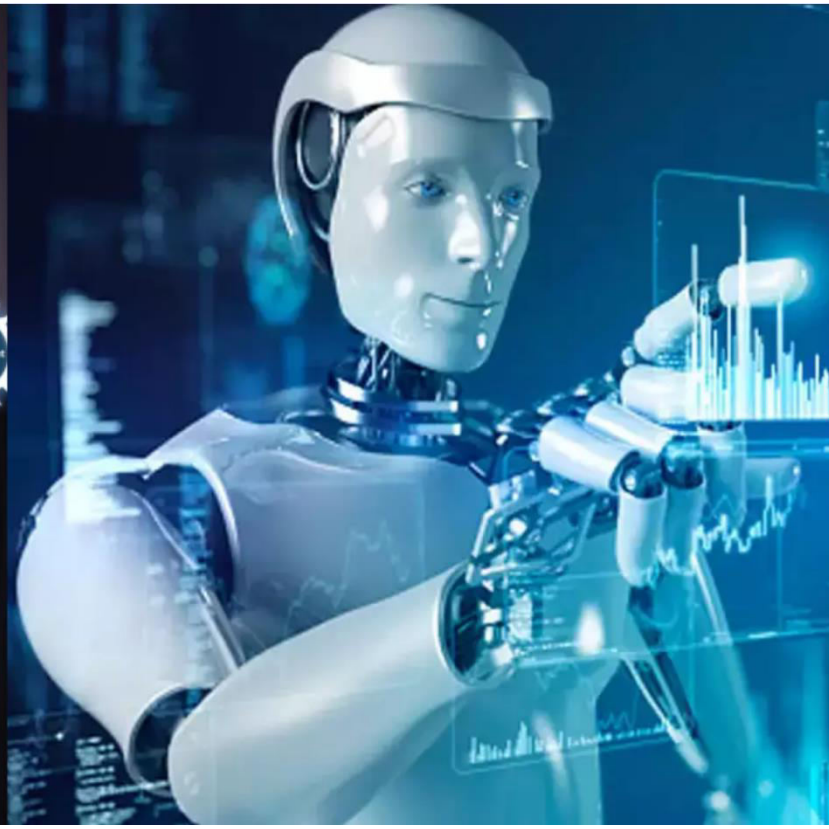
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 14, Issue 7, July 2025

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

Smart Contract Defense: Practical Security Measures for Blockchain

P. Preethy Jemima, Dr P Privietha, Dr. Sylvia Grace J, Dr. Srideivanai Nagarajan, Dr. Balapriya S

Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Chennai, India

Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Chennai, India

Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Chennai, India

Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Chennai, India

Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Jeppiaar Nagar, Chennai, India

ABSTRACT: Smart contracts, which are implemented on decentralized blockchain systems to provide transparency, security, and immutability, are self-executing contracts with terms directly encoded into software. By removing middlemen and increasing efficiency and trust, these contracts automatically carry out predetermined activities once criteria are satisfied. Finance, supply chain management, real estate, and gaming are among the key uses.

Notwithstanding their potential, smart contracts have some serious drawbacks, the most significant of which being security flaws. Incidents like the DAO hack on Ethereum provide as examples of how bugs, re-entrancy attacks, and unrestricted external calls have resulted in large financial losses. Widespread use requires addressing these issues.

Developers should do thorough testing, including unit, integration, and edge case tests, and adhere to published recommendations such as Ethereum Smart Contract Security Best Practices in order to address security vulnerabilities in smart contracts. Thorough audits of both internal and external code are crucial, as are post-audit corrections. While proper management of external interactions, such as avoiding external calls and validating return values, lowers risks, security tools like Certora, MythX, and Slither can assist in identifying vulnerabilities. Security is further improved by employing multi-signature wallets for high-stakes transactions, using Checks-Effects-Interactions patterns and mutex locks to mitigate reentrancy attacks, and using Flashbots to stop transaction front-running.

Additional protections include fallback procedures for recovering from unforeseen failures, monitoring solutions, and reusable contracts for emergency halts. When combined, these steps provide a strong foundation for risk reduction, enhanced security, and guaranteeing the dependability of smart contracts in practical implementations.

KEYWORDS: Smart contracts, blockchain, security vulnerabilities, code audits, reentrancy attacks, decentralized finance (DeFi).

I. INTRODUCTION

Agreements that are legally enforceable are called contracts. An agreement becomes a contract when it meets all the requirements for a legally binding agreement. An agreement can never be a legally binding contract if it is missing one of the necessary components

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

[1]. Contracts were written and signed by hand in the past, necessitating physical documentation—what we refer to as paper-based documentation—which is expensive and slow. To get around these issues, businesses started automating certain aspects of contracts (such bills or subscriptions) with digital systems, but these needed a reliable central authority to handle data [2]. Simple conditional transactions were made possible by early software automation and digital payment networks, but there was little transparency. The idea was easily adaptive to the decentralized nature of blockchain technology [3].

Nick Szabo, a cryptographer and computer scientist, first proposed the idea of smart contracts in 1994. Without the use of middlemen, Szabo envisioned a digital protocol that would enable, validate, or enforce a contract's terms. His research established the theoretical underpinnings of smart contracts and highlighted how they could automate and improve the effectiveness of business and legal agreements. He compared it to a vending machine, which, when certain requirements (money inserted) are fulfilled, automatically carries out the agreed transaction (dispenses a product) [4].

The emergence of decentralized apps (dApps) demonstrated how smart contracts may be used to automate a wide range of tasks, including asset trading and lending. Cutting-edge systems like Uniswap, Aave, and Compound were made possible by smart contracts. A new need for smart contracts that could automate financial transactions, lending, borrowing, and exchanges without the use of banks or brokers was brought about by the emergence of decentralized finance (DeFi) [5].

Bitcoin, which introduced blockchain technology in 2009, offered a transparent, decentralized, and impenetrable data storage method. Because blockchain's immutability and transparency could be used to automate and enforce contract terms without the need for a central authority, this provided the ideal framework for the creation of smart contracts. Launched in 2015 by Vitalik Buterin and associates, Ethereum expanded blockchain technology by introducing Solidity, a Turing-complete programming language that made smart contract development possible. Ethereum's blockchain enabled the development of decentralized apps (dApps) and the autonomous execution of sophisticated logic on the chain[6,7].

The intermediates were the primary goal; they can be weak, expensive, and compromised. However, cryptographic evidence was implemented since trustless systems were becoming more and more popular. Smart contracts can operate in a way that ensures security and transparency without requiring faith in any one party because to blockchain's use of encryption[8].

II. LITERATURE SURVEY

Making sure smart contracts are free of errors and vulnerabilities is essential to preserving the integrity of decentralized apps (dApps), especially in light of high-profile occurrences like the DAO breach. Particularly in big codebases, traditional approaches to identifying vulnerabilities in Solidity-based smart contracts may miss minor problems and have significant false positive rates. Transformer-based deep learning models, which have demonstrated impressive performance in natural language processing (NLP) tasks, are used to increase the detection accuracy. They modify these models, which are commonly employed in text processing, in order to comprehend and identify possible weaknesses in the code of smart contracts. Compared to earlier static analysis methods, which were constrained by pattern-based analysis, these models can detect vulnerabilities more reliably by being refined using pertinent smart contract datasets. [9].

Solidity is made to support Ethereum's decentralized architecture, allowing special features like data types, storage formats, and contract inheritance that affect the overall security and efficacy of smart contracts. Code complexity and how it affects contract vulnerabilities (e.g., accidental overflows, gas inefficiency). Solidity contracts' state management, including memory and storage usage, is essential to comprehending security and performance issues. utilization of empirical data from actual Solidity codebases to examine the practical applications of specific features. Errors and misuse are more likely to occur with certain Solidity features.

How developers reuse and expand smart contract code could be used to evaluate Solidity's object-oriented design, particularly its inheritance capabilities. The application of statistical analysis to Solidity code, potentially through Etherscan or other smart contract analysis platforms. The characteristics of solidity are used, along with any possible

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

drawbacks. Ideas for improving Solidity's usability, security, and dependability in next iterations. For instance, they could suggest changes to the language's structure to facilitate developers' creation of safe and effective smart contracts [10].

- Due to their immutability once implemented on the blockchain, smart contracts are vulnerable to a number of security flaws that, if taken advantage of, might result in large financial losses (e.g., re-entrancy attacks, integer overflows, and access control issues).
- Automating the identification of vulnerabilities in smart contract code through the use of deep learning models, specifically neural networks. Stress that deep learning can assist in identifying vulnerabilities that are otherwise challenging to detect using conventional techniques because of its capacity to process big datasets and learn intricate patterns.
- To train the deep learning models, the authors compiled a large dataset of smart contract code tagged with known vulnerabilities. For the model to learn how to distinguish between secure and susceptible code, this dataset is essential.

Convolutional neural networks (CNNs), which are popular in natural language processing jobs and can be modified for smart contract code analysis, or transformer-based models (like BERT or GPT) should be used. [13]

III. METHODOLOGY

Smart contract security is achieved by a multi-step process that combines rigorous testing methods with safe development processes. First, pre-audited libraries and standardized design patterns are used in trusted and dependable programming frameworks to reduce common vulnerabilities. A systematic testing procedure is then implemented, starting with unit testing to confirm the accuracy of separate parts separately. After that, integration testing is carried out to make sure that various modules work together seamlessly and to find problems that unit tests could miss. Additionally, edge case testing is used to evaluate how well the contract performs in unusual or extreme circumstances, enhancing its usability and dependability. Fuzz testing is carried out with sophisticated tools like FunFuzz to improve vulnerability detection even further. This function-based fuzzer separates and evaluates every.Using symbolic execution and mutation-based approaches, this function-oriented fuzzer creates targeted and varied test cases by independently isolating and testing each function. This method improves efficiency and efficacy in identifying known and unknown vulnerabilities by removing unnecessary tests and concentrating on probable failure sites. The approach places a strong emphasis on spotting important risks at every stage, including as defects in consensus methods, network-level assaults like Sybil and 51% attacks, and smart contract vulnerabilities.

IV. IMPLEMENTATION

A.Simple Steps for Securing Smart Contracts

1. Use Secure Frameworks: To steer clear of common blunders, begin with reliable programming frameworks.

2. Unit, Integration, and Edge Case Testing: Verify the functionality of each component separately. Make sure all the parts fit together perfectly. Verify resilience by simulating extreme or uncommon situations.

Unit testing, which tests individual components separately, is usually the initial step in the traditional testing technique. The interaction between the merged components is then confirmed through integration testing. Better Bug Detection: Interface-related problems that unit tests might miss are found via integration testing. Faster Feedback Loops: Agile teams gain from the ability to detect system-level flaws more quickly. Improved Test Prioritization: The scope and design of upcoming unit tests are influenced by integration testing.[16].

Emphasizes finding and fixing vulnerabilities like:

- Smart contract errors.
- Attacks at the network level, such as the Sybil and 51% attacks.
- The consensus algorithm's shortcomings [15].

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

3. Fuzz Testing: To find vulnerabilities using random inputs, use automated tools. By separating distinct functions within smart contracts and applying fuzzing techniques to each function independently, the FunFuzz approach uses function-oriented fuzzing. This technique finds function-specific flaws that conventional fuzzing techniques might overlook. The tool generates inputs based on code analysis and the behavior of the contract by utilizing symbolic execution to investigate different execution paths. By eliminating redundant testing, concentrating on the most likely places where vulnerabilities arise, and creating targeted test cases that put particular functionalities under stress, FunFuzz increases efficiency. The program generates a variety of test scenarios using random input generation and mutation-based fuzzing, which aids in identifying potential vulnerabilities under various circumstances. Evaluations on actual Ethereum smart contracts showed that FunFuzz outperformed conventional fuzzers in terms of detection rates and performance, detecting both known and unknown vulnerabilities more successfully. Its main features include function isolation, which enables a more detailed and focused investigation of contract security, high efficacy in identifying missed vulnerabilities, and high efficiency in lowering testing overhead. As a result, it is a useful tool for developers [17].

Table 1: TYPES OF TESTS AND ITS PURPOSE

Method	**Purpose**
Unit tests	Quick, cheap, easy debugging
Integration tests	Identify issues between components.
Fuzz tests	Uncover unexpected edge cases
Edge Case Testing	to improve the quality, reliability, and usability of a system.

4. Code Audits: Conduct in-depth security assessments both internally and outside.

5. Security Tools: Utilize code analysis tools such as Slither, MythX, and Certora. By examining and validating their code, tools such as Certora, MythX, and Slither are essential for improving the security of smart contracts. By logically demonstrating how smart contracts behave under predetermined situations, Certora is used for formal verification, guaranteeing the accuracy of smart contracts.

- MythX: Using a combination of static and dynamic analysis, MythX offers automatic vulnerability detection to find problems including overflow vulnerabilities, gas limit concerns, and reentrancy.
- **Slither** : A static analysis tool called Slither provides quick and comprehensive analysis, identifying inefficiencies and security vulnerabilities in Solidity programs. These tools are crucial for safe blockchain development because they work together to help developers find important flaws, improve contract performance, and guarantee the stability of smart contracts. [18].
- Oytene : Not as well-known as Certora, MythX, or Slither, Oytene is another tool for analyzing and securing smart contracts. Although Oytene might not be as well-known, its primary purpose is probably the same as those of the other tools you listed: offering security analysis for smart contracts, identifying vulnerabilities, and making sure the code complies with performance and safety best practices[19].

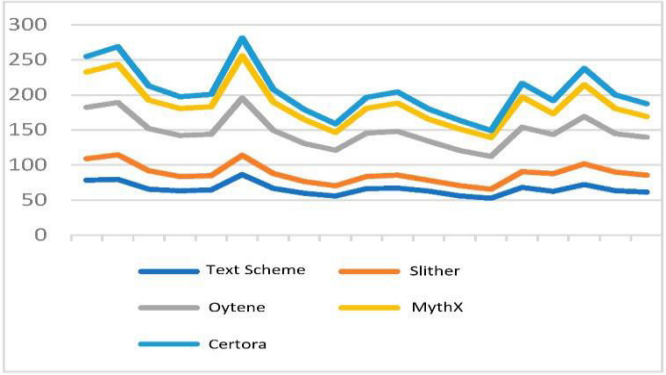


Fig1. Comparison of Security tools

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

Certora, the most popular, is the greatest choice because of its formal verification features. MythX is a great option, particularly for in-the-moment analysis while development is underway. If you're already using Solidity, Slither is perfect.

TABLE II. PROS AND CONS OF THE TOOLS

Tools	Advantage	Disadvantage
Certora	Formal verification, customizable rule and error detection	Complexity, resource-intensive,limited coverage
MythX	Easy integration, vulnerability detection, detailed reports	Speed and limited free usage
Slither	Open source and wide coverage	Limited to static analysis, no formal verification.

6. Minimize External Calls: Minimize and verify communication with unreliable systems.

7. Secure Coding Patterns: Employ mutex locks and adhere to Checks-Effects-Interactions. To avoid typical vulnerabilities in smart contracts, secure coding patterns like Checks-Effects-Interactions and the use of mutex locks are crucial. By ensuring that the contract's state is updated prior to engaging with external systems, the Checks-Effects-Interactions design lowers the possibility of reentrancy attacks, in which an external call could change the contract's state before the function is finished. Additionally, mutex locks make sure that sensitive actions are non-reentrant and impossible for attackers to exploit by preventing numerous function invocations at once. By reducing the possibility of unexpected behavior and enhancing the contract's resistance to frequent security risks, these patterns help writers create safe, dependable smart contracts.

8. Reentrancy : Before the initial transaction is finished, the external contract (attacker's contract) then invokes the function of the susceptible contract once again. The attacker can repeatedly invoke the withdrawal function and drain funds because the contract's state hasn't been changed yet. To Prevent Reentrancy- Use patterns and locks to block reentrant attacks. Mitigating Reentrancy Attacks by:

A. Checks-Effects-Interactions Pattern: Prior to engaging with external contracts, always update the contract's state (such as balances). This prevents the attacker from reentering the function by guaranteeing that any modifications to the contract state are finished prior to the external call.

Fig 2. A diagrammatic representation of Checks-Effects-Interactions

B. Using Reentrancy Guards: To stop re-entry into the function, use a mutex lock (reentrancy guard). Usually, a modification that prevents more function calls if the contract is currently running is used to do this.

C. Use of Pull Payments: You can prevent the possibility of an external contract calling back into the susceptible function by allowing users to "pull" their cash by calling a different claim function rather than transferring them directly within the contract.

9. Multisig Wallets: Multiple approvals should be needed for critical actions. The security of cold wallets used by centralized cryptocurrency exchanges can be enhanced by native multi-signature (multisig) systems. Cold wallets are

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

susceptible to single points of failure, like compromised private keys, even if they keep cryptocurrencies offline for increased protection. In order to allow sensitive actions, including transaction signing, the protocol suggested in this article incorporates multisig technology into the cold wallet management system and requires multiple approvals from various devices and administrators. By using this cooperative strategy, the system reduces the possibility of unwanted access and enhances the general security of the assets kept in cold wallets, guaranteeing that no one person may manage or access the money on their own [19].

10. Monitor and Update: Contracts should be monitored on a continuous basis and updated as needed.

V. CONCLUSION

In summary, ensuring the dependability and security of blockchain applications requires safeguarding smart contracts. By employing a systematic approach that includes the use of secure frameworks, extensive testing (unit, integration, edge case, and fuzz testing), comprehensive code audits, the use of powerful security tools like Certora, MythX, and Slither, and adherence to secure coding patterns, developers can decrease common vulnerabilities and increase contract robustness. Reducing the quantity of external calls, implementing reentrancy protection, utilising multisignature wallets for significant transactions, and routinely evaluating and updating contracts are other security methods. By following these best practices, developers may create smart contracts that are more secure, reliable, and efficient, hence enhancing the overall trust and integrity of the blockchain ecosystem.

This study demonstrates the effectiveness of Augmented Reality as a transformative tool in architectural visualization. By allowing users to interact with 3D models in real time through mobile devices, the developed AR application bridges the gap between technical design and user comprehension. The system's marker-based approach, built using Unity and Vuforia, offers a lightweight, affordable, and user-friendly solution that enhances spatial understanding and design communication. While current features focus on model display and basic interaction, the results confirm AR's potential to improve client engagement, streamline feedback loops, and support more informed design decisions. As AR technology continues to evolve, its integration into architectural workflows will likely expand, enabling richer, more immersive, and collaborative design experiences. This positions AR not only as a visualization tool but as a vital component in the future of architecture and construction.

VI. FUTURE SCOPE

By combining machine learning and artificial intelligence approaches, smart contract security can be further improved in the future by anticipating and identifying weaknesses based on past attack patterns. There is great potential for the creation of increasingly complex fuzzing tools that can replicate actual attack scenarios and produce security updates on their own. Furthermore, developers may be able to mathematically demonstrate the accuracy and security of their contracts if formal verification techniques are adopted more widely. As multi-blockchain ecosystems and interoperability protocols gain traction, cross-chain smart contract security will likewise become an increasingly important field. Building user trust and expediting the verification process can be achieved through the development of automated auditing systems and standardized security certifications.To keep ahead of new dangers and guarantee strong smart contract security, developers, researchers, and auditors must collaborate and conduct ongoing research as blockchain technology advances.

REFERENCES

[1] Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., & Bani-Hani, A. (2021). Blockchain smart contracts: Applications, challenges, and future trends. Peer-to-peer Networking and Applications, 14, 2901-2925.
[2] Ante, L. (2021). Smart contracts on the blockchain–A bibliometric analysis and review. Telematics and Informatics, 57, 101519.
[3] Vacca, A., Di Sorbo, A., Visaggio, C. A., & Canfora, G. (2021). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. Journal of Systems and Software, 174, 110891.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 14, Issue 7, July 2025

|DOI: 10.15680/IJIRSET.2025.1407041|

[4] Singh, R., Gupta, A., & Mittal, P. (2023, November). A Deep Dive into Smart Contracts and Their Emerging Applications. In International Conference on Intelligent Vision and Computing (pp. 72-83). Cham: Springer Nature Switzerland.M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[5] Wang, S., Ouyang, L., Yuan, Y., Ni, X., Han, X., & Wang, F. Y. (2019). Blockchain-enabled smart contracts: architecture, applications, and future trends. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 49(11), 2266-2277.
[6] Taherdoost, H. (2023). Smart contracts in blockchain technology: A critical review. Information, 14(2), 117.
[7] Singh, J., Rani, S., & Kumar, P. (2024, April). Blockchain and Smart Contracts: Evolution, Challenges, and Future Directions. In 2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS) (Vol. 1, pp. 1-5). IEEE.
[8] Spataru, A. L., Pungila, C. P., & Radovancovici, M. (2021). A high-performance native approach to adaptive blockchain smart-contract transmission and execution. Information Processing & Management, 58(4), 102561.
[9] Kim, J., Lee, S., & Kim, H. (2024). Robust Vulnerability Detection in Solidity-Based Ethereum Smart Contracts Using Fine-Tuned Transformer Encoder Models. IEEE Access.
[10] Wang, Z., Chen, X., Zhou, X., Huang, Y., Zheng, Z., & Wu, J. (2021, December). An empirical study of solidity language features. In 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 698-707). IEEE.
[11] Umucu, E. H. (2021). Solidity: Smart Contract Language or Legal Contract Language. Available at SSRN 3916072.
[12] Chaliasos, S., Gervais, A., & Livshits, B. (2022). A study of inline assembly in solidity smart contracts. Proceedings of the ACM on Programming Languages, 6(OOPSLA2), 1123-1149.
[13] Tang, X., Du, Y., Lai, A., Zhang, Z., & Shi, L. (2023). Deep learning-based solution for smart contract vulnerabilities detection. Scientific Reports, 13(1), 20106.
[14] Rosaire, S. M., & Jules, D. (2022). Smart contracts security threats and solutions. International Journal of Information Technology and Web Engineering (IJITWE), 17(1), 1-30.
[15] Lal, C., & Marijan, D. (2021). Blockchain testing: Challenges, techniques, and research directions. arXiv preprint arXiv:2103.10074.
[16] Shahabuddin, S. M., & Prasanth, Y. (2016). Integration testing prior to unit testing: A paradigm shift in object oriented software testing of agile software engineering. Indian Journal of Science and Technology, 9(20), 1-10.
[17] Ye, M., Nan, Y., Dai, H. N., Yang, S., Luo, X., & Zheng, Z. (2024). FunFuzz: A Function-Oriented Fuzzer for Smart Contract Vulnerability Detection with High Effectiveness and Efficiency. ACM Transactions on Software Engineering and Methodology, 33(7), 1-20.
[18] Feist, J., Grieco, G., & Groce, A. (2019, May). Slither: a static analysis framework for smart contracts. In 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB) (pp. 8-15). IEEE.
[19] Ebrahimi, S., Hasanizadeh, P., Aghamirmohammadali, S. M., & Akbari, A. (2021). Enhancing cold wallet security with native multi-signature schemes in centralized exchanges. arXiv preprint arXiv:2110.00274.
[20] Çağlayan Aksoy, P. (2022). Smart contracts: to regulate or not? Global perspectives. Law and Financial Markets Review, 16(3), 212-241.
[21] Cong, L. W., & He, Z. (2019). Blockchain disruption and smart contracts. The Review of Financial Studies, 32(5), 1754-1797.
[22] Alharby, M., & Van Moorsel, A. (2017). Blockchain-based smart contracts: A systematic mapping study. arXiv preprint arXiv:1710.06372.
[23] Watanabe, H., Fujimura, S., Nakadaira, A., Miyazaki, Y., Akutsu, A., & Kishigami, J. (2016, January). Blockchain contract: Securing a blockchain applied to smart contracts. In 2016 IEEE international conference on consumer electronics (ICCE) (pp. 467-468). IEEE.
[24] Singh, A., Parizi, R. M., Zhang, Q., Choo, K. K. R., & Dehghantanha, A. (2020). Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities. Computers & Security, 88, 101654.
[25] Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C., & Santamaría, V. (2018). Blockchain and smart contracts for insurance: Is the technology mature enough?. Future internet, 10(2), 20.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details