

Navigating the smart contract threat landscape: a systematic review

Unyime Ufok Ibekwe¹, Uche M. Mbanaso², Nwojo Agwu Nnanna¹, Umar Adam Ibrahim³

¹Department of Computer Science, Nile University of Nigeria, Abuja, Nigeria

²Center for Cyberspace Studies, Nasarawa State University, Keffi, Nigeria

³Software Engineering and Information Technology Department, Nile University of Nigeria, Abuja, Nigeria

Article Info

Article history:

Received May 2, 2024

Revised Sep 11, 2024

Accepted Sep 30, 2024

Keywords:

Blockchain

Cyberattacks

Cybersecurity

Smart contracts

Vulnerability

ABSTRACT

Smart contracts have emerged as a transformative technology within the blockchain ecosystem, facilitating the automated and trustless execution of agreements. Their adoption spans diverse sectors such as education, agriculture, healthcare, government, real estate, transportation, supply chain, and global initiatives like Central Bank Digital Currencies (CBDCs). However, the security of smart contracts has become a significant concern, as vulnerabilities in their design and implementation can lead to severe consequences such as financial losses and system failures. This systematic review consolidates findings from 78 selected research articles, identifying key vulnerabilities affecting smart contracts and categorizing them into a taxonomy encompassing code-level, environment-dependent, and user-related vulnerabilities. It also examines the threats that exploit these vulnerabilities and the most effective detection techniques. The domain-based classification presented in this review aims to assist researchers, software engineers, and developers in identifying and mitigating significant security flaws related to the design, implementation, and deployment of smart contracts. A comprehensive understanding of these issues is essential for enhancing the security and reliability of the blockchain ecosystem, ultimately fostering the development of more secure and robust decentralized applications for end users.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Unyime Ufok Ibekwe

Department of Computer Science, Nile University of Nigeria

Plot 681, Cadastral Zone C, OO, Research and Institution Area

Airport Road, Jabi, Abuja, Nigeria

Email: 211333010@nileuniversity.edu.ng

1. INTRODUCTION

Blockchain technology, known for its decentralized and transparent nature has transformed the way transactions and agreements are conducted. It functions as a distributed ledger composed of interconnected blocks, cryptographically linked together with each active node on the network maintaining a local copy of all transactions [1] to ensure transparency and security. Originally developed to enhance financial transactions within the Bitcoin network [2], blockchain has gained widespread attention and is now adopted across various industries including supply chain management, record keeping, and identity management [3].

Central to many blockchain platforms are smart contracts, which are self-executing codes that automatically carry out tasks when specific predefined conditions are met [4], thereby eliminating the need for intermediaries. Unlike traditional agreements that rely on trusted third parties and arbitration, blockchain-based

smart contracts directly define the terms of agreements between parties. Nick Szabo introduced the concept of smart contracts in the early 1990s [5], aiming to enable transactions and agreements without middlemen, thereby reducing costs and enhancing transparency. By leveraging the immutability and decentralization of blockchain, smart contracts provide a trusted and transparent way to fulfill contractual obligations, allowing the automatic execution of agreed terms between untrusted parties.

Despite their potential, smart contracts have increasingly become targets for sophisticated cyber threats that can compromise the integrity, availability, and confidentiality of entire blockchain platforms. This vulnerability arises from the inherent complexity of smart contracts, their unique programming paradigm, and the immutable nature of blockchain transactions [6]. Exploiting vulnerabilities in smart contract code can lead to unexpected outcomes, tarnishing the reputation and undermining trust in the blockchain platform. A notable instance of such an exploit occurred in 2016 with the Decentralized Autonomous Organization (DAO), where attackers took advantage of a vulnerability in the DAO's contract, leading to the theft of 3.5 million Ethers, valued at approximately 45 million USD at the time [7].

Researchers have proposed various methodologies for detecting vulnerabilities in smart contracts utilizing symbolic and dynamic analysis as well as machine learning techniques. One of the earliest tools developed for this purpose is OYENTE [8], which utilizes static symbolic execution to reveal vulnerabilities in smart contracts. OYENTE was tested on 19,366 Ethereum smart contracts and successfully detected vulnerabilities in 8,833 of them, including the notorious DAO issue. This tool can identify vulnerabilities such as reentrancy, transaction ordering dependency, timestamp dependence, and mishandled exceptions.

Another tool, Mythril [9], utilizes symbolic execution techniques to evaluate potential vulnerabilities in smart contracts. When it detects undesirable conditions, it validates or dismisses them based on specific assumptions. Similarly, Slither [10], designed to detect reentrancy vulnerabilities, utilizes static analysis technique by converting the solidity abstract syntax tree (AST) into an internal representation language called SlithIR, which is then analyzed for vulnerabilities.

Securify [11], a dynamic analysis tool, detects flaws in Ethereum smart contracts by extracting semantic information through symbolic execution on the contract's dependency graph. It observes compliance and violation patterns to verify specific properties. Likewise, sFuzz [7] combines a lightweight, adaptive strategy with American fuzzy lop (AFL) fuzzing to detect up to nine types of smart contract vulnerabilities. Contract-Fuzzer [12] generates fuzzing inputs based on the smart contract's constructor arguments and its application binary interface (ABI) to detect a range of vulnerabilities including gasless sends, block number dependency, reentrancy, exception handling issues, frozen ether, and delegate call vulnerabilities.

Additionally, a machine learning-based vulnerability detection model [13] using K-nearest neighbors (KNN) has been proposed, effectively identifying vulnerabilities such as access control, arithmetic errors, bad randomness, denial of service, reentrancy, and unchecked low-level calls. Another approach [14], employing heterogeneous graph transformers (HGTs) and node-level attention has shown success in detecting vulnerabilities at both coarse and fine-grained levels. Furthermore, a model called the serial-parallel convolutional bidirectional gated recurrent network [15] was developed to preserve the spatial and temporal structure of smart contracts while extracting features from the input sequence to detect vulnerabilities such as reentrancy, timestamp dependency, and infinite loops.

Despite the growing attention to smart contract security, research on vulnerability identification, analysis, and mitigation remains fragmented. Many studies focus on isolated aspects of smart contract security, leaving gaps in understanding how the different elements interact. Additionally, there is a lack of comprehensive domain-based classification and a structured approach to categorizing evolving threats. This fragmented understanding hampers the ability of researchers, developers, and policymakers to effectively communicate, analyze, and address the multifaceted security challenges in the blockchain ecosystem.

This systematic review addresses these gaps by thoroughly examining existing literature and categorizing smart contract vulnerabilities into code-level, environment-dependent, and user-related domains. By organizing smart contract threats into coherent categories and analyzing the associated attack vectors and detection methods, the review offers valuable insights for researchers, developers, and the broader blockchain community. It offers a comprehensive framework for understanding smart contract threats and stakeholder roles, enabling the development of effective security strategies, informing regulations and standards that promote secure development practices and promoting trust in blockchain technology. It also serves as a foundation for further research into specific smart contract vulnerabilities and corresponding mitigation strategies.

2. METHOD

This study utilized a systematic review (SR) methodology to thoroughly examine the current threat landscape of smart contracts. The review was conducted using the preferred reporting items for systematic reviews and meta-analysis (PRISMA) guidelines [16], ensuring a transparent and replicable process. These guidelines provide a structured framework that helps maintain consistency, transparency, and replicability in conducting and reporting systematic reviews. By adhering to PRISMA, we focused on planning, executing, and presenting systematic reviews with enhanced quality, reliability, and comparability. The systematic review process in this study involved five key stages: planning, searching, filtering, selection, and eligibility assessment. This approach is based on empirical data and theoretical analysis, which guarantees the quality and comprehensiveness of the research.

2.1. Planning the review

This systematic review follows a structured methodology that thoroughly evaluates the existing research on the topic. The process begins by identifying the need for the review and formulating research questions, which serve as guiding criteria for selecting and assessing all reference articles included in the review. This study used four specific research questions to direct the systematic review process. A detailed description of these questions is provided in Table 1.

Table 1. Research questions and objectives

Research questions	Objectives
RQ1: What are the most common code-level vulnerabilities found in smart contracts?	To identify the primary code-level vulnerabilities in smart contracts offering a detailed analysis of their impact.
RQ2: What environment-dependent vulnerabilities can impact the security of smart contracts?	To identify environment-dependent vulnerabilities in smart contracts and assess their impact.
RQ3: What user-related vulnerabilities are linked to smart contracts?	To explore user-related threats in smart contracts, emphasizing how user interactions and behaviors can contribute to security breaches.
RQ4: What are the most effective techniques for detecting vulnerabilities in smart contracts?	To examine and assess the effectiveness of various techniques and approaches for detecting vulnerabilities in smart contracts, including their strengths and limitations.

2.2. Database and search strategy

We thoroughly searched several electronic scholarly databases, including IEEE Xplore, Science Direct, MDPI, Springer Link, ACM Digital Library, and Google Scholar. Additionally, we reviewed relevant journals in the field, such as the Indonesian Journal of Electrical Engineering and Computer Science (IJEECS) and the International Journal of Computers and Applications (IJCA). The search used a range of keywords, including “smart contracts,” “vulnerabilities,” “attacks,” “blockchain,” “security,” “threats,” “defense mechanisms,” “detection,” “mitigation strategies,” and “bugs,” as detailed in Figure 1. The search focused exclusively on peer-reviewed journal articles, conference proceedings, and book chapters published in English from January 2016 to June 2024. This process yielded a total of 342 articles, with contributions from IEEE Xplore (79), Science Direct (42), ACM Digital Library (67), Springer Link (31), MDPI (68), Google Scholar (50), IJEECS (3), and IJCA (2).

("smart contracts" and "vulnerabilities", "smart contracts" and "attacks", "blockchain" and "security", "smart contracts" and "security", "smart contracts" and "threats", "blockchain" and "vulnerabilities", "smart contracts" and "flaws", "smart contracts" and "bugs", "smart contracts" and "defense mechanism", "smart contracts" and "vulnerability detection")

Figure 1. Search strategy terms

2.3. Inclusion and exclusion criteria

We implemented a set of inclusion (IC) and exclusion (EC) criteria, outlined in Table 2, to ensure the relevance and quality of the selected studies. Articles that did not meet these criteria were excluded from the analysis as illustrated in Figure 2.

Table 2. Inclusion and exclusion criteria

Inclusion criteria	Exclusion criteria
Articles published from 2016 to 2024.	Duplicate literature.
Studies that satisfy at least one of the specified search criteria.	Studies that are not relevant to the topic.
Articles published in scholarly journals, conference proceedings, or academic periodicals.	Non-peer-reviewed articles, including blog posts and opinion pieces.
Studies offering empirical data or theoretical analysis.	Research papers not written in English.

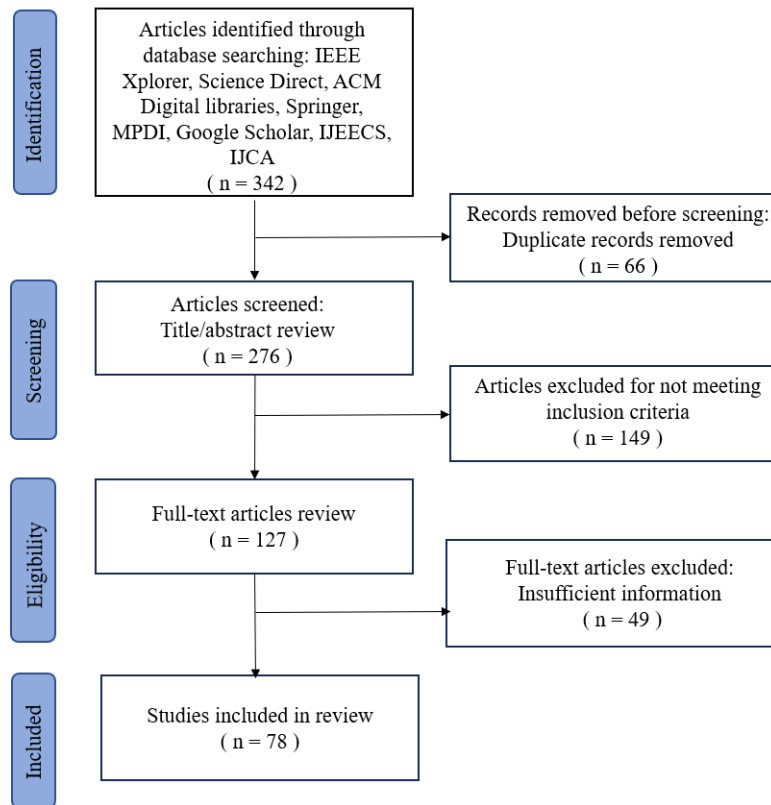


Figure 2. PRISMA flow diagram

2.4. Paper selection

The selection process involved three main stages: i) Removing duplicate articles and performing an initial screening of titles and abstracts to eliminate studies that did not meet the specified criteria and keywords. ii) Conducting a full-text review and thorough examination of the papers for final selection. iii) Assessing the quality of the remaining papers and discard those that did not meet the inclusion criteria. The initial search produced 342 articles. After eliminating duplicates and reviewing the titles and abstracts, 127 articles remained for further evaluation. An additional 49 articles were excluded due to insufficient information, resulting in 78 articles for the systematic review.

2.5. Eligibility

The selection results showed that 264 articles did not meet the inclusion criteria, leaving 78 articles for the systematic review. These selected studies were thoroughly examined, and data were extracted using a standardized form to ensure consistency. The extracted data included: (i) bibliographic details (author, title, publication year, journal/conference), (ii) types of vulnerabilities identified, (iii) associated attack vectors and exploitation techniques, (iv) detection methods and experimental setups, (v) proposed solutions and mitigation strategies, and (vi) key findings. This data was synthesized to create a comprehensive conceptual framework that categorizes smart contract vulnerabilities into three main domains: code vulnerabilities, execution environment threats, and user-related threats.

3. RESULTS AND DISCUSSION

This section consolidates the findings from the systematic review (SR) of 78 selected articles, providing a detailed analysis to address the research questions. The SR identified key vulnerabilities in smart contracts and their underlying causes, categorizing these vulnerabilities and related attacks into three domains: code-level, environment-dependent, and user-related. Additionally, the review examined the techniques used to detect these vulnerabilities.

The proposed classification of vulnerabilities into code-level, environment-dependent, and user-related domains extends the existing smart contract weakness classification registry (SWC). While the SWC offers a comprehensive taxonomy based on technical characteristics and root causes, our new classification scheme introduces a fresh perspective, emphasizing the diverse nature of the challenges and the various stakeholders involved, such as developers, platform providers, and users. This new approach provides valuable insights that can guide the development of targeted mitigation strategies and best practices tailored to each vulnerability category. It also aids in prioritizing and allocating resources more effectively, focusing on the most critical types of vulnerabilities. This SR identified thirteen (13) of the most common smart contract vulnerabilities, which are discussed in detail as we address the research questions. Table 3 lists these prominent vulnerabilities, categorized into the three domains based on the selected papers.

Table 3. Taxonomy of smart contract vulnerabilities

Domain	Vulnerability type	Known attacks
Code-level vulnerabilities	Reentrancy [6], [8], [12], [15], [17]-[40]	DAO attack [22], [37], [38]
	Integer overflow/underflow [5], [15], [17]-[19], [23], [31]-[34], [41]-[47]	BEC contract attack [45], [46] Poolz finance hack [47]
	Mishandled exception [8], [13], [17]-[18], [26], [31]-[32], [36], [40]-[41], [47]	King of the ether throne attack [8]
	Freezing ether [12], [15], [18], [31], [48]-[50]	2017 Parity multisig wallet attack [12], [15], [49], [50]
	Uninitialized storage variables [6], [23]	Parity multi-signature wallets [6]
	Inconsistent access control [40]-[41], [51]-[53]	data leakage [52]
	Self-destruct [52]	-
Environment-dependent vulnerabilities	Short address [13], [40], [51]	-
	Timestamp dependency [5], [8], [12],[15], [17], [19], [26], [28], [31]-[32],[38], [47], [54], [55]	-
	External oracles dependency [29], [55], [56], [57]	DeFi 'Flash Loan' attack [55]
User-related vulnerabilities	tx.origin [25], [26], [31], [32], [36], [52]	Social engineering and phishing attacks [26]
	Transaction ordering [5], [8], [17], [18], [23], [29], [47], [49], [55], [58], [59]	MEV crisis [55]
	Denial of service[13], [52], [60]	King Of Ether Throne threat [60]

3.1. RQ1: What are the most common code-level vulnerabilities found in smart contracts?

The literature review on code-level vulnerabilities in smart contracts reveals several significant issues resulting from programming errors, logical flaws, or improper use of language features. The review identified reentrancy, integer overflow/underflow, and mishandled exceptions as the most common code-level vulnerabilities. Developers can mitigate these inherent vulnerabilities by embracing secure coding practices, performing thorough testing, and following established security practices and guidelines. Implementing these strategies enhances smart contracts' security and reliability, reducing the likelihood of successful attacks.

3.1.1. Reentrancy

In this review, studies [17]-[40] have identified reentrancy vulnerabilities in smart contracts. These vulnerabilities arise when a malicious contract calls another contract that lacks sufficient security checks before the initial execution is finished. Similarly, researchers in [6], [8], [12], and [15] have also investigated reentrancy issues. Such vulnerabilities allow attackers to manipulate the contract's state. In these scenarios, the malicious contract may repeatedly call the original contract, resulting in unintended modifications to its state. By employing a recursive callback within the main function, an attacker can create an infinite loop to drain funds. A notable example of this vulnerability is the 2016 DAO attack [22], [37], [38], which led to the unauthorized withdrawal of over 50 million USD in Ether [19], [26], [39]. To mitigate this risk, it is advised

to use ‘send()’ or ‘transfer()’ for transfer operations and to update the internal state before executing low-level calls [32], [40].

3.1.2. Integer overflow/underflow

In smart contract code, integers are usually defined as fixed-size or non-signed integer types [5] thereby restricting the range of values that integer variables can hold. Integer overflow occurs when an integer variable exceeds its maximum allowable value, causing it to wrap around and restart from the minimum value. Conversely, integer underflow happens when an integer variable falls below its minimum allowable value, leading it to wrap around and assume the maximum value of the integer’s data type [41]. These vulnerabilities can result in incorrect transfers, loss of funds, and other security issues.

The review identified from studies [5], [15], [17]-[19], [23], [31]-[34], [41]-[47] that arithmetic operations producing unexpected values due to exceeding or falling below the defined integer range can lead to integer overflow/underflow vulnerabilities. A notable example is the Beauty Chain (BEC) contract attack [45], [46] in April 2018, where an attacker exploited an integer overflow vulnerability to duplicate tokens endlessly, leading to the instantaneous loss of over 900 million USD [17]. Additionally, Poolz Finance suffered a significant arithmetic overflow hack due to a flaw in its pool creation method, involving manual summation of token counts, resulting in a 6,650,000 USD loss [47]. Without a mechanism to detect integer overflow and underflow, attackers can potentially gain more tokens than they are entitled to. To address these risks, researchers recommend using safe math libraries or built-in functions [31], [32] which are designed to detect and manage overflow or underflow conditions and provide mechanisms to reverse transactions if such conditions are detected.

3.1.3. Mishandled exceptions

Studies [8], [13], [17], [18], [40], [41], [47] have identified mishandled exception vulnerabilities in smart contracts. These vulnerabilities occur when the contract fails to properly handle exceptions or errors during execution or from external function calls, potentially leading to unauthorized access to funds or other resources. Mishandled errors are also identified as unchecked calls in literature [26], [31], [32], [36]. Smart contracts can interact with one another through low-level functions such as delegatecall, send and call [17], which do not automatically trigger exceptions when errors occur. Instead, these functions return a boolean value indicating the success of the call. If the return values of these low-level calls are not adequately verified, it can lead to “fail-open” scenarios and other unintended consequences [13], [41]. Contracts with functions like send, call, callcode, and delegatecall are particularly vulnerable to unchecked low-level call issues [26]. Notable examples of attacks exploiting mishandled exception vulnerabilities include the Parity multi-signature wallet incident on Ethereum, which resulted in the loss of over 31,000,000 USD worth of Ether [40], and the King Of The Ether Throne attack [8]. To mitigate these vulnerabilities, it is recommended to use higher-level functions like transfer and send, which automatically revert on failure, rather than low-level functions like call and delegatecall. Additionally, contracts should verify the return values of instructions to ensure proper execution [17].

3.1.4. Freezing Ether

The freezing Ether vulnerability occurs when Ether becomes locked and inaccessible. Studies [12], [15], [18], [31], [49], [50] have identified that this issue arises because smart contract developers often fail to properly account for the Ether transfer function when designing their contracts. They frequently focus solely on the receive function, which allows the contract to accept ether deposits, without ensuring that the contract can also transfer it out. As a result, Ether received by the contract can become frozen and unusable. This issue is particularly problematic for contracts that depend on external contracts (via delegatecall) for Ether transfers. If these external contracts perform a suicide or selfdestruct operation, the calling contract loses its ability to send Ether, leading to the freezing of all its Ether. A notable example of this vulnerability is the 2017 Parity multisig wallet incident, where the self-destruction of the Parity wallet library contract resulted in over 280 million USD worth of Ether being frozen and inaccessible [12], [15], [49], [50]. This vulnerability is specific to Ethereum-based smart contracts. To mitigate this risk, it is advised that contracts designed to receive Ether should include mechanisms for Ether withdrawal or transfer, using functions such as transfer, send, or call.value() operations [31].

3.1.5. Uninitialized storage variables

Studies [6], [23] have highlighted that vulnerabilities related to uninitialized storage variables arise when storage pointers are left uninitialized, resulting in unintended overwriting of storage variables. A notable example is the Parity multi-signature wallet attack [6] in July 2017, which exploited a vulnerability associated with uninitialized function access control in the smart contract library used by Parity multi-signature wallets. This vulnerability allowed attackers to gain control of the wallet and steal over USD 150 million worth of Ether. To mitigate such risks, it is essential to explicitly set default values for variables. This practice ensures that variables have well-defined states before interacting with other variables, thereby preventing unexpected behaviors.

3.1.6. Inconsistent access control

This issue often arises from the oversight or inexperience of contract developers in creating and implementing access control mechanisms [41]. Improperly configured visibility settings in smart contracts can give attackers easy direct access to a contract's private values or internal logic [51]. If sensitive data or critical functions are not appropriately marked as private or internal, they become vulnerable to exploitation by malicious actors. Attackers might be able to directly interact with these exposed elements, allowing them to read private data or bypass intended access controls. This can lead to poor permission management, permitting malicious users to access or change sensitive information and functions. Such access control issues pose significant security risks, including loss of funds, contract tampering, and data leakage [52]. To prevent inconsistent access control vulnerabilities, developers should clearly specify the visibility of all contract functions [40]. Use the 'private' keyword for internal logic and data that should remain inaccessible outside the contract, 'internal' for functions and data that should only be accessible within the contract or its derived contracts and implement access control modifiers such as 'onlyOwner' or custom access control checks to restrict access to sensitive functions.

3.1.7. Self-destruct

The self-destruct vulnerability in smart contracts occurs when a contract includes a function that allows it to be terminated. As noted by Liao *et al.* [52], if a contract has a selfdestruct function, an attacker who gains control of the contract could call this function, leading to the contract's unexpected termination. This can result in the loss of funds or disruption of the contract's intended operations. Often, contracts include critical logic or perform calculations based on their balance within the fallback function. However, this logic can be bypassed using the self-destruct function, which lets a user specify a beneficiary for any remaining Ether. Consequently, a vulnerable contract can be exploited to transfer all funds to the attacker's account while shutting down the contract's operations.

3.1.8. Short address

The short address vulnerability occurs when a contract receives fewer bytes of data than expected. This issue arises because the Ethereum virtual machine (EVM) incorrectly accepts arguments with insufficient padding [51]. When this happens, the EVM automatically fills the missing bytes with zeros, starting from the highest byte of the subsequent parameter [13]. Since the deployed smart contract cannot prevent this automatic padding, it may interpret these zeros as valid input data. This vulnerability results from the EVM's failure to validate address lengths. To mitigate this issue, smart contract developers should implement checks within their contract code to verify the length of transaction input data [40].

3.2. RQ2: What environment-dependent vulnerabilities can impact the security of smart contracts?

Smart contracts as essential elements of blockchain-based systems, are inherently influenced by the complexities of the blockchain environment. Therefore, environment-dependent vulnerabilities stem from the interplay between smart contracts and the broader blockchain ecosystem and the distinctive features of the underlying distributed ledger technology [53], [54]. Addressing these environment-dependent vulnerabilities requires a thorough understanding of blockchain architecture, smart contract integration, and the development of robust mechanisms to manage external dependencies effectively, thereby improving the security and reliability of smart contract-based applications.

3.2.1. Timestamp dependency

Several studies [5], [8], [12], [15], [17], [19], [26], [28], [31], [32], [38], [47], [55] have highlighted vulnerabilities related to smart contracts that depend on block timestamps for entropy or for executing critical

operations. These vulnerabilities arise when smart contracts use precise block timestamps to make significant control flow decisions [5]. This dependence on timestamps renders contracts vulnerable to manipulation by attackers. In a decentralized blockchain network, miners can adjust block timestamps within a window of less than 900 seconds [28], [32], potentially exploiting arbitrage opportunities or gaining unfair advantages [38]. This issue also contributed to the miner extractable value (MEV) crisis [55]. To address this vulnerability, it is advisable to use block numbers instead of timestamps, as block numbers are resistant to manipulation by malicious miners [31].

3.2.2. External oracles dependency

Research, including studies [29], [55]-[57] have emphasized the risks associated with smart contracts relying on third-party data sources and oracles. Smart contracts frequently rely on oracles to supply essential external data, such as pricing information, meteorological data, or market trends, to execute blockchain transactions. However, ensuring accurate, consistent, and reliable data is more complex than it might seem. The design of the Oracle system and its integration with the smart contract can make it vulnerable to manipulation by adversaries who can exploit the data source for malicious purposes. A notable example of this vulnerability was the 2020 DeFi 'Flash Loan' attack, which resulted in the irreversible loss of approximately USD 100 million due to such exploits [55]. To mitigate this risk, researchers have suggested using time-weighted average prices when consulting price oracles on the Ethereum platform [29].

3.3. RQ3: What user-related vulnerabilities are linked to smart contracts?

One category of vulnerabilities in smart contracts arises from interactions between users and the contract itself. These user-related vulnerabilities often result from factors such as limited security awareness among users, insufficient user authentication mechanisms, poor key management practices, and the exploitation of human error. In 2022, over 1.9 billion USD was estimated to be lost due to breaches exploiting weaknesses in smart contract logic and user errors [47]. To address these vulnerabilities, a user-centric approach is essential including effective user education, creating secure user interfaces, and implementing robust key management practices.

3.3.1. Transaction ordering

Studies [5], [8], [17], [18], [23], [29], [47], [49], [55], [58], [59] have reported vulnerabilities related to transaction ordering, which depend on the sequence in which transactions are executed. In a blockchain network, the order of transaction execution is determined by miners [5]. Some contracts require transactions to follow a specific sequence. When multiple transactions are submitted simultaneously, the miner decides their execution order [49]. The user whose transaction is processed first may receive a reward, so the miner's chosen transaction order influences the final state of the contract. Malicious miners could exploit this by prioritizing their transactions or reordering the sequence [8], [17]. This issue contributed to the MEV crisis, where high gas fees paid by bots exploiting these opportunities create substantial security risks at the consensus layer [55]. To address this vulnerability, researchers suggest implementing a cryptographic commit-reveal scheme [59]. This method involves obfuscating sensitive information within the transactions, such as gas prices or transaction values. By concealing this information, it becomes more challenging for attackers to manipulate transaction ordering and exploit the vulnerability.

3.3.2. Denial of service

DoS attacks pose a significant risk to the stability and reliability of smart contracts arising from various causes. In a DoS attack, the attacker aims to inundate the smart contract with excessive requests or computationally demanding operations, effectively blocking the contract from serving legitimate users. This disruption can impair the normal functioning of the smart contract, potentially causing its failure [13]. For example, an attacker might cause certain operations to fail intentionally to execute a DoS attack [52]. If a function recursively transfers Ethers to a list of users and one of these transactions fails, the entire operation may be reverted. An attacker can exploit this by deliberately causing the failure to execute a denial-of-service attack. The King Of Ether Throne incident [60] illustrates such an exploitation of this vulnerability.

3.3.3. tx.origin

Vulnerabilities related to tx.origin have been highlighted in studies [25], [26], [31], [32], [36], [52]. The tx.origin variable in smart contracts represents the address of the original transaction initiator and is sometimes used to verify the legitimacy of a function caller for critical operations. However, this method is prob-

lematic because tx.origin reflects the original transaction creator's address even if another smart contract calls the function. This can mislead the contract into believing the call comes from a legitimate user, which may be from a malicious contract. Consequently, attackers can exploit this flaw to trick users into performing actions on compromised contracts [32], making them vulnerable to social engineering attacks [26]. For example, adversaries can employ phishing techniques using a malicious intermediary contract to deceive victims into executing critical functions. To address this vulnerability, 'tx.origin' should not be used for access control and authorization. Instead, use 'msg.sender' which accurately identifies the immediate caller of the function [31], [36].

3.4. RQ4: What are the most effective techniques for detecting vulnerabilities in smart contracts?

The security of smart contracts has emerged as a significant concern for developers and researcher, prompting the investigation of various techniques to identify and detect vulnerabilities within these systems. To address RQ4, a comprehensive review of empirical studies on smart contract vulnerability detection was performed, analyzing 78 selected articles. The detection techniques identified were grouped as traditional methods, machine learning-based methods and hybrid methods based on the technologies used. Table 4 outlines the different detection techniques discussed in the literature.

3.4.1. Traditional methods

Traditional vulnerability detection techniques include static analysis, dynamic analysis, taint analysis, fuzz testing, and formal verification. Our review identified several static analysis tools used for detecting smart contract vulnerabilities such as Oyente [8], Silther [10], SmartCheck [31], Vandal [36], MSmart [42], Madmax [44], MAIAN [61], Ethertrust [62], SmartScan [63], Ethainter [64], and AChecker [65]. Static symbolic execution tool identify potential bugs without executing the code. The first tool to use symbolic execution for smart contract vulnerability detection is Oyente [8]. Other tools like Mythril [9], DefectChecker [25], Manticore [66] and GasChecker [67] also utilize symbolic execution to detect contract vulnerabilities. Symbolic execution involves replacing program variables with symbols and performing symbolic computations during code traversal, generating path conditions for each execution path, which are crucial for verifying path feasibility and detecting potential issues [68]. Osiris [6] combines symbolic execution with taint analysis to identify integer bugs, while Sereum [24] uses taint analysis to uncover vulnerabilities.

Fuzzing techniques are employed by tools such as sFuzz [7], ContractFuzzer [12], ReGuard [37], CONFUZZIUS [69], and HFContractFuzzer [70]. These tools detect anomalies by inputting large volumes of randomly generated data into smart contracts and analyzing runtime behaviors to identify vulnerabilities. ContractFuzzer [12] was the first tool to apply fuzzing techniques to smart contracts, examining the ABI specification to generate inputs and identify defects. The only formal verification tool identified in our review is Securify [11]. Formal verification uses logical models and rigorous mathematical reasoning to verify the correctness and security of smart contracts [32].

3.4.2. Machine learning-based methods

Our SR examined various studies that employed traditional machine learning and advanced deep learning techniques for detecting smart contract vulnerabilities. These methods have demonstrated effectiveness in analyzing large datasets and uncovering patterns that traditional techniques may overlook. Several studies have successfully applied machine learning and deep learning techniques to identify smart contract vulnerabilities such as reentrancy, timestamp dependency, integer overflow/underflow and transaction ordering.

For example, studies [13], [17], [27], [39], [71], [72] have employed machine learning algorithms to detect vulnerabilities. Specifically, study [13] applied KNN to identify smart contract bugs. In study [17], researchers utilized five machine learning algorithms: eXtreme gradient boosting (XGBoost), KNN, random forest (RF), support vector machine (SVM), and adaptive boosting (AdaBoost) to detect smart contract vulnerabilities. Study [27] developed a RF classifier to monitor blockchain transaction balances and metadata for vulnerability identification. Study [39] used a neural network-based static analysis tool for detection of smart contract vulnerabilities. Study [71] utilized four machine learning classifiers: RF, SVM, decision tree (DT), and neural network (NN) to identify vulnerabilities in smart contracts, while study [72] employed unsupervised learning algorithms such as K-means, agglomerative clustering, HDBSCAN, Spectral, and OneClass SVM to analyze transaction behavior for vulnerability detection. Additionally, our review highlighted various studies that focused on deep learning techniques. For instance, study [4] introduced SmartEmbed for detecting contract code clones and bugs, while study [26] developed CodeNet, a CNN architecture for flaw detection in

smart contracts. Study [38] proposed the use of graph neural networks (GNN) for vulnerability identification, while study [49] presented ESCORT, a deep neural network framework for detecting flaws in smart contracts. Furthermore, studies [73] and [74] introduced long short-term memory (LSTM) networks for transaction-based vulnerability classification and detection. Study [75] proposed the use of multiple-objective detection neural network (MODNN) for smart contract vulnerability detection.

3.4.3. Hybrid methods

Our review revealed that hybrid methods, which integrate traditional techniques with machine learning or combine multiple machine learning and deep learning approaches, are designed to leverage the strengths of each method to enhance vulnerability detection. Various studies have implemented these hybrid models for detecting vulnerabilities in smart contracts. For instance, study [5] applied BiLSTM and other deep learning techniques for smart contract vulnerability detection, while study [14] incorporated multiple methods for smart contract vulnerability detection. Study [15] introduced the SPCBIG-EC for this purpose. Similarly, study [19] combined CNN with a self-attention mechanism. Study [28] explored the use of GNNs in conjunction with expert knowledge, and study [29] proposed GraphCodeBERT for smart contract bug detection. Study [32] employed a bidirectional gated recurrent unit network (BiGRU) enhanced by an attention mechanism for smart contract vulnerability detection. Study [41] also utilized BiLSTM and other deep learning techniques, while study [48] proposed a method that merges deep learning with multimodal decision fusion for detecting contract vulnerabilities. Further noteworthy approaches include study [51], which integrated machine learning with fuzz testing, and study [76], which utilized a combination of recurrent neural networks (RNN) and CNN for vulnerability detection. Study [77] introduced SCVDIE, a tool that employs neural networks and ensemble learning to identify vulnerabilities in smart contracts.

Additionally, study [78] merged Siamese networks with a deep learning model, and study [79] developed a tool using neural networks and slice matrices for detecting smart contract vulnerabilities. Finally, study [80] presented a hybrid model that combines various word embeddings (Word2Vec, FastText) with deep learning methods (LSTM, GRU, BiLSTM, CNN, BiGRU), while study [81] developed a reentrancy vulnerability detection model using BiGRU and SVM.

3.5. Evaluation of detection techniques

Our systematic review assessed the strengths and limitations of various smart contract vulnerability detection techniques. Although traditional methods can be effective, they have notable drawbacks. For example, symbolic execution can achieve high accuracy by generating control flow graphs (CFGs) from the target code [39]. However, this process is computationally intensive and time-consuming as it involves exploring all possible states the code may transition through [17], [26], [49], [73]. Additionally, both symbolic execution and fuzzing often incur significant execution overhead due to the need to run contracts symbolically. Formal verification methods face limitations due to the lack of specifications for built-in functions [22].

Static analysis techniques have improved in detecting vulnerabilities in smart contracts, yet they often exhibit relatively low accuracy rates [28]. These methods generally rely on expert-defined patterns which can result in high false-positive and low false-negative rates due to potential errors in pattern formulation [5], [32], [38]. As the number of smart contracts grows rapidly, it becomes increasingly impractical for experts to review all contracts and create accurate patterns [28].

Machine learning-based techniques have overcome some challenges of traditional methods by learning features directly from code, enabling versatile and efficient analysis. However, these methods mostly perform binary classification, categorizing contracts as either vulnerable or safe. They often struggle to identify multiple types of vulnerabilities, which limit their adaptability [49]. While deep learning methods are efficient and accurate with rapid detection rates [26], they are constrained by their dependence on a single representation of code. This approach can overlook important semantic and syntactic features, as it often treats code as text sequences and neglects critical variables in the data flow [28]. Additionally, this single-network architecture may lead to incomplete extraction of vital information. Furthermore, deep learning models are often viewed as "black boxes," which makes their predictions harder to interpret [73].

Hybrid methods present significant advantages by combining the strengths of various techniques, leading to improved performance in detecting smart contract vulnerabilities compared to the traditional or single deep learning models [15]. By integrating multiple algorithms, hybrid approaches effectively address the weaknesses of individual methods, resulting in more accurate and robust detection. However, they require considerable data, computational power, and resources for practical refinement and optimization.

Overall, our findings indicate that while traditional, machine learning-based, and deep learning methods each possess unique strengths and limitations, hybrid methods offer a promising path for more comprehensive and effective vulnerability detection in smart contracts. By combining multiple techniques, we can enhance accuracy and robustness, which is essential for addressing the increasing complexity and volume of smart contracts.

Table 4. Smart contract vulnerability detection approaches

Detection technique	Pros	Cons	Discussed tools
Traditional methods	Detect few known vulnerabilities.	Static analysis relies on expert-defined patterns which are prone to errors, low accuracy rates, poor scalability, high false positive, low false negative rates and low level of automation. Symbolic execution and fuzzing are typically time consuming and introduces execution overhead due to the necessity of running smart contracts symbolically.	Static analysis: Oyente [8], Silther [10], SmartCheck [31] Vandal [36], MSmart [42] Madmax [44], MAIAN [61], Ethertrust [62] SmartScan [63], Ethainter [64], Achecker [65]
		Formal verification methods are constrained by the lack of specifications for built-in functions.	Dynamic analysis: Mythril [9], DefectChecker [25] Manticore [66], GasChecker [67] Taint analysis: Osiris [6], Sereum [24] Formal verification: Securify [11] Fuzzing techniques: sFuzz [7], ContractFuzzer [12] ReGuard [37], CONFUZZIUS [69] HFCContractFuzzer [70]
ML/DL methods	Detects more vulnerabilities, versatile, time efficient analysis, scalable, high accuracy.	Does not consider the semantic information and context of the source code and poor interpretability.	Machine learning: Xu <i>et al.</i> [13], ContractWard [17] Dynamit [27], Eth2Vec [39] Momeni <i>et al.</i> [71], Agarwal <i>et al.</i> [72] Deep learning: SmartEmbed [4], CodeNet [26] DR-GCN [38], ESCORT [49] Hu <i>et al.</i> [73], AFS [74], MODNN [75]
Hybrid methods	Higher accuracy, extensible.	Requires substantial data, computational power, and resources for practical refinement and optimization	Shenyi [5], MANDO-HGT [14] SPCBIG-EC [15], Sun and Gu [19] Liu <i>et al.</i> [28], Peculiar [29], HAM [32] Wang <i>et al.</i> [41], Russell <i>et al.</i> [76] SCVDIE [77], SCVSN [78] BiGAS [81]

3.6. Interpretation of key findings

The review reveals the significant impact of code vulnerabilities on the security of smart contracts. Issues such as reentrancy bugs, integer overflow/underflow and other coding flaws were found to be common across various blockchain platforms, enabling attackers to exploit these vulnerabilities and potentially causing severe financial and reputational harm. The analysis also highlights the importance of considering the execution environment of smart contracts, noting that threats from the blockchain infrastructure, external system integration, and user-related factors can heavily influence overall security.

While previous research has focused on isolated aspects of smart contract threats or individual security challenges, this review in contrast, provides a more integrated and holistic framework that captures the complexity of security issues across multiple domains. For instance, study [82] examined vulnerabilities in smart contracts without proposing specific countermeasures. Study [83] focused on techniques and tools for testing smart contract vulnerabilities without exploring the vulnerabilities themselves. Researchers in [84] primarily addressed tools for analyzing vulnerabilities in Ethereum smart contracts. Study [85] conducted a survey on attacks and defenses on smart contracts but did not consider vulnerability detection tools. Additionally, study [86] reviewed types of smart contract vulnerabilities, attacks, defense mechanisms and only on a single detection tool. Furthermore, Durieux *et al.* [87] centered their analysis on evaluating and comparing automated vulnerability assessment tools.

The proposed classification scheme in this review enhances and refines existing taxonomies, offering a more robust framework for researchers, developers, and policymakers to better identify, communicate, and address emerging security challenges.

A major strength of this study is its thorough and systematic approach, which identified three primary domains of smart contract vulnerabilities. The new taxonomy presented offers a clear and structured method for understanding the varied nature of these challenges and the roles of different stakeholders, including developers, platform providers, and users. This new perspective provides valuable insights that can guide the development of targeted mitigation strategies and best practices tailored to specific vulnerability categories, enabling more effective prioritization and resource allocation.

However, the review has some limitations. Most of the studies examined were centered on the Ethereum blockchain, which, while prominent, may not fully represent the security challenges faced by other blockchain platforms. Additionally, while the review addresses a broad range of security threats, it does not sufficiently cover specific mitigation strategies or their effectiveness.

3.7. Implications and future research directions

The findings from this systematic review hold significant implications for developers, researchers, and policymakers working in the blockchain and smart contract fields. The proposed classification taxonomy provides a unified framework for communicating, analyzing and addressing the complex security challenges within the blockchain ecosystem.

A major implication is the need for more comprehensive and integrated smart contract security frameworks capable of tackling evolving threats across various domains. This includes developing robust and scalable detection methods suited to the complexity of real-world smart contracts. Furthermore, the review underscores the need to address user-related threats and human factors that impact blockchain application security, an area relatively underexplored in current literature.

Future research should expand on this study by incorporating a wider range of blockchain platforms, exploring specific mitigation strategies and best practices for identified smart contract vulnerabilities, and investigating emerging security challenges as blockchain technology advances. Collaborative efforts among academia, industry, and regulatory bodies will be essential for advancing the development of secure and reliable blockchain ecosystems, thereby realizing the full potential of this transformative technology.

4. CONCLUSION

The systematic review conducted, presents a comprehensive analysis of the ever-evolving threat landscape of smart contracts, drawing from 78 carefully selected articles to provide new insights into the nature and origins of smart contract vulnerabilities. Through the categorization of vulnerabilities into code-level, environment-dependent, and user-related domains, the SWC has been expanded, offering a more nuanced perspective that accurately represents the diverse challenges faced by developers, platform providers, and users. This review also includes an extensive evaluation of empirical studies on smart contract vulnerability detection techniques, which have been classified into three main groups: traditional, machine learning-based, and hybrid. Such classification, centered on the technologies employed, serves to underscore the strengths and limitations of each approach, thereby providing a roadmap for advancing detection capabilities in the future.

The proposed classification scheme not only enhances the understanding of smart contract vulnerabilities but also serves as a foundation for developing more effective mitigation strategies. By focusing on specific domains and the stakeholders involved, it enables better resource prioritization and more targeted efforts to secure smart contracts. The identification of thirteen common vulnerabilities, as detailed in this review, underscores the critical areas that demand attention to enhance smart contract security. When combined with the insights gained from the analysis of detection techniques, this review offers valuable guidance for future research and development activities, serving to inform the implementation of best practices, the refinement of detection methods, and the establishment of policies aimed at strengthening the security of blockchain ecosystems. As smart contract technology continues to advance, maintaining vigilance and proactivity in addressing emerging vulnerabilities is essential to ensure the integrity and reliability of decentralized applications.

REFERENCES

- [1] V. Bidve, A. Hamine, S. Akre, Y. Ghan, P. Sarasu, and G. Pakle, "Blockchain based drug supply chain for decentralized network," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 33, no. 1, pp. 485–495, Jan. 2024, doi: 10.11591/ijeecs.v33.i1.pp485-495.
- [2] R. A. Al-Kaabi and A. A. Abdullah, "A survey: medical health record data security based on interplanetary file system and blockchain technologies," *Indonesian Journal of Electrical Engineering and Computer Science (IJECS)*, vol. 30, no. 1, pp. 586–597, Apr. 2023, doi: 10.11591/ijeecs.v30.i1.pp586-597.
- [3] T. Abdellatif, K.-L. Brousmiche, and K.-L. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1-5, February 2018. [Online]. Available: <https://hal.science/hal-01760787>.
- [4] Z. Gao, "When deep learning meets smart contracts," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, Melbourne, Australia: ACM, Sep. 2020, pp. 1400–1402.
- [5] S. Qian, H. Ning, Y. He, and M. Chen, "Multi-label vulnerability detection of smart contracts based on Bi-LSTM and attention mechanism," *Electronics (Switzerland)*, vol. 11, no. 19, Oct. 2022, doi: 10.3390/electronics11193260.
- [6] C. F. Torres, J. Schütte, and R. State, "Osiris: hunting for integer bugs in ethereum smart contracts," in *ACM International Conference Proceeding Series, Association for Computing Machinery*, Dec. 2018, pp. 664–676. doi: 10.1145/3274694.3274737.
- [7] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "Sfuzz: an efficient adaptive fuzzer for solidity smart contracts," in *Proceedings - International Conference on Software Engineering, IEEE Computer Society*, Jun. 2020, pp. 778–788. doi: 10.1145/3377811.3380334.
- [8] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the ACM Conference on Computer and Communications Security, Association for Computing Machinery*, Oct. 2016, pp. 254–269. doi: 10.1145/2976749.2978309.
- [9] B. Mueller, "Mythril-Reversing and bug hunting framework for the Ethereum blockchain." 31 Dec 2021. [Online]. Available: <https://pypi.org/project/mythril/0.8.2>.
- [10] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *Proceedings - 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2019*, May 2019, pp. 8–15, doi: 10.1109/WETSEB.2019.00008.
- [11] P. Tsankov, A. Dan, D. D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: practical security analysis of smart contracts," in *Proceedings of ACM SIGSAC conference on computer and communications security*, October 2018, pp. 67-82. [Online]. Available: <http://arxiv.org/abs/1806.01143>.
- [12] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: fuzzing smart contracts for vulnerability detection," in *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Association for Computing Machinery, Inc, Sep. 2018, pp. 259–269. doi: 10.1145/3238147.3238177.
- [13] Y. Xu, G. Hu, L. You, and C. Cao, "A novel machine learning-based analysis model for smart contract vulnerability," *Security and Communication Networks*, vol. 2021, pp. 1–12, Aug. 2021, doi: 10.1155/2021/5798033.
- [14] H. H. Nguyen *et al.*, "MANDO-HGT: heterogeneous graph transformers for smart contract vulnerability detection," *IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia, 2023, pp. 334-346, doi: 10.1109/MSR59073.2023.00052.
- [15] L. Zhang *et al.*, "SPCBIG-EC: a robust serial hybrid model for smart contract vulnerability detection," *Sensors*, vol. 22, no. 12, Jun. 2022, doi: 10.3390/s22124621.
- [16] M. J. Page *et al.*, "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *The BMJ*, vol. 372, p. n71, Mar. 2021, doi: 10.1136/bmj.n71.1.
- [17] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: automated vulnerability detection models for ethereum smart contracts," *IEEE Trans Netw Sci Eng*, vol. 8, no. 2, pp. 1133–1144, Apr. 2021, doi: 10.1109/TNSE.2020.2968505.
- [18] D. Perez and B. Livshits, "Smart contract vulnerabilities: vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 1325-1341. 2021. [Online]. Available: www.usenix.org/conference/usenixsecurity21/presentation/perez.
- [19] Y. Sun and L. Gu, "Attention-based Machine learning model for smart contract vulnerability detection," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Mar. 2021, doi: 10.1088/1742-6596/1820/1/012004.
- [20] T. Sharma, Z. Zhou, A. Miller, and Y. Wang, "Exploring security practices of smart contract developers," *Arxiv*, Apr. 2022, [Online]. Available: <http://arxiv.org/abs/2204.11193>.
- [21] C. F. Torres, M. Baden, R. Norvill, B. B. F. Pontiveros, H. Jonker, and S. Mauw, "ÆGIS: shielding vulnerable smart contracts against attacks," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pp. 584-597, Oct. 2020, [Online]. Available: <http://arxiv.org/abs/2003.05987>.
- [22] Y. Zhang and D. Liu, "Toward vulnerability detection for Ethereum smart contracts using graph-matching network," *Future Internet*, vol. 14, no. 11, Nov. 2022, doi: 10.3390/fi14110326.
- [23] P. Zhang, F. Xiao, and X. Luo, "A framework and dataset for bugs in ethereum smart contracts," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 139-150. Sep. 2020, [Online]. Available: <http://arxiv.org/abs/2009.02066>.
- [24] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: protecting existing smart contracts against re-entrancy attacks," *Arxiv*, Dec. 2018, [Online]. Available: <http://arxiv.org/abs/1812.05934>.
- [25] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "DefectChecker: Automated Smart Contract Defect Detection by Analyzing EVM Bytecode," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2189–2207, Jul. 2022, doi: 10.1109/TSE.2021.3054928.
- [26] S. J. Hwang, S. H. Choi, J. Shin, and Y. H. Choi, "CodeNet: code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access*, vol. 10, pp. 32595–32607, 2022, doi: 10.1109/ACCESS.2022.3162065.
- [27] M. Eshghie, C. Artho, and D. Gurov, "Dynamic vulnerability detection on smart contracts using machine learning," in *Evaluation and assessment in software engineering*, 2021, pp. 305-312.
- [28] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Trans Knowl Data Eng*, vol. 35, no. 2, pp. 1296–1310, 2023, doi: 10.1109/TKDE.2021.3095196.




- [29] H. Wu *et al.*, “Peculiar: smart contract vulnerability detection based on crucial data flow graph and pre-training techniques,” In *IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, Wuhan, China, 2021, pp. 378-389, doi: 10.1109/IS-SRE52982.2021.00047.
- [30] A. Alkhalifah, A. Ng, P. A. Watters, and A. S. M. Kayes, “A mechanism to detect and prevent ethereum blockchain smart contract reentrancy attacks,” *Front Comput Sci*, vol. 3, Feb. 2021, doi: 10.3389/fcomp.2021.598780.
- [31] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “SmartCheck: Static analysis of ethereum smart contracts,” in *Proceedings - International Conference on Software Engineering*, IEEE Computer Society, May 2018, pp. 9–16. doi: 10.1145/3194113.3194115.
- [32] H. Wu, H. Dong, Y. He, and Q. Duan, “Smart contract vulnerability detection based on hybrid attention mechanism model,” *Applied Sciences (Switzerland)*, vol. 13, no. 2, Jan. 2023, doi: 10.3390/app13020770.
- [33] H. Wang, Y. Li, S. -W. Lin, L. Ma and Y. Liu, “VULTRON: catching vulnerable smart contracts once and for all,” *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, Montreal, QC, Canada, 2019, pp. 1-4, doi: 10.1109/ICSE-NIER.2019.00009.
- [34] T. D. Nguyen, L. H. Pham and J. Sun, “SGUARD: towards fixing vulnerable smart contracts automatically,” *2021 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2021, pp. 1215-1229, doi: 10.1109/SP40001.2021.00057.
- [35] S. Grossman *et al.*, “Online detection of effectively callback free objects with applications to smart contracts,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, Jan. 2018, doi: 10.1145/3158136.
- [36] L. Brent *et al.*, “Vandal: a scalable security analysis framework for smart contracts,” *arxiv*, Sep. 2018, [Online]. Available: <http://arxiv.org/abs/1809.03981>.
- [37] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, “ReGuard: finding reentrancy bugs in smart contracts,” in *Proceedings - International Conference on Software Engineering, IEEE Computer Society*, May 2018, pp. 65–68. doi: 10.1145/3183440.3183495.
- [38] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, “Smart contract vulnerability detection using graph neural networks,” in *Proceedings of the Twenty-Ninth International Conference on Artificial Intelligence*, 2021, pp. 3283-3290. <https://dl.acm.org/doi/abs/10.5555/3491440.3491894>.
- [39] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, “Eth2Vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts,” in *BSCI 2021 - Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, co-located with ASIA CCS 2021, Association for Computing Machinery, Inc, May 2021, pp. 47–59. doi: 10.1145/3457337.3457841.
- [40] H. Chen, M. Pendleton, L. Njilla, and S. Xu, “67 a survey on ethereum systems security: vulnerabilities, attacks, and defenses,” *ACM Computing Surveys*, vol. 53, 2020, doi: 10.1145/3391195.
- [41] M. Wang, Z. Xie, X. Wen, J. Li, and K. Zhou, “Ethereum smart contract vulnerability detection model based on triplet Loss and BiLSTM,” *Electronics (Switzerland)*, vol. 12, no. 10, May 2023, doi: 10.3390/electronics12102327.
- [42] J. Fei, X. Chen, and X. Zhao, “MSmart: smart contract vulnerability analysis and improved strategies based on smartcheck,” *Applied Sciences (Switzerland)*, vol. 13, no. 3, Feb. 2023, doi: 10.3390/app13031733.
- [43] E. Lai and W. Luo, “Static analysis of integer overflow of smart contracts in ethereum,” in *ACM International Conference Proceeding Series, Association for Computing Machinery*, Jan. 2020, pp. 110–115. doi: 10.1145/3377644.3377650.
- [44] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, “MadMax: surviving out-of-gas conditions in ethereum smart contracts,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, Nov. 2018, doi: 10.1145/3276486.
- [45] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, “EASYFLOW: keep ethereum away from overflow,” in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019, Institute of Electrical and Electronics Engineers Inc.*, May 2019, pp. 23–26. doi: 10.1109/ICSE-Companion.2019.00029.
- [46] T. Sun and W. Yu, “A formal verification framework for security issues of blockchain smart contracts,” *Electronics (Switzerland)*, vol. 9, no. 2, Feb. 2020, doi: 10.3390/electronics9020255.
- [47] L. S. H. Colin, P. M. Mohan, J. Pan, and P. L. K. Keong, “An integrated smart contract vulnerability detection tool using multi-layer perceptron on real-time solidity smart contracts,” *IEEE Access*, vol. 12, pp. 23549–23567, 2024, doi: 10.1109/ACCESS.2024.3364351.
- [48] W. Deng, H. Wei, T. Huang, C. Cao, Y. Peng, and X. Hu, “Smart contract vulnerability detection based on deep learning and multimodal decision fusion,” *Sensors*, vol. 23, no. 16, Aug. 2023, doi: 10.3390/s23167246.
- [49] C. Sendner *et al.*, “Smarter contracts: detecting vulnerabilities in smart contracts with deep transfer learning,” in *30th Annual Network and Distributed System Security Symposium, NDSS 2023*, The Internet Society, 2023. doi: 10.14722/ndss.2023.23263.
- [50] P. Qian, Z. Liu, Q. He, R. Zimmermann, and X. Wang, “Towards automated reentrancy detection for smart contracts based on sequential models,” *IEEE Access*, vol. 8, pp. 19685–19695, 2020, doi: 10.1109/ACCESS.2020.2969429.
- [51] J. Ye, M. Ma, Y. Lin, L. Ma, Y. Xue, and J. Zhao, “VULPEDIA: detecting vulnerable ethereum smart contracts via abstracted vulnerability signatures,” *Journal of Systems and Software*, vol. 192, p. 111410, Oct. 2022, doi: 10.1016/j.jss.2022.111410.
- [52] J.W. Liao, T.T. Tsai, C.-K. He, and C.-W. Tien, “SoliAudit: smart contract vulnerability assessment based on machine learning and fuzz testing,” in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, 2019, pp. 458–465. doi: 10.1109/IOTSMS48152.2019.8939256.
- [53] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10204 LNCS, 2017, pp. 164–186.
- [54] M. Bartoletti and L. Pompianu, “An empirical analysis of smart contracts: platforms, applications, and design patterns,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10323 LNCS, 2017, pp. 494–509.
- [55] K. Tjiam, R. Wang, H. Chen, and K. Liang, “Your smart contracts are not secure: investigating arbitrageurs and oracle manipulators in ethereum,” in *CYSARM 2021 - Proceedings of the 3rd Workshop on Cyber-Security Arms Race, co-located with CCS 2021, Association for Computing Machinery, Inc*, Nov. 2021, pp. 25–35. doi: 10.1145/3474374.3486916.

- [56] M. C. Argañaraz, M. M. Berón, M. J. V. Pereira, and P. R. Henriques, "Detection of vulnerabilities in smart contracts specifications in ethereum platforms," in *OpenAccess Series in Informatics, Schloss Dagstuhl- Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing*, Sep. 2020. doi: 10.4230/OASICS.SLATE.2020.2.
- [57] M. Wöhrer and U. Zdun, "Design patterns for smart contracts in the ethereum ecosystem," *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 1513-1520, doi: 10.1109/Cybermatics-2018.2018.00255.
- [58] S. Sayeed, H. Marco-Gisbert, and T. Cairra, "Smart contract: attacks and protections," *IEEE Access*, vol. 8, pp. 24416–24427, 2020, doi: 10.1109/ACCESS.2020.2970495.
- [59] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: lessons and insights from a cryptocurrency lab," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9604 LNCS, 2016, pp. 79–94.
- [60] Z. A. Khan and A. S. Namin, "A survey on vulnerabilities of Ethereum smart contracts," *Arxiv*, Dec. 2020, [Online]. Available: <http://arxiv.org/abs/2012.14481>.
- [61] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *ACM International Conference Proceeding Series, Association for Computing Machinery*, Dec. 2018, pp. 653–663. doi: 10.1145/3274694.3274743.
- [62] I. Grishchenko, M. Maffei, and C. Schneidewind, "EtherTrust: sound static analysis of ethereum bytecode." *Technische Universität Wien*, Tech. Rep, 2018, pp.1-41. [Online]. Available: <https://www.netidee.at/sites/default/files/2018-07/staticanalysis.pdf>.
- [63] N. F. Samreen and M. H. Alalfi, "SmartScan: an approach to detect denial of service vulnerability in ethereum smart contracts," in *IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 17-26, May 2021, [Online]. Available: <http://arxiv.org/abs/2105.02852>.
- [64] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: a smart contract security analyzer for composite vulnerabilities," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Association for Computing Machinery, Jun. 2020, pp. 454–469. doi: 10.1145/3385412.3385990.
- [65] A. Ghaleb, J. Rubin, and K. Pattabiraman, "AChecker: statically detecting smart contract access control vulnerabilities," in *Proceedings of the 45th International Conference on Software Engineering*, May 2023, pp. 945–956, doi.org/10.1109/ICSE48619.2023.00087.
- [66] M. Mossberg *et al.*, "Manticore: a user-friendly symbolic execution framework for binaries and smart contracts," *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, 2019, pp. 1186-1189, doi: 10.1109/ASE.2019.00133.
- [67] T. Chen *et al.*, "GasChecker: scalable analysis for discovering gas-inefficient smart contracts," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1433–1448, 2021, doi: 10.1109/TETC.2020.2979019.
- [68] P. Li *et al.*, "A vulnerability detection framework for hyperledger fabric smart contracts based on dynamic and static analysis," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, 2022, pp. 366-374. 2022. doi:10.1145/3530019.3531342.
- [69] C. F. Torres, A. K. Iannillo, A. Gervais and R. State, "ConFuzzius: a data dependency-aware hybrid fuzzer for smart contracts," *2021 IEEE European Symposium on Security and Privacy (EuroS-P)*, Vienna, Austria, 2021, pp. 103-119, doi: 10.1109/EuroSP51992.2021.00018.
- [70] M. Ding, P. Li, S. Li, and H. Zhang, "HFContractFuzzer: fuzzing hyperledger fabric smart contracts for vulnerability detection," in *ACM International Conference Proceeding Series, Association for Computing Machinery, Jun. 2021*, pp. 321–328. doi: 10.1145/3463274.3463351.
- [71] P. Momeni, Y. Wang, and R. Samavi, "Machine learning model for smart contracts security analysis," in *2019 17th International Conference on Privacy, Security and Trust (PST)*, 2019, pp. 1–6. doi: 10.1109/PST47121.2019.8949045.
- [72] R. Agarwal, T. Thapliyal, and S. Kumar Shukla, "Vulnerability and transaction behavior based detection of malicious smart contracts," in *Cyberspace Safety and Security: 13th International Symposium, CSS 2021, Virtual Event, Proceedings 13*, November 9–11, 2021, pp. 79-96, Springer International Publishing. [Online]. Available: <http://arxiv.org/abs/2106.13422>.
- [73] T. Hu *et al.*, "Transaction-based classification and detection approach for Ethereum smart contract," *Information Processing and Management*, vol. 58, no. 2, Mar. 2021, doi: 10.1016/j.ipm.2020.102462.
- [74] B. Wang, H. Chu, P. Zhang, and H. Dong, "Smart contract vulnerability detection using code representation fusion," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, IEEE Computer Society*, 2021, pp. 564–565. doi: 10.1109/APSEC53868.2021.00069.
- [75] L. Zhang, J. Wang, W. Wang, Z. Jin, Y. Su, and H. Chen, "Smart contract vulnerability detection combined with multi-objective detection," *Computer Networks*, vol. 217, p. 109289, Nov. 2022, doi: 10.1016/J.COMNET.2022.109289.
- [76] R. L. Russell *et al.*, "Automated vulnerability detection in source code using deep representation learning," *Arxiv*, Jul. 2018, [Online]. Available: <http://arxiv.org/abs/1807.04320>.
- [77] L. Zhang *et al.*, "A novel smart contract vulnerability detection method based on information graph and ensemble learning," *Sensors*, vol. 22, no. 9, May 2022, doi: 10.3390/s22093581.
- [78] R. Guo, W. Chen, L. Zhang, G. Wang, and H. Chen, "Smart contract vulnerability detection model based on siamese network(SCVSN): a case study of reentrancy vulnerability," *Energies (Basel)*, vol. 15, no. 24, Dec. 2022, doi: 10.3390/en15249642.
- [79] C. Xing, Z. Chen, L. Chen, X. Guo, Z. Zheng, and J. Li, "A new scheme of vulnerability analysis in smart contract with machine learning," *Wireless Networks*, 2020, doi: 10.1007/s11276-020-02379-z.
- [80] L. Zhang *et al.*, "CBGRU: a detection method of smart contract vulnerability based on a hybrid model," *Sensors*, vol. 22, no. 9, May 2022, doi: 10.3390/s22093577.
- [81] L. Zhang *et al.*, "A novel smart contract reentrancy vulnerability detection model based on BiGAS," *Journal of Signal Processing Systems*, vol. 96, no. 3, pp. 215–237, Mar. 2024, doi: 10.1007/s11265-023-01859-7.
- [82] S. Vani, M. Doshi, A. Nanavati, and A. Kundu, "Vulnerability analysis of smart contracts," *Arxiv*, Dec. 2022, [Online]. Available: <http://arxiv.org/abs/2212.07387>.




- [83] R. Sujeetha and C. A. S. D. Preetha, "A literature survey on smart contract testing and analysis for smart contract based blockchain application development," in *Proceedings - 2nd International Conference on Smart Electronics and Communication, ICOSEC 2021, Institute of Electrical and Electronics Engineers Inc.*, 2021, pp. 378–385. doi: 10.1109/ICOSEC51865.2021.9591750.
- [84] M. Di Angelo and G. Salzer, "A survey of tools for analyzing ethereum smart contracts," in *Proceedings - 2019 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPCON 2019, Institute of Electrical and Electronics Engineers Inc.*, Apr. 2019, pp. 69–78. doi: 10.1109/DAPPCON.2019.00018.
- [85] M. Saad *et al.*, "Exploring the attack surface of blockchain: a systematic overview," *Arxiv*, Apr. 2019, [Online]. Available: <http://arxiv.org/abs/1904.03487>.
- [86] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, vol. 107, pp. 841–853, Jun. 2020, doi: 10.1016/J.FUTURE.2017.08.020.
- [87] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings - International Conference on Software Engineering*, Jun. 2020, pp. 530–541, doi: 10.1145/3377811.3380364.

BIOGRAPHIES OF AUTHORS






Unyime Ufok Ibekwe    is an Information and Cyber Security professional with over sixteen (16) years of experience leading and implementing Information Security Strategies, Policies and Standards. She is currently serving as a key member of the Nigeria Financial Industry Computer Emergency Respond Team (NFICERT). Her expertise primarily revolves around Information security, Cyber Threat intelligence, Security Incident Management and Response, IT Risk Management, Vulnerability Management, Security Operation Center (SOC) management, Digital Forensic, Artificial Intelligence, Machine learning, Deep learning, Blockchain and Smart Contract security. She holds a A Bachelor of Science (B.Sc) degree in Computer Science, a Master's Degree (M.Sc) in Information Technology and a Master of Business Administration (MBA) degree. Presently, she is actively engaged as a researcher at Nile University of Nigeria. She can be contacted at email: 211333010@nileuniversity.edu.ng.






Uche M. Mbanaso    is a Cybersecurity expert and teaches at Nasarawa State University, Nigeria. He's also a visiting academic at the University of Witwatersrand, South Africa, where he teaches Cybersecurity and Digital Governance. He is a prominent figure in the development of Cybersecurity standards and policies. He has published over 30 research journal articles and 30 conference proceedings on the subject. He is the lead author of *Research Techniques for Computer Science, Information Systems and Cybersecurity* – the first African-authored book on research published by Springer Nature Switzerland AG. He holds an undergraduate degree in Electronics and Communications Engineering from Nigeria, an M.Sc in Information Technology, and a Ph.D. in Communications and Information Security, both from the UK. He can be contacted at email: u.m.mbanaso@nsuk.edu.ng.



Nwojo Agwu Nnanna    is a Professor and Head of Department of the Computer Science Department of Nile University of Nigeria. Dr. Nnanna received his Ph.D. degree from Texas Tech University, Lubbock Texas, United States of America in 1996. He has taught Computer Science and Mathematics at both undergraduate and graduate levels in the United States and Nigeria. His research interests are in the areas of artificial intelligence, design of robust optimization models for machine learning algorithms, robotics, dynamical systems, numerical computation, and systems and control theory. He has written several peer-reviewed journal articles in numerical computation and systems and control theory and application of semi-Lagrangian integration techniques to Navier-Stokes equations. He can be contacted at email: nagwu@nileuniversity.edu.ng.



Umar Adam Ibrahim    holds a B.Sc. in Computer Engineering, an M.Sc and a Ph.D. in Computer Science. His research focuses on natural language processing, particularly speech recognition and its application to local languages. As a certified AI practitioner, startup advisor, and mentor to Science, Technology, Engineering and Mathematics (STEM) students. He is deeply committed to advancing technology and innovation. He is a passionate advocate of the Sustainable Development Goals (SDGs) 4, 9, and 13, demonstrating leadership qualities that have earned him several awards. He can be contacted at email: umaradamibrahim@nileuniversity.edu.ng.