vulnerability Detection:** Our system employs static analysis and dynamic testing to enhance detection accuracy. Static analysis finds codebase flaws, whereas dynamic testing confirms them and finds runtime vulnerabilities, strengthening the evaluation.
- **Reduction in false positives and negatives:** A major issue in single-method analysis. Security evaluations are more accurate and smart contracts are properly checked for dangers.

The proposed method fills vulnerability detection gaps and ensures smart contract reliability and safety.

1.3 *Outline*

The structure of the paper is as follows: Section 2 provides an literature review of various research methodologies employed for analyzing Ethereum smart contracts. In Section 3, we describe our proposed unified security approach in detail. Section 4 presents the evaluation of our approach, including a comparison with existing methods. Finally, Section 5 presents the limitations and future research and section 6 presents the conclusion.

2. Related Work

Since the introduction of Ethereum, several methods have been proposed to analyze its blockchain and smart contracts. [1] Used Gigahose IR to convert low-level EVM bytecode into a high-level intermediate representation for static analysis, mainly to detect gas-related issues. [2] Applied a three-phase static analysis approach involving SIF assertion insertion, input generation for instrumented contracts, and execution analysis. [3] Propsed a static analysis technique to identify gas-consuming EVM bytecode patterns, categorized as loops and redundant code, using symbolic execution. [4] Analyzed Solidity code through SlithIR using control flow graphs, data flow analysis, and taint tracking to detect vulnerabilities. [5] Introduced symbolic execution–based static analysis that adds policy predicates as assertions and converts contracts into LLVM bytecode for verification. [6] Proposed Oyente, a dynamic analysis tool

that uses symbolic execution to explore smart contract execution paths. [7] Used static analysis to identify critical financial transaction paths that may lead to security risks. [8] Developed a static method that simulates EVM execution to show bytecode behavior, stack status, and control flow. [9] Applied XPath pattern matching by converting Solidity code into XML to detect vulnerabilities. [10] Proposed a static analysis approach using Invocation Relationship Analysis and Logic Risk Expansion and Location, which maps function invocation relationships to identify logical errors.

3. Proposed Methodology

The proposed approach evaluates Ethereum smart contract security using both static and dynamic analysis. Static analysis parses Solidity code into an Abstract Syntax Tree (AST), applies security rules, examines data and control flow, and uses formal verification to ensure correctness. Dynamic testing generates and executes test cases, including fuzz testing, to analyze runtime behavior and detect hidden issues. Results from both analyses are combined into a unified security report that ranks identified vulnerabilities by severity, as shown in Figure 1.

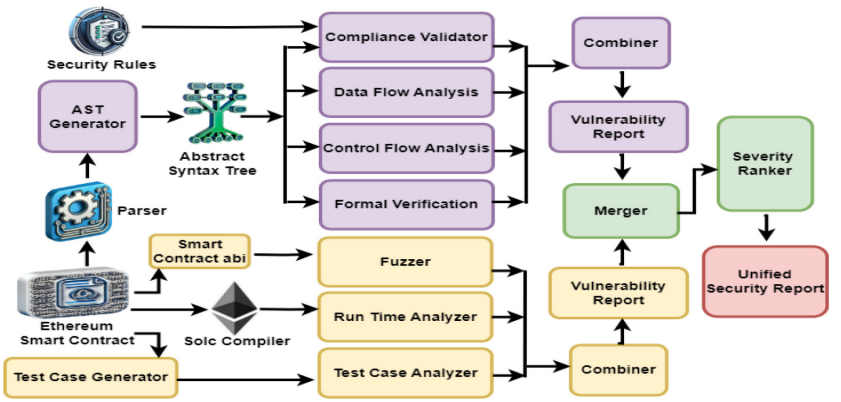

Figure 1

Proposed model architecture

Unified Security Algorithm (*C*)

Input: Ethereum Smart Contract Solidity Source Code ***C.***
Output: Unified Security Report V_r.

Step 1: Static Analysis Engine

1.1. $P \leftarrow$ **Parse (**C**)** $//P$ is the parsed representation of C.

1.2. $A \leftarrow$ **Generate AST from** P $//A$ is the **AST** generated from P.

1.3. $R_s \leftarrow$ **Check rules on** A $//R_s$ represents the rule violations identified in A.

1.4. $D_f \leftarrow$ **Analyze data flow in** A $//D_f$ represents the data flow analysis results.

1.5. $C_f \leftarrow$**Analyze control flow in** A $//C_f$ represents the control flow analysis results.

1.6. $V \leftarrow$ **Perform formal verification on** A $//V$ represents the formal verification results.

Step 2: Dynamic Testing

2.1. $T_c \leftarrow$ Generate test cases for C $//T_c$ represents the generated test cases.

2.2. $T_d \leftarrow$ Deploy C in testing $//T_d$ represents the testing environment deployment.

2.3. $T_e \leftarrow$ Execute tests on C $//T_e$ represents the test execution results.

2.4. $F \leftarrow$ Perform fuzz testing on C// F represents the fuzz testing results.

2.5. $R_a \leftarrow$ Analyze runtime behavior of C $//R_a$ represents the runtime analysis results.

Step 3: Unified Security Report

3.1. $F_c \leftarrow$ Consolidate findings from (R_s, D_f, C_f, V, T_c, T_d, T_e, F, R_a) $//F_c$ represents the consolidated findings.

3.2. $S_r \leftarrow$ ***Rank*** severity of issues in // S_r represents the severity ranking.

3.3. $V_r \leftarrow$ Compile a report detailing identified vulnerabilities $//V_r$ represents the vulnerability report.

End Algorithm

4. Results and Discussion

We compiled a dataset consisting of 1381 Ethereum smart contract solidity source codes that are susceptible to vulnerabilities. These codes were obtained from the Ether-scan API [11] and have been marked with 3193 vulnerabilities. The evaluation is conducted on two elements of the proposed approach: vulnerability detection and minimizing false positives. To facilitate comparison, we have chosen the following existing

approaches: Securify [12], Mythril [13], Oyente [6], Slither [4], and SmartCheck [9].

Table 1
Comparison of Analysis Tools for Vulnerability Detection Capabilities

	Proposed model	Mythril	Oyente	Securify	Smartcheck	Slither
Detected	2906	2509	1917	1245	2020	2570
Not Detected	287	684	1276	1948	1173	623

Table 2
Comparison of Analysis Tools for Detection Capabilities of the Top Five Vulnerabilities

	Classified Vulnerabilities	Proposed model	Mythril	Oyente	Securify	Smartcheck	Slither
Arithmetic Underflow	**1182**	989	933	922	----	743	953
Timestamp Dependancy	**352**	296	277	274	229	237	291
Reentrency	**925**	774	729	721	520	539	740
Self-Destruct	**288**	240	226	----	193	196	229
Tx.origin	**447**	374	341	----	291	304	358

Table 1 and 2 displays the comparative outcomes of the proposed methodology and existing methodologies on the established Dataset. The results demonstrate that the proposed approach is more efficient than existing approaches and technologies in terms of vulnerability detection. Figure 2 presents a comparison of the detection capabilities between the proposed work and existing tools and methodologies. The suggested approach successfully detects 91% of vulnerabilities in the dataset that was developed. The current techniques have a high likelihood of producing false positives since they primarily prioritize detecting the highest number of security flaws and their coverage. Figure 2 displays the comparative outcomes of the proposed work's false positive rate in contrast to existing

methodologies. The proposed strategy yields a lower number of incorrect positive results in comparison to alternative methods.

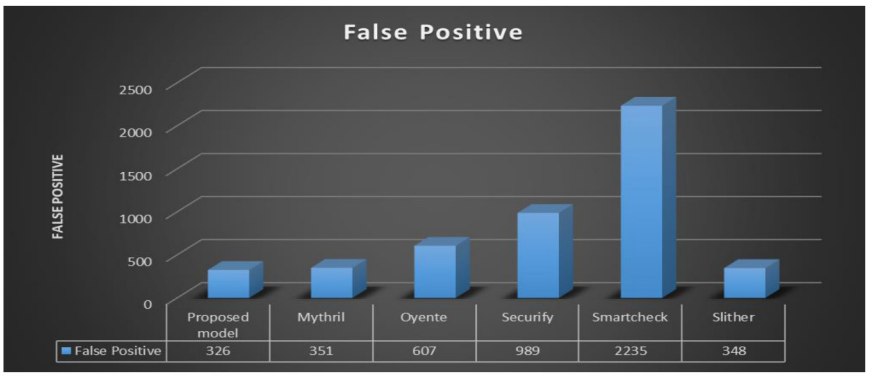

Figure 2

False positive comparison

5. Limitations and Future Research Directions

While the proposed model achieves high vulnerability detection accuracy and a significantly lower false-positive rate compared to existing tools, several limitations remain. The evaluation is based on a curated dataset of 1,381 Ethereum smart contracts, which may not fully represent the diversity and complexity of contracts deployed in real-world environments. Additionally, the analysis focuses primarily on known vulnerability classes, leaving emerging or zero-day vulnerabilities largely unexplored. Future research will involve expanding the dataset to include more diverse and complex contract patterns, integrating adaptive detection techniques to identify unknown vulnerabilities [14], [15], [16], and testing the model within live or near-real-time blockchain environments. Further improvements may also explore combining the proposed approach with dynamic analysis tools and incorporating machine learning–based classifiers to enhance precision and reduce false positives even further.

6. Conclusion

Ethereum smart contracts are essential for financial transactions and blockchain agreement enforcement, thus protecting them is critical. Due to their limitations, static code analysis and dynamic execution testing-only methods are insufficient. Dynamic testing may not cover all execution

paths or codebase vulnerabilities, whereas static analysis may miss execution-only issues. Our integrated security strategy uses static analysis and dynamic testing to overcome these limits. This hybrid method improves vulnerability discovery, reduces false positives, and enhances security evaluation. Through this integrated technique, we want to create a reliable foundation for Ethereum smart contract analysis and security before deployment. Our research demonstrates that this integrated method dramatically improves smart contract security, making blockchain applications safer and more trustworthy. Next steps will include improving this methodology, adding advanced detection methods, and expanding its applicability to more blockchain platforms and contract categories.

References

[1] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Art. no. 116, pp. 1–27 (2018), doi: 10.1145/3276486.

[2] S. Akca, A. Rajan, and C. Peng, "Solanalyser: A framework for analysing and testing smart contracts," in *Proc. 26th Asia-Pacific Softw. Eng. Conf. (APSEC),* Auckland, New Zealand, pp. 482–489 (Dec. 2019).

[3] T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, and X. Zhang, "GasChecker: Scalable analysis for discovering gas-inefficient smart contracts," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1433–1448 (Jul.–Sep. 2021).

[4] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB),* Montreal, QC, Canada, pp. 8–15 (May 2019).

[5] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS),* San Diego, CA, USA, pp. 1–12 (Feb. 2018).

[6] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Vienna, Austria, pp. 254–269 (Oct. 2016).

[7] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, "sCompile: Critical path identification and analysis for smart contracts," in *Proc.*

Int. Conf. Formal Eng. Methods (ICFEM), Cham, Switzerland: Springer, pp. 286–304 (Oct. 2019).

[8] R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, "Visual emulation for Ethereum's virtual machine," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Taipei, Taiwan, pp. 1–4 (Apr. 2018).

[9] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, Gothenburg, Sweden, pp. 9–16 (May 2018).

[10] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security assurance for smart contract," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Paris, France, pp. 1–5 (Feb. 2018).

[11] Etherscan, "Etherscan: The Ethereum Blockchain Explorer." [Online]. Available: https://etherscan.io. [Accessed: Nov. 5, 2022].

[12] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, pp. 67–82 (Oct. 2018).

[13] Mythril, "Mythril: Security analysis tool for EVM bytecode," GitHub repository. [Online]. Available: https://github.com/ConsenSysDiligence/mythril. [Accessed: Jun. 12, 2024].

[14] V. P. Singh, S. S. Biswas, B. Alankar, and S. Tanweer, "Cryptographic modeling and discrete structures for intrusion detection in Ethereum smart contracts," *J. Discrete Math. Sci. Cryptogr.*, vol. 28, no. 8, pp. 3039–3048 (2025).

[15] A. Dadhich, B. Keswani, and D. Goyal, "Comparative analysis of a novel smart contract-based hybrid access control model for blockchain-enabled secure IoT home automation systems," *J. Discrete Math. Sci. Cryptogr.*, vol. 28, no. 7, pp. 2875–2888 (2025).

[16] V. Ramachandran, H. M. A. Ghanimi, B. Marapelli, N. Kaur, M. R. Laxmi, R. K. Bommisetti, S. Sengan, and P. Dadheech, "An enterprise blockchain model: A reliable cryptography-based cyber-physical systems for securing user data," *J. Discrete Math. Sci. Cryptogr.*, vol. 28, no. 5-B, pp. 2103–2114 (2025).

Received December, 2025