International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Survey on Smart Contract Vulnerabilities and Safeguards in Blockchain

Mrs. Rohini Pise[1], Dr. Sonali Patil[2]

Submitted: 05/09/2022 **Accepted:** 24/12/2022

Abstract: Blockchain technology is developing rapidly as a result of its numerous applications, security features and smart contracts embedded in it. Smart contracts are software codes written in a programming language. They get automatically executed on the Blockchain network when certain condition is met written in that program code. The distinctive characteristics of smart contracts led Blockchain technology to be used in applications beyond cryptocurrencies, including healthcare, IoT, supply chain, digital identification, digital asset exchange, crowdfunding, intellectual property, and many more. Millions were stolen and lost as a result of technical flaws and various vulnerabilities present in smart contracts. Many tools and methodologies have been proposed to address these challenges, and additional research is underway to build unique tools that enable the discovery of vulnerabilities in smart contract code.

Ethereum is a well-known public Blockchain platform supporting smart contracts. Additionally, Hyperledger Fabric is private Blockchain platform featuring smart contracts in private sector. This survey presents, a bird's eye view of smart contract languages, vulnerabilities and security tools in Public and Private Blockchain. The paper also looks at the different formal verification approaches used to identify the vulnerabilities present in the smart contract.

The intent of the paper is to focus on smart contract challenges and vulnerabilities, Security tools in Public and private Blockchain and Formal verification Methods for validation of smart contracts.

Keywords: Smart contracts, Blockchain, security, Ethereum, Hyperledger fabric, Formal Verification

1. INTRODUCTION

Blockchains are incredibly popular nowadays and are a relatively new technology with the primary purpose of achieving security. Blockchain stores information in blocks. Initially, this concept and the technology were described by some researchers with a major intention of timestamping the digital documents to avoid tampering with them. After that, Satoshi Nakamoto took forward this technology in 2009 to create the digital cryptocurrency Bitcoin[1]. A Blockchain is a digital ledger of transactions, decentralized, distributed database that duplicates and distributes transactions across the entire network. It is highly transparent and open to anyone. Satoshi Nakamoto [1]has introduced this technology to provide a platform for exchanging cryptocurrencies by eliminating third parties.

[1]*Pimpri Chinchwad College of Engineering, Pune, Maharashtra, India*
Email: rohini.pise@pccoepune.org
[2]*Pimpri Chinchwad College of Engineering, Pune, Maharashtra, India*
Email: sonali.patil@pccoepune.org

1.1 Structure and Working of Blockchain

To study the structure of Blockchain[2], first, let us understand the structure of a block. Every block contains information for instance data, the hash of the previous block and hash of block itself. The type of Blockchain decides which data is to be stored inside that block. In particular, Bitcoin Blockchain[1] stores the details about a transaction, for example a sender, receiver, and the amount, along with some other information. As discussed, earlier block also has a hash that identifies a block and its contents. The hash of every block is always unique, just as the fingerprint. When any transaction is successful, a block is created along with its hash value. The third element inside each block is the hash value of the previous block. This effectively creates and maintains a chain of blocks known as Blockchain.

The figure shows the structure of Blockchain[2]. As discussed, data blocks in Blockchain are chronologically linked to one another. The first block in every Blockchain is special, as being first, it cannot point to previous blocks. This block is known as genesis block. Header block contains the information about which

block will be appended next. This block sequence is retained in the Markle root block, and all transaction information is kept safe in this block[3]. As blocks are connected to each other, making changes in a block will make all following blocks invalid. To generate the hash of a block, hashing algorithms are used, such as SHA-256. Another benefit of blockchain technology is that data replication is not possible. This minimizes the number of data blocks that aren't needed or duplicated.

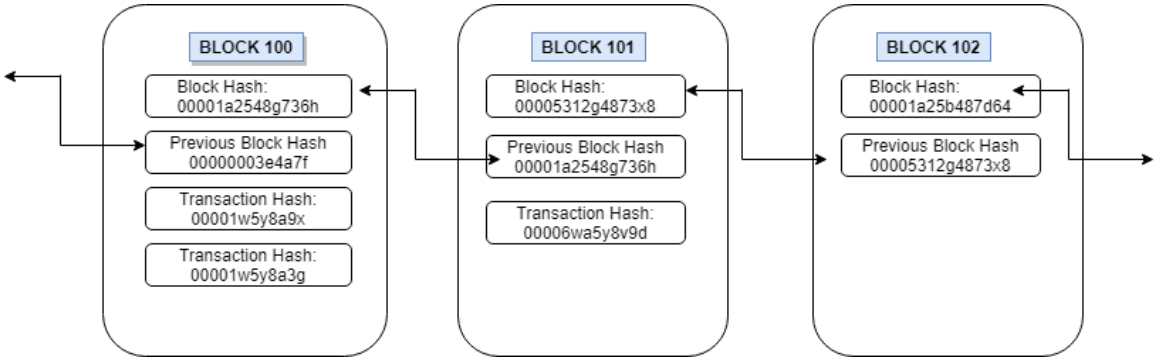

Figure 1. Structure of Blockchain

1.2 Blockchain Features:

Blockchain technology comes with many incredible features[4]. One of them is it provides transparency and privacy to the transactions. Transactions stored in blocks are transparent; that is, every user is aware of the transactions. Immutability is another feature that means the data cannot be changed. Greater transparency enhances security, and easier traceability in Peer-to-Peer networks makes secure Transactions[5].

Along with hash, it also has another feature known as Proof-of-Work which contributes in improving the security of Blockchain. Proof-of-work is an oldest consensus algorithm which validates the transactions. When PoW is used, it is resistant to tamper with the blocks because if one block gets tampered, then it is required to recalculate the proof-of-work for all the following blocks, and it is very time-consuming.

Another feature to make Blockchain more secure is its distributed nature[6]. In Blockchain there is no central entity responsible for managing the chain, instead it uses peer-to-peer network where each and every one in Blockchain manages the data.

1.3 Evolution in Blockchain

Blockchain has many unique and considerable benefits. The tremendous advancement in Blockchain technology has addressed the numerous research gaps and opportunities for researchers. Initially Blockchain was introduced as Bitcoin Cryptocurrency[1] by Satoshi Nakamoto. With the tremendous features of Bitcoin, it gained a massive popularity. But the block size and scalability are the limitations in Bitcoin Blockchain[7]. So, the next advancement in Blockchain is use of Smart contracts. Smart contracts were developed in public and private Blockchain platforms as they are secure, transparent and immutable. The challenges for smart contracts are writing smart contracts in programming languages and various vulnerabilities present in it. Then as a step forward, Blockchain is used now in Decentralized Applications (DApp). With rapid need of DApp in industry and society, there is need of integration towards business needs and demands of Industry 4.0[7]. This evolution is represented in given below figure 2.

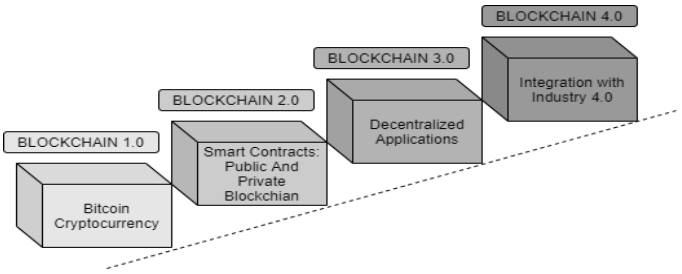

Figure 2. Evolution in Blockchain

1.4 Applications of Blockchain beyond cryptocurrency:

Cryptocurrency is the first real-world application of blockchain technology. But it is only the beginning. Apart from cryptocurrency[8], the Blockchain can be used for various applications simply as a digital record. A critical review is performed to study Blockchain applications[9] . Here are a few more incredible applications of blockchain technology, such as tracking the Real Estate, Finance[10][11], Property transfers[8], [12], Digital Asset exchange[13], use of smart contracts for IoT applications [14], and also IoT security[15]. Medical data sharing[16], [17], and Healthcare applications [18] are gaining more attention. Digital Identity and access management [19] is another important application area where Blockchain is used very efficiently. Retail Loyalty Programs, Money Transfers across Borders, and Crowdfunding[13] are some booming application areas. Supply Chain Management [20], Food Safety, Digital Voting[21], or e-

Voting[22] are the applications where Blockchain technology is being used, and it is proving to be more secure and efficient.

Figure 3. Application Domains of Blockchain

1.5 Blockchain Types:

Public and private Blockchains are the two most popular types[5]. Other variants based on hybridization, such as Hybrid Blockchain and the federated notion, Hyperledger, are also proposed based on the needs of the applications.

1. 5.1 Public Blockchain

As the name suggests, it is a Non-restrictive, permissionless distributed ledger system[23]. Any person with the internet becomes an authorized node by signing in to the Blockchain platform and can input transactions and be involved in the consensus process.

Public Blockchain Use Case: Bitcoin, Ethereum, Stellar, and Dash

1.5.2 Private Blockchain

It is a restrictive and permissioned Blockchain[23]. In this case, a central in-charge selects who gets to mine and who doesn't and also has the rights in the consensus process or decision-making. So, the users who are permitted by central authority can join the Blockchain and submit and read the transactions or participate in a consensus mechanism.

Private Blockchain Use Case: Hyperledger Fabric, Hyperledger Sawtooth, Corda, and Multichain.

1.5.3 Consortium Blockchain

It is a combination of both public and private Blockchains, i.e., semi-decentralized. As the access to all the users is restricted, only permissioned users have the right to add transactions and involve in the consensus process.

Use case: Quorum, Corda, and Hyperledger, J.P. Morgan Coin

1.5.4 Hybrid Blockchain

It combines features of a public and private Blockchain.
Use case: Dragonchain

1.6 Challenges in Blockchain:

Though Blockchain has tremendous applications and demand, it has specific challenges[24]. Few of the challenges are related to performance and security with Blockchain. When the blocks are processed faster, more forks are to be maintained, leading to security issues. The next challenge is with Block size. If we consider an example of Bitcoin, it has block size as 1 MB means very few transactions per second and so limitation on a greater number of transactions. 'Selfish mining attacks' is one of the significant challenges for Blockchain. Scalability and Privacy leakage are also major challenges while using Blockchain technology. Smart contract security [5], [25]is another main challenge in Blockchain, as smart contracts are more prone to attacks because of their vulnerabilities.

1.7 Introduction to Smart Contracts

There are continues improvements and inventions happening in Blockchain. The most popular and recent development is the creation of Smart Contracts[26]. The immutable program code stored on Blockchain is known as a smart contract[27]. It is responsible for implementing business logic and executing it when specific criteria are satisfied. Smart contracts are used to automate corporate activities in both public and permissioned blockchains[28]. The need of automation of traditional legal contracts along with certain laws of execution led smart contract to be invented[29]. Recently it has been an important feature of Blockchain applications. The self-executing nature of smart contracts[30] provides a tremendous opportunity in many fields. It is used in a range of applications, including finance, e-voting system, supply chain, digital identity, healthcare system, business process management, and even the Internet of Things and more, thanks to smart contracts. In the early 1990s, Nick Szabo invented and proposed the concept of smart contracts [25], [31]. The execution flow of the smart contract can be explained as shown in the figure below.

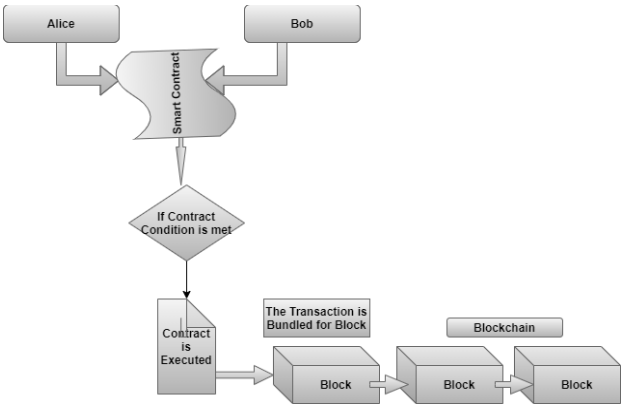

Figure 4. Smart contract execution

While doing a literature review of smart contracts, the answers were needed to various questions such as

RQ1. What platforms are available and what programming languages are used for smart contracts?

RQ2. Do smart contracts have any security flaws or limitations?

RQ3. Are there any solutions available to identify the vulnerabilities in smart contracts?

A search is done towards related publications in IEEE, ACM, Springer, and Elsevier, among other places. A lot of information was discovered about various Blockchain platforms and languages used to write smart contracts. Along with the different security issues and vulnerabilities that exist in contract code, the methods to address them were also discovered.

RELATED WORK

Many publications on smart contract challenges and vulnerabilities are studied throughout this review. Some articles have contributed to the discovery of various vulnerabilities in public Blockchains by studying the Ethereum platform. The weaknesses of smart contracts in private Blockchain are discussed in a few studies, and they did this by researching Hyperledger Fabric Blockchain. After studying significant number of the papers, around 27 papers are shortlisted here which focuses on smart contracts and their security in Ethereum platform, whereas being private Blockchain very few research is found on smart contract security in Hyperledger Fabric Blockchain. Around 20 papers are studied here on Formal verification techniques to verify the code of smart contracts.

This research study can be found unique as it highlights the majority of the flaws in both public and private Blockchain smart contracts at one place. Also, it covered the tools and strategies for detecting deficiencies in both blockchain platforms. The focus of this document is to present the study in an effective way so as to find an efficient solution for identified issues and facilitate the research in that direction. As a result, this article explores various smart contract platforms and domain-specific programming languages along with vulnerabilities.

The organization of the rest of the paper is as follows. Section 2, covers literature survey which elaborates security challenges in smart contracts. Section 3 addresses the vulnerabilities and attacks in smart contracts on public (Ethereum) and private Blockchain (Hyperledger Fabric). In section 4 security solutions for smart contracts containing tools and formal methods are explained. Finally, the survey is summarized with the help of research challenges, future directions and research needs.

2. LITERATURE SURVEY

This section presents study of different Blockchain platforms and Smart contracts[32]. The first subsection describes the platforms and languages[33] used to write smart contracts. In the following subsection, various challenges faced by developers while developing smart contracts[28] are discussed. The next subsection describes multiple vulnerabilities in smart contracts detected in Public and Private Blockchains[34]. The tools and methods to address these vulnerabilities[35] are discussed in the following subsection.

2.1 Platforms and Smart Contract Languages

Bitcoin was one of the first digital currencies to be utilized in a public blockchain. It is a decentralized digital currency developed for transferring Bitcoin (its cryptocurrency) without the need for intermediaries. Also, another public blockchain platform is Ethereum[34][36]. It is a decentralized open-source blockchain technology with smart contract functionality.

One of the famous and widely accepted private Blockchain frameworks is Hyperledger Fabric[37]. Hyperledger Fabric uses general-purpose programming languages, e.g., Go, Node.js, and Java, to implement smart contracts (called chain code in Hyperledger Fabric) [37].

In Table.1. a comparative study is given, including the various platforms, their features like type, languages used to write a smart contract, and whether it is turning complete and its paradigm.

Table 1: Blockchain Platforms and languages

Platform	Public or Permissioned	Supports Smart Contracts	Smart Contract Language	Bitcoin Cryptocurrency	Turing Complete	Paradigm
Bitcoin	Public	Yes	Ivy, RSk, BitML	Yes	No	Stack-Based (High Level)
Ethereum	Both	Yes	Solidity, Flint, SCILLA	Yes	Yes	Object-Oriented
Hyperledger Fabric	Permissioned	Yes	Go, Node.js, Java	No	Yes	General-purpose language
Neo	Both	Yes	C#,VB.Net, Java, Kotlin, Python	Yes	Yes	Object-Oriented, interpreted
Quorum	Permissioned	Yes	Solidity	No	Yes	Object-Oriented
Cardano	Public	Yes	Plutus (Functional language)	Yes	Yes	Functional language
EOS	Public	Yes	C++	Yes	Yes	Object-Oriented
R3 Corda	Permissioned	Yes	Kotlin	No	Yes	General-Purpose Language

2.2 Security Challenges in Smart Contracts:

Despite the increasing popularity of smart contracts, there are few potential challenges that developers are facing[25]. As described earlier, Blockchain technology has its own challenges, and among those, one crucial challenge is Smart contract security[38]. While writing the smart contracts and maintaining their security [38] [39], the developers found more challenges. Some of them are mentioned below.

2.2.1. Security: There is a high requirement for code security because of the sensitive nature of the information. The transactions are irreversible, and code is unmodifiable after deployment, so code must be secure enough. Then another challenge is, it is hard to guarantee security because of certain flaws in the compiler, limited tools or techniques to verify the code's correctness, and auditing the code. There is a need to apply best practices to write safe code and demand for efficient Formal verification techniques

2.2.2. Debugging: Debugging is painful as there are limited powerful interactive debuggers and current practices need improvement.

2.2.3. Programming Language: There are limited programming languages used for developing smart contracts. There are also limitations on logging or reporting features and a lack of standards or rules in available programming languages. There are also constraints on several local variables.

2.2.4. Ethereum Virtual Machine (EVM): One of the critical challenges is EVM. EVM's limitations include limited support for debugging, the inefficiency of bytecode execution, and limited stack size.

2.2.5. Gas: Special attention to gas consumption and error handling is needed. There are difficulties in handling gas problems, so there is a need for a tool to estimate gas usage.

2.2.6. Limited standardized knowledge and support from the Community: There is a lack of awareness of the technology and its societal features and challenges.

3. SMART CONTRACTS VULNERABILITIES IN PUBLIC AND PRIVATE BLOCKCHAIN

The last part states that smart contracts on the Ethereum Blockchain are written in the Domain-specific language (DSL) Solidity. In contrast, smart contracts on the Hyperledger Fabric network are written in the Go or JAVA languages. The following sections explain numerous vulnerabilities in Ethereum and Hyperledger Fabric smart contracts [41]in the form of a table that summarises the vulnerability name, description, and severity.

3.1 Vulnerabilities in Public Blockchain: Ethereum

Table 2 attempts to describe Ethereum smart contract vulnerabilities[41], [42]. It is discovered that three factors cause these flaws: first, the Solidity programming

language, second, the platform employed, and third, misunderstanding of standard practices. So here prepared the table listing these vulnerabilities and their description, explaining their rationale, and suggesting ways to mitigate them.

Table 2: Smart contract vulnerabilities in Ethereum

Cause	Vulnerability Name	Description	Weakness	Severity
Solidity Language	Reentrancy[41]	A function makes an external call to another untrusted contract, which now controls flow and calls back calling contract leading to exploitation of loophole and money withdraw	Improper behavioral workflow	Severe
	Unprotected self-destruct[41]	permitting access to an unauthorized actor	Improper access control	Severe
	Integer overflow (underflow)[41]	It occurs when an arithmetic operation results in an out-of-range number. (Maximum or minimum) of a type. So, use SafeMath libraries.	Incorrect Calculation	Severe
	Locked Money[41]	The wrong address is entered to transfer the money.	Improper Initialization	Severe
	Delegate calls to untrusted contract[41]	Calling a function of another untrusted contract.	Inclusion of functionality from untrusted control	Severe
EVM bytecode	Immutable bugs[39], [41]	Some bugs in the bytecode of Solidity are not detected.	Bugs in bytecode	Severe
	Ether lost in transfer[41]	Ethers are lost if the proper address of the receiver is not mentioned.	Ethers lost if sent to the orphan address.	Medium
	Stack size limit	Stack size is sometimes not adequate, which leads to error.	Call stack size limitation	Severe
Blockchain Platform	Transaction order dependence[42]	The reward is given to that person who first solves the maths function. So, commitment scheme is to be applied.	Race condition	Medium
	Weak randomness from chain attributes[42]	Random numbers generated by the algorithm can be predicted. So do it in two phases: 1. Commit 2. Reveal	Use of insufficiently random source	Medium
	Timestamp dependence[41]	Malicious miners adjust the timestamp so that they will get the benefit.	Inclusion of functionality from untrusted control	Medium
Misunderstanding of common practices	Mishandled exceptions[41]	A mishandled exception may cause an attack.	Improper handling of exceptional conditions	Severe
	Replay attack[43]	While using Digital signatures, users have to be careful as impersonating signatures can harm the system.	Improper cryptographic understanding	Severe

3.2 Attacks Exploiting the Vulnerabilities:

As per the study, there are various vulnerabilities in smart contracts because of different causes like Solidity language, EVM bytecode, or Blockchain system itself. These vulnerabilities are exploited by attackers, and they become successful in attacking the system, which leads to substantial financial loss. The following table 3 summarizes the various vulnerabilities and attacks[34], [35][43]

Table 3: Vulnerabilities causing various attacks.

Cause	Vulnerability Name	Attacks
Solidity Language	Reentrancy	The DAO attack[43]
	Unprotected self-destruct	Parity Multisig Wallet Attack[43]
	Integer overflow (underflow)	Integer overflow /underflow attack[43]
	Locked Money	The DAO attack[43]
	Call to the unknown/ Delegate call to untrusted contract	The DAO attack[43]
	Gasless send, Exception disorders	King of the Ether Throne [43]
	Keeping secrets	Multi-player games[43]
EVM bytecode	Immutable bugs, stack size limit	Rubixi, GovernMental attack[34], [43]
Blockchain Platform	Transaction order dependence	-
	Unpredictable state	GovernMental attack[35]
	Weak randomness from chain attributes	-
	Timestamp dependence	GovernMental attack [43]
Misunderstanding of common practices	Mishandled exceptions	The DAO attack [43]
	Replay attack	Replay attack [43]

3.3 Vulnerabilities in Private Blockchain: Hyperledger Fabric

Table 4 covers the numerous Fabric chaincode vulnerabilities[44]. The Fabric, as previously stated, employs the Go programming language[44] to build smart contracts. Three aspects are examined here as well to categorize the source of vulnerabilities: The first is the Go language, the second is the blockchain platform, and the third is a misunderstanding of standard practices, as well as the language's non-deterministic behaviour. In this section, a comprehensive survey of the risks developers faces while designing smart contracts and the risks they face when implementing their business logic on Hyperledger Fabric. The hazards and justifications are summarised in Table 4.

Table 4: Smart Contract Vulnerabilities in Hyperledger Fabric

Cause	Vulnerability Name	Rationale
Go Language (Non-determinism arising from language instructions)[44]	Global variables (Global State variables)	Global variables have a scope limited to only a single node, and they might no longer be consistent over all peers, which leads to inconsistency.
	KVS structure	Not deterministic
	Reified object address	Memory addresses used are dependent on the environment. Non-determinism occurs because of reified object addresses.
	Concurrency (Goroutines)	Non-deterministic behavior in Go is caused if programs running concurrently are not handled appropriately.
	Random number generation	May give different results as during the endorsing phase [19].
	System Timestamp	There may be variations in timestamp functions in peer-to-peer.
	Field declarations	Variable declared in a structure field may have different times for each peer
Non-determinism Caused From Accessing	Web Service	The difference in results after calling web services.
	System Command Execution	The output after execution of the system command should be the same for all peers.

Outside of Blockchain[44]	External File Accessing	Once the external file is accessed, the results should be the same.
	External Library Calling	The behavior of the external library should be considered. Developers need to be careful while applying third-party libraries.
State Database Specification[44]	Range query risk (Phantom reads)	In validation phase GetQueryResult are not re-executed,i.e.phantom reads (dirty data) are not detected. For example GetHistoryForKey() , GetPrivateDataQueryResult() need to be carefully used.
Fabric Specification (Undesired behavior arising from platform features)[44]	Cross Channel Chaincode Invocation	If two chain codes use the same channel, invoke a chain code from another[19]-[24]. But if not on the same channel, the data will not be saved in another channel.
	Read your write (read after write)	Only after a transaction is committed, a written statement takes its effect.
Misunderstanding of common practices[44]	Unhandled errors	Return values generated after errors should not be ignored, as ignored errors might lead to faulty execution.
	Unchecked input arguments	To avoid accessing a non-existent element, input arguments must be checked carefully.

4. SECURITY SOLUTIONS FOR SMART CONTRACTS SECURITY:

To solve these security problems, there are two approaches:

- Using Vulnerability scanning tools[41]
- Using Formal Verification method[45]: Mathematically proving the software is correct

Smart Contracts handles massive financial projects, digital assets, stock, or governmental applications, which can lead to many severe errors like money loss or privacy leakage. For example, DAO (Decentralized Autonomous Organization)[43] was the victim of an attack that happened in June 2016. The reason was a bug in its code and that cost loss of 60 million USD. The attacker exploited the reentrancy vulnerability. In the previous sections, we studied various vulnerabilities present in smart contracts and Blockchain platforms. To identify and detect these vulnerabilities, enormous tools are available. Some of the tools available for Ethereum Smart Contracts are:

- Security Tools - Input to this tool is provided in the form of either source code or the bytecode.
- Visualization Tools - It gives outputs in graphical form such as CFG (Control Flow Graphs) or Dependency Graphs.
- Dissemblers – It performs reverse task as that of assemblers. It converts the binary code back into high-level language code.
- Decompilers – Using decompilers, the binary code is converted to low-level language code.
- Miscellaneous Tools - which help give standard code metrics

Security Testing Methods:

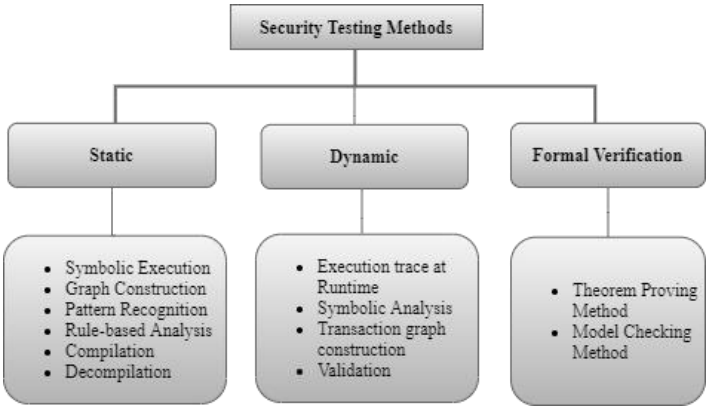

Figure 5: Security Testing methods

4.1 Security Tools or Vulnerability Scanning Tools for Smart Contracts

This section gives a list of different tools that identify specific security issues or vulnerabilities in a smart contract in public and private Blockchain[46]. Table 5. Summarizes tools used for Public Blockchain.

4.1.1. Smart Contracts Security Analysis Tools In Public Blockchain (Ethereum)

Table 5. Security Tools in Public Blockchain

Sr. No	Name Of the Tool	Description	Detecting Vulnerabilities
1	OYENTE[45] [46]	Two inputs are provided to this tool. 1. Ethereum smart contract bytecode 2. global state. Four components work as **CGFBuilder**: creates a Control Flow Graph **Explorer**: Symbolically executes contracts **Core Analysis**: Analyses to find any issue **Validator**: Validates to help in removing false positives	Timestamp dependency, Reentrancy, Call stack depth, Exception handling, Transaction order. Also detects Integer overflow/underflow and The DAO attack
2	EthIR[46][48]	EthIR is an extended version of OYENTE, which works as a decompiler. CFGs generated by OYENTE are modified, and the bytecodes' rule-based representation (RBR) is created.	Destroyable/Suicidal contract, Unsecured balance. Reentrancy, Unchecked, and failed send. It also detects DAO.
3	SMARTINSPECT [46][48]	Analyses the smart contracts which are deployed with the help of decompilation techniques and mirror-based reflection.	Reentrancy, highlight potential vulnerabilities in the code. It also detects The DAO attack
4	GASTAP[47][48]	Calculates the upper bound (Maximum) of Gas for Ethereum smart contracts to avoid the out-of-gas vulnerability. 1.GasTap constructs CFGs with the help of OYENTE 2. It decompiles low-level code to high-level representation using EthIR. 3. SACO determines the size of the relations 4. It generates the closed-form gas bands and even defines the equations required to calculate Gas.	Detects Out of Gas Vulnerability
5	SECURIFY[48]	EVM bytecode is decompiled, and semantic facts are extracted. It checks the security pattern, which is represented in Domain-specific language (DSL).	Transaction ordering ordering-dependent amount, receiver and transfer, Unhandled exceptions, Call stack depth limitation, Unchecked and Failed send, Non-validated arguments Unrestricted ether flow Detects Parity multisig wallet attack
6	MAIAN[48]	Analysis tool represented by Nikolic*etal.*. It is used to detect the specific behaviors of Ethereum smart contracts [28] as **Greedy contract**: Locks Ethers **Prodigal contract:** Leak Ether to an unknown address **Suicidal Contracts:** The contracts commit suicide because of the arbitrary external account.	Detects Greedy, Prodigal, Destroyable /Suicidal contracts, Unsecured balance, Call stack depth limitation, and Parity Multisig wallet attack
7	VANDAL[48]	Static security analysis framework. It translates smart contract byte codes to logic	Destroyable/Suicidal contract, Unsecured balance, Use of

		relations, including a bytecode scraper, a disassembler, a decompiler, and an extractor.	Origin Reentrancy, Detects DAO and Parity multisig wallet attack
8	SMARTCHECK[48]	This tool translates the code written in Solidity to the XML format and analyses using XPath queries to identify the issue.	Checks XPath patterns to highlight potential vulnerabilities in the code.
9	GASPER[48]	It detects overcharged representative patterns automatically	Detects those codes patterns which consume more Gas
10	MYTHRIL[48]	It is an open-source tool that uses a symbolic execution technique. Executes smart contract bytecode in a custom-built EVM	Exceptions, External Calls, Multiple Sends, Suicide
11	OSIRIS[46][48]	Tools are specifically designed to detect integers numbers vulnerabilities	Vulnerabilities related to integers
12	SLITHER[46][48]	Static analysis framework. It is used for vulnerability detection, automated optimization detection, code understanding, and assisted code review	It shows the position of the error in code and detection of low false-positives
13	ZEUS[46][48]	This tool is used to verify and validate the fairness of smart contracts.	Detects re-entrancy, unchecked, failed send, integer Overflow /underflow.

4.1.2 Smart Contracts Security Analysis Tools in Private Blockchain Hyperledger Fabric

In private Blockchain like Hyperledger fabric, developers use GO language to develop smart contracts. If developers use the GO tools properly for writing chain codes, the risks can be prevented. With the help of Go development tools used for chain code development, developers can minimize the risks.

A. Go Tools:

To make developers' tasks easy, a collection of tools and libraries are defined in GO-Tool[44].

In the table below, various Go tools are mentioned, designed to code and detect risks.

Table 6: Go tools

Sr. No	Name Of the Tool	Description	Detecting Vulnerabilities
1	gochecknoglobals	Global variables may create vulnerability in code, so no globals should be there in the code of Go language.	Global Variable, KVS Structure Iteration, Random Number Generation
2	Varcheck	This helps find global variables and constants that are not used in code [12][19].	Global Variable, KVS Structure Iteration
3	Errcheck	Unchecked errors are detected to prevent “Unhandled Errors” risks.	unchecked errors
4	Gosec	Gosec inspects the risks as Web Service System Command Execution, Unhandled errors, and Generating Random Number.	Random Number, web service
5	Golint	Golint makes suggestions for mechanically checkable items and CodeReview Comments.	System Command Execution, Unhandled Errors, web service
6	Go test-race	The aim is to avoid race conditions in code	System Command Execution

B. Chaincode Specific Tool

ChainSecurity has developed a vulnerability scanner tool that was made available to the developers as a web application known as Chaincode Scanner[44]. Whenever any Go project for non-commercial purposes is to be tested, the URL is provided to the Go project, and related risks can be identified.

Table 7. Chaincode specific tools

Sr. No	Name Of the Tool	Description	Detecting Vulnerabilities
1	Goroutines	Concurrency may lead to errors, so it is not preferred in chain code, and for this tool is used.	KVS Structure Iteration, Random Number Generation, global Variable, Web Service, Field Declarations
2	Global State	It takes care of global variable vulnerability.	Global Variable
3	Blacklisted Imports	Takes care of those files or libraries which may cause non-determinism (ex. timestamp).	System Timestamp, Random Number Generation
4	Map Range Iterations	It avoids range iterations as they are non-deterministic.	KVS Structure Iteration
5	Unchecked Input Arguments	It checks several arguments before they are used.	Unchecked Input Arguments, unhandled Errors,
6	Unhandled Errors	If errors are ignored or not handled properly, it may lead to improper execution.	Unhandled Errors, Unchecked Input Arguments
7	Read After Write	It is used to take care of the read and write sequence of variables.	Read Your Write
8	Phantom Read of Ledger	Results of phantom reads should not be used to manipulate the ledger.	Range Query Risk

4.2 FORMAL VERIFICATION METHOD:

Writing correct code is very hard, and when it is said that code is correct, it does precisely the same thing you think it does. Formal verification is the process of checking if the design satisfies the system requirements (properties) to check if the code behaves as expected using some formalism[45][49]. It is very hard to check the absence of undesired behavior, and the formal verification process helps in this regard. The formal verification is different than testing. Testing is done only on a finite set of specified inputs and checks the test cases where formal verification covers all input scenarios and all corner cases and detects bugs[49].

Formal verification is an effective method for ensuring the accuracy of smart contracts. Verification results are better when formal approaches are used. Formal verification ensures that nothing is lost in translation and the program is built to perform what it is supposed to accomplish[50]. An unambiguous mathematical language is used in this method to define the correctness and security of smart contracts.

4.2.1 Various Approaches in Formalization

The code and execution of smart contracts can be verified using two wide classifications such as contract-level and program-level[51]. In the contract-level approach, the transactions in smart contracts are considered as a black box where it only concerned about the smart contract's high-level behaviour. The details about implementation and execution are not verified in depth. In the program-level approach, more focus is on the implementation (i.e., source code) of smart contracts, which makes it platform-dependent[51].

Contract Level

The examples of contract level[51] models are state-transition systems, process algebra and set-based methods. In process algebra, concurrency is achieved. In-state transition systems, smart contracts are modelled in timed automata and Markov decision processes. Model checkers verifies state transition models concerning contract-specific properties in the state transition approach. Set theory and logic are used in the set-based model in formalizing the contracts.

Program level

With the help of lower-level representation, the Program level provides a white-box view of the smart contracts[50]. The source code of the smart contract, compiled bytecode, Abstract Syntax Tree (AST), control flow graphs are also verified[51][52]

4.2.2 Formal Verification Framework

With the use of a rigorous mathematical model[53], formal verification methods are utilized to validate and verify smart contracts[54]. Theorem provers are used in formal verification methods to prove specific features of a code of programming language, for instance functional correctness, runtime safety, soundness, and dependability[51][53].

This is accomplished by converting both the code and the specification into mathematical representations that

are then compared using mathematical proof. If they do not match, then the problem is identified and fixed. The main goal of formal verification is to make smart contracts less complicated.

Figure 6. depicts the framework[55] for formal verification.

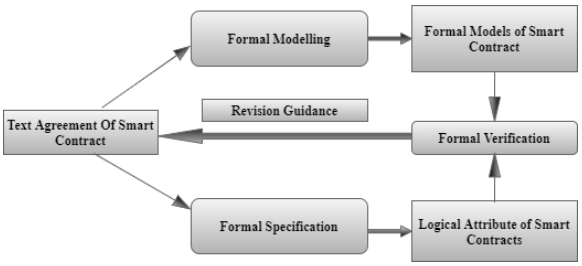

Figure 6. Formal Verification Method Framework[55]

4.3 Formal Verification Techniques

The Following are the techniques implied for formal verification of the smart contracts[50]:

4.3.1 Model checking:

Checks finite-state systems[51]. It verifies a system model by comparing it with smart contracts specifications.

4.3.2 Theorem Proving:

This method encodes the contracts and their properties in mathematical logic[56]. It Supports the verification of infinite systems. As this approach is not totally automatic, it requires human involvement and expertise.

4.3.3 Symbolic execution[57]:

It explores possible execution paths. Symbols are used along with symbolic state. It performs symbolic execution by traversing Control Flow graphs generated from bytecode.

4.3.4 Runtime Verification and Testing[58]:

It verifies and tests the properties of an executing program.

4.3.5 Program Based Formal Verification:

Program-based verification places a greater emphasis on smart contract code. After translating smart contract programming into formal languages, vulnerabilities are discovered. The following are some instances and tools used in program-based verification.

- VaaS Platform[59]: VaaS is used with Blockchain platforms like EOS, Ethereum, and Fabric.
  In this method, smart contract code is translated to Coq code, and then correctness is checked.
- Formal symbolic process virtual machine (FSPVM): Yang and Lei[49][56] proposed this method. The reliability and security properties of smart contracts are verified in this method.
- Isabelle /HOL proof assistant model[59]: This verifies the binary Ethereum code. So binary code is extracted and then translated into an AVL tree which is analysed for verification.
- F* Framework [60]: The Solidity language is translated into a functional F* language where the correctness of contract is verified. It also analyses the EVM bytecode of the contracts.

4.3.6 Behavior-Based Formal Verification:

Behavior-based verification[51] is used to address issues encountered during the execution of smart contracts. For instance improper operations and malicious attacks are identified. Following are the methods used in Behavior-Based Formal Verification[51]

- Finite State Machine and PRISM Tool[61]
- Runtime verification method[58]
- Probabilistic formal models to verify contracts[56]
- Promela language and SPIN tool to verify smart contracts [51]
- Use BIP (Behavior Interaction Priorities) Framework[51]

5. RESEARCH ISSUES AND CHALLENGES

Though Smart contacts are beneficial and immutable, they are vulnerable to various attacks, which causes significant financial loss or serious breach in security. If the smart contracts are not written with proper care and precautions, attackers can exploit them, leading to a major financial crisis. The study found that there are certain vulnerabilities in smart contracts at the design level and code level. To address these issues, many tools and methods have been invented[62], [63]. Certain formal verification techniques are designed for identifying the vulnerabilities in EVM bytecode as well[64].But not all or maximum vulnerabilities are addressed by a single tool or any single formal verification method. Though there are plenty of tools available to detect vulnerabilities, some vulnerabilities are not yet discovered, specifically vulnerabilities caused by programming languages used to write smart contracts and EVM bytecode. Throughout this literature survey, it is analyzed that tools like OYENTE, EthIR, SECURIFY, MAIAN, SMARTCHECK, GASPER can detect the majority of the vulnerabilities such as Reentrancy, Integer Overflow and Underflow, Transaction Order Dependency and so on[65]. But the vulnerabilities present in the bytecode of smart contracts are not yet detected by any of the tools and not by any formal verification methods. So, there is a need for novel, efficient and powerful techniques to detect the vulnerabilities and resolve them to make the system more secure. The loopholes present in EVM bytecode

are challenging to identify because it is in the form of machine-level language. As bytecode is very difficult to understand and interpret, it requires very highly skilled resources to go through the details of bytecode and find out the vulnerability. The Vulnerabilities present in EVM bytecode result in some ruinous attacks for example the Rubixi attack or Governmental attack.

By study, it can be stated that more focus should be given to the programming of smart contracts. So, program-level security is important. The root cause of bugs should be identified, new vulnerabilities should be discovered, and the generalization of maximum vulnerabilities can be defined accordingly.

To address the vulnerabilities in Blockchain smart contracts that are not detected by existing tools and formal verification methods but are harmful to the applications, there is a need for a novel approach or framework. There is an urgent need for a thorough and extensive method to ensure the security and correctness of smart contracts. To achieve this, a mathematical model of formal specification for identifying the functional correctness of Blockchain Smart Contracts can be proposed. Along with vulnerability scanning discovering unknown vulnerabilities and optimizing these methods can be achieved in the future. The objective is to design a formal verification framework for the effective verification of undetected smart contract vulnerabilities. The expected outcomes will be an efficient, improved formal verification method to detect the vulnerabilities in smart contracts. Along with this using safe language for writing smart contracts will help to eliminate many security risks. The contract capable of healing itself is also one of the prominent ways to reduce risks.

6. CONCLUSION AND RESEARCH OPPORTUNITIES

Throughout this research, a systematic review is performed on the introduction of smart contracts, various Blockchain platforms, programming languages, numerous vulnerabilities, and tools for security in Ethereum and Hyperledger Fabric and formal verification methods for smart contract correctness assurance. The study discovered that there is a significant need to improve the security and performance of smart contracts to cope with practical and competitive decentralized applications in our review. Ethereum is the most popular blockchain platform and Solidity is widely used programming language now a days. At the same time, as the number of enterprise apps grows, private Blockchain is becoming more important. According to the study, permissioned blockchain platforms for example Corda, Tendermint, Hyperledger Fabric, and Quorum will be in more demand. As a result, executing smart contracts, enhancing their security, correctness, and performance, and implementing blockchain-dependent apps all necessitate more and more study to make blockchain-based applications competent in a real-time context.

With the improved popularity and adoption of Blockchain Technology, implementing formal verification techniques for valid and error-free smart contracts has been a demanding research topic in modern age. In the future, the research in the direction of smart contract designing and formulation along with security awareness using formal verification methods will be more fruitful for the Blockchain fraternity.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System.” [Online]. Available: www.bitcoin.org

[2] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” in *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, Sep. 2017, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.

[3] X. Xu *et al.*, “A Taxonomy of Blockchain-Based Systems for Architecture Design,” in *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, May 2017, pp. 243–252. doi: 10.1109/ICSA.2017.33.

[4] X. Xu *et al.*, “The blockchain as a software connector,” in *Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016*, Jul. 2016, pp. 182–191. doi: 10.1109/WICSA.2016.21.

[5] O. Ali, A. Jaradat, A. Kulakli, and A. Abuhalimeh, “A Comparative Study: Blockchain Technology Utilization Benefits, Challenges and Functionalities,” *IEEE Access*, vol. 9, pp. 12730–12749, 2021, doi: 10.1109/ACCESS.2021.3050241.

[6] Institute of Electrical and Electronics Engineers and IEEE Technology and Engineering Management Society, *2017 IEEE Technology and Engineering Management Conference (TEMSCON).*

[7] P. Bhattacharya, A. Singh, A. Srivastava, and A. Mathur, “A Systematic Review on Evolution of Blockchain Generations ITEE Journal A Systematic Review on Evolution of Blockchain Generations,” 2018. [Online]. Available: https://www.researchgate.net/publication/330358000

[8] M. H. Miraz and M. Ali, “Applications of blockchain technology beyond cryptocurrency,” *Annals of Emerging Technologies in Computing*, vol. 2, no. 1, pp. 1–6, Jan. 2018, doi: 10.33166/AETiC.2018.01.001.

[9] International Conference on Electrical Engineering and Computer Science 2017 Palembang, Institute of Electrical and Electronics Engineers Indonesia Section, International Conference on Electrical Engineering and Computer Science 2017.08.22-23 Palembang, ICECOS Conference 2017.08.22-23 Palembang, and ICECOS 2017.08.22-23 Palembang,

Sustaining the cultural heritage toward the smart environment for better future ICECOS 2017 Conference : proceedings : August 22-23, 2017, Horison Ultima Hotel, Palembang.

[10] I. Eyal, "COVER FEATURE BLOCKCHAIN TECHNOLOGY IN FINANCE."

[11] D. Magazzeni, P. McBurney, and W. Nash, "COVER FEATURE BLOCKCHAIN TECHNOLOGY IN FINANCE Validation and Verification of Smart Contracts: A Research Agenda." [Online]. Available: www.nortonrosefulbright.com/knowledge/publica-

[12] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart Contract Templates: foundations, design landscape and research directions," Aug. 2016, [Online]. Available: http://arxiv.org/abs/1608.00771

[13] G. W. Peters and E. Panayi, "Understanding Modern Banking Ledgers through Blockchain Technologies: Future of Transaction Processing and Smart Contracts on the Internet of Money," 2015. [Online]. Available: http://ssrn.com/abstract=2692487

[14] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4. Institute of Electrical and Electronics Engineers Inc., pp. 2292–2303, 2016. doi: 10.1109/ACCESS.2016.2566339.

[15] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, May 2018, doi: 10.1016/j.future.2017.11.022.

[16] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proceedings - 2016 2nd International Conference on Open and Big Data, OBD 2016*, Sep. 2016, pp. 25–30. doi: 10.1109/OBD.2016.11.

[17] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeDShare: Trust-Less Medical Data Sharing among Cloud Service Providers via Blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, Jul. 2017, doi: 10.1109/ACCESS.2017.2730843.

[18] M. Hölbl, M. Kompara, A. Kamišalić, and L. N. Zlatolas, "A systematic review of the use of blockchain in healthcare," *Symmetry*, vol. 10, no. 10, 2018, doi: 10.3390/sym10100470.

[19] T. Mikula and R. H. Jacobsen, "Identity and access management with blockchain in electronic healthcare records," in *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018*, Oct. 2018, pp. 699–706. doi: 10.1109/DSD.2018.00008.

[20] *Hawaii International Conference on System Sciences 2020*.

[21] P. Mccorry, S. F. Shahandashti, and F. Hao, "A Smart Contract for Boardroom Voting with Maximum Voter Privacy."

[22] N. Kshetri and J. Voas, "Blockchain-Enabled E-Voting," *IEEE Software*, vol. 35, no. 4, pp. 95–99, Jul. 2018, doi: 10.1109/MS.2018.2801546.

[23] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on Blockchain technology? - A systematic review," *PLoS ONE*, vol. 11, no. 10, Oct. 2016, doi: 10.1371/journal.pone.0163477.

[24] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *International Journal of Network Security*, vol. 19, no. 5, pp. 653–659, Sep. 2017, doi: 10.6633/IJNS.201709.19(5).01.

[25] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. Seneviratne, "Blockchain-based Smart Contracts - Applications and Challenges," Sep. 2018, [Online]. Available: http://arxiv.org/abs/1810.04699

[26] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F. Y. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, Nov. 2019, doi: 10.1109/TSMC.2019.2895123.

[27] W. Egbertsen, G. Hardeman, M. van den Hoven, G. van der Kolk, and A. van Rijsewijk, "Replacing Paper Contracts With Ethereum Smart Contracts Contract Innovation with Ethereum," 2016.

[28] L. W. Cong and Z. He, "Blockchain Disruption and Smart Contracts," *Review of Financial Studies*, vol. 32, no. 5. Oxford University Press, pp. 1754–1797, May 01, 2019. doi: 10.1093/rfs/hhz007.

[29] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 50759–50779, 2019. doi: 10.1109/ACCESS.2019.2911031.

[30] 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," 2015.

[31] M. Alharby and A. van Moorsel, "Blockchain Based Smart Contracts : A Systematic Mapping Study," Aug. 2017, pp. 125–140. doi: 10.5121/csit.2017.71011.

[32] 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)*, 2017, vol. 10323 LNCS, pp. 494–509. doi: 10.1007/978-3-319-70278-0_31.

[33] D. Harz and W. Knottenbelt, "Towards Safer Smart Contracts: A Survey of Languages and Verification Methods," Sep. 2018, [Online]. Available: http://arxiv.org/abs/1809.09805

[34] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts." [Online]. Available: https://coinmarketcap.com/currencies/ethereum

[35] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart Contract: Attacks and Protections," *IEEE Access*, vol. 8, pp. 24416–24427, 2020, doi: 10.1109/ACCESS.2020.2970495.

[36] "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER EIP-150 REVISION."

[37] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned

Blockchains," Jan. 2018, doi: 10.1145/3190508.3190538.

[38] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart contract security: A software lifecycle perspective," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 150184–150202, 2019. doi: 10.1109/ACCESS.2019.2946988.

[39] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey," Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.08605

[40] R. Gupta, S. Tanwar, F. Al-Turjman, P. Italiya, A. Nauman, and S. W. Kim, "Smart Contract Privacy Protection Using AI in Cyber-Physical Systems: Tools, Techniques and Challenges," *IEEE Access*, vol. 8, pp. 24746–24772, 2020, doi: 10.1109/ACCESS.2020.2970576.

[41] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey," Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.08605

[42] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons, "Smart contracts vulnerabilities: A call for blockchain software engineering?," in *2018 IEEE 1st International Workshop on Blockchain Oriented Software Engineering, IWBOSE 2018 - Proceedings*, Mar. 2018, vol. 2018-January, pp. 19–25. doi: 10.1109/IWBOSE.2018.8327567.

[43] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart Contract: Attacks and Protections," *IEEE Access*, vol. 8, pp. 24416–24427, 2020, doi: 10.1109/ACCESS.2020.2970495.

[44] R. Tonelli, IEEE Computer Society, Institute of Electrical and Electronics Engineers, and E. IEEE International Conference on Software Analysis, *IWBOSE '19 : 2019 IEEE 2nd International Workshop on Blockchain Oriented Software Engineering (IWBOSE '19) : February 24, 2019, Hangzhou, China*.

[45] X. Bai, Z. Cheng, Z. Duan, and K. Hu, "Formal modeling and verification of smart contracts," in *ACM International Conference Proceeding Series*, Feb. 2018, pp. 322–326. doi: 10.1145/3185089.3185138.

[46] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart contract security: A software lifecycle perspective," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 150184–150202, 2019. doi: 10.1109/ACCESS.2019.2946988.

[47] J. Liu and Z. Liu, "A Survey on Security Verification of Blockchain Smart Contracts," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 77894–77904, 2019. doi: 10.1109/ACCESS.2019.2921624.

[48] A. L. Vivar, A. T. Castedo, A. L. S. Orozco, and L. J. G. Villalba, "An analysis of smart contracts security threats alongside existing solutions," *Entropy*, vol. 22, no. 2, Feb. 2020, doi: 10.3390/e22020203.

[49] Z. Yang and H. Lei, "Formal process virtual machine for smart contracts verification," *International Journal of Performability Engineering*, vol. 14, no. 8, pp. 1726–1734, Aug. 2018, doi: 10.23940/ijpe.18.08.p9.17261734.

[50] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A Survey of Smart Contract Formal Specification and Verification," *ACM Computing Surveys*, vol. 54, no. 7, pp. 1–38, Sep. 2022, doi: 10.1145/3464421.

[51] T. Abdellatif, K.-L. Brousmiche, and K.-L. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models." [Online]. Available: https://hal.archives-ouvertes.fr/hal-01760787

[52] W. Ahrendt *et al.*, "Verification of Smart Contract Business Logic Exploiting a Java Source Code Verifier." [Online]. Available: https://git.io/fx6cn.

[53] W. Xu and G. A. Fink, "Building Executable Secure Design Models for Smart Contracts with Formal Methods."

[54] L. Alt and C. Reitwiessner, "SMT-based verification of solidity smart contracts," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11247 LNCS, pp. 376–388. doi: 10.1007/978-3-030-03427-6_28.

[55] 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.

[56] Z. Yang and H. Lei, "Lolisa: Formal syntax and semantics for a subset of the solidity programming language in Mathematical Tool Coq," *Mathematical Problems in Engineering*, vol. 2020, 2020, doi: 10.1155/2020/6191537.

[57] I. Grishchenko, M. Maffei, and C. Schneidewind, "A Semantic Framework for the Security Analysis of Ethereum smart contracts," Feb. 2018, doi: 10.1007/978-3-319-89722-6_10.

[58] J. Ellul and G. J. Pace, "Runtime Verification of Ethereum Smart Contracts," in *Proceedings - 2018 14th European Dependable Computing Conference, EDCC 2018*, Nov. 2018, pp. 158–163. doi: 10.1109/EDCC.2018.00036.

[59] S. Amani, M. Bortin, M. Bégel, and M. Staples, "Towards verifying ethereum smart contract bytecode in Isabelle/HOL," in *CPP 2018 - Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, Co-located with POPL 2018*, Jan. 2018, vol. 2018-January, pp. 66–77. doi: 10.1145/3167084.

[60] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, C´edric Fournet, Anitha Gollamudi, Georges Gonthier, and Nadim Kobeissi, "F star Bhargavan formal verification paper".

[61] A. Mavridou and A. Laszka, "Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach," Nov. 2017, [Online]. Available: http://arxiv.org/abs/1711.09327

[62] S. Akca, A. Rajan, and C. Peng, "SolAnalyser: A Framework for Analysing and Testing Smart Contracts," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Dec. 2019, vol.

2019-December, pp. 482–489. doi: 10.1109/APSEC48747.2019.00071.

[63] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "SAFEVM: A Safety Verifier for Ethereum Smart Contracts," Jun. 2019, [Online]. Available: http://arxiv.org/abs/1906.04984

[64] E. Hildenbrandt *et al.*, "KEVM: A complete formal semantics of the ethereum virtual machine," in *Proceedings - IEEE Computer Security Foundations Symposium*, Aug. 2018, vol. 2018-July, pp. 204–217. doi: 10.1109/CSF.2018.00022.

[65] J. Liu and Z. Liu, "A Survey on Security Verification of Blockchain Smart Contracts," *IEEE Access*, vol. 7. Institute of Electrical and Electronics Engineers Inc., pp. 77894–77904, 2019. doi: 10.1109/ACCESS.2019.2921624.