An emperical study for automatic detection of Blockchain API misuses

Sarmad Wani
Supervisor Mojtaba Shahin
RMIT University

June 2024

Abstract

The Australian federal government released its National Blockchain Roadmap in February 2020, citing predictions that blockchain technology would generate USD 175 billion in global trade value by 2025, and more than USD 3 trillion by 2030. According to Fortune Business Insights, the global blockchain market is expected to hit USD 163 billion by 2029 (up from its current state of USD 7 billion). [2] Since its inception in 2009, the blockchain technology has shown promising application prospects. Although there have been in past some studies on the security and privacy issues of blockchain as a whole, there lacks a empirical study on the API vulnerabilities and misuses dimension in Blockchain system. In this paper, we conduct an empirical study on the security threats to blockchain API sub system on popular Ethereum framework.

1 Introduction

Ethereum is now (02/01/2024) the most widely used blockchain supporting smart contracts, where the number of weekly developers (306 weekly) and total number of applications per week (4,479 total dApps), provides a robust environment for application interoperability and innovation [36]. The structure of Ethereum based Dapp can be shown in figure 1 below:

The introduction of Turing-complete programming languages to enable users to develop Smart Contracts (SCs) running on the blockchain marks the start of blockchain 2.0 era since 2015. Smart contracts are essentially containers of code that encode and mirror the real-world contractual agreements

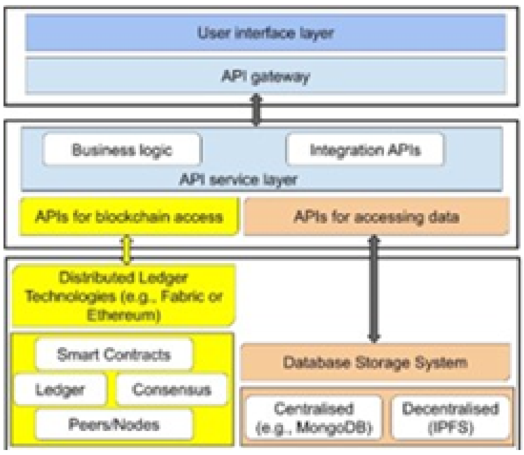


Figure 1: Key Components of dAPPs (Struture)

in the cyber realm. A key premise for contracts is that they represent a binding agreement between two or more parties, where every entity must fulfil their obligations according to the agreement. Another important element is that the agreement is enforceable by law, usually through a legal centralised entity (organisation). However, smart contracts replace the trusted third parties; that is, the intermediaries between contract members. They leverage this with the help of automatic code execution that is distributed and verified by the network nodes in a decentralised blockchain network. They also enable transactions between untrusted parties without any intermediary commission fees, (ii) the third-party dependence, and (iii) the need of mutual interaction directly of the counter-parties [40].

With the decentralized **consensus mechanism** of blockchain, smart contracts allow mutually distrusted users to complete data exchange or transaction without the need of any third-party trusted authority. There are different blockchain platforms that can be utilised to develop smart contracts, with Ethereum being the most common [39]. In the existing blockchain systems and SC System, there are four major consensus mechanisms [41]: PoW (Proof of Work), PoS (Proof of Stake), PBFT (Practical Byzantine Fault Tolerance), and DPoS (Delegated Proof of Stake). (Other consensus mechanisms, such as PoB (Proof of Bandwidth), PoET (Proof of Elapsed Time), PoA(Proof of Authority) and so on, are also used in some blockchain systems). The two most popular blockchain systems (i.e., Bitcoin and Ethereum) use the PoW mechanism that uses the solution of puzzles to prove the credibility of the data.

In different blockchain systems, the **block structure** may vary in detail. Typically in Bitcoin, each block contains PrevHash, nonce, and Tx [41]. In

$$SHA256\left(PrevHash \,||\, Tx1 \,||\, Tx2 \,||\, \ldots \,||\, nonce\right) < Target \tag{1}$$

Figure 2:

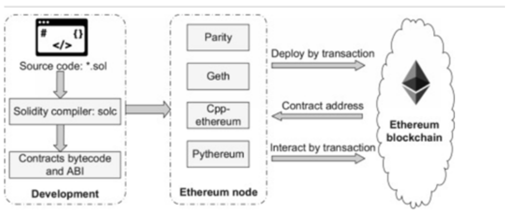


Figure 3: SC Development and Deployment

particular, PrevHash indicates the hash value of the last generated block, and Tx s denote the transactions included in this block. The value of nonce is obtained by solving the PoW puzzle. A correct nonce should satisfy that the hash value shown in Eq. (1)/Fig 2 is less than a target value, which could be adjusted to tune the difficulty of PoW puzzle.

In blockchain 2.0 stage, **smart contract** is introduced so that developers can create various applications through smart contracts. A smart contract can be considered as a lightweight dAPP (decentralized application) and Ethereum is a typical system of blockchain 2.0. Each Ethereum node runs an EVM (Ethereum Virtual Machine) that executes smart contracts. In Ethereum, developers can use a variety of programming languages to develop smart contracts, such as Solidity (the recommended language), Serpent, and LLL. Since these languages are Turing-complete, smart contracts can achieve rich functions. Figure 1 shows the process of smart contracts' development, deployment and interaction. Each deployed smart contract corresponds to a unique address, through which users can interact with the smart contract through transactions by different clients (e.g., Parity, Geth, etc.). Since smart contracts can call each other through messages, developers can develop more feature-rich dAPPs based on available smart contracts.

Since blockchain is one of the core technology in FinTech (Financial Technology) industry, users are very concerned about its security. Some security vulnerabilities and attacks have been recently reported. Loi et al. discover that 8,833 out of 19,366 existing Ethereum contracts are vulnerable [37]. For instance, in June 2016, the criminals attacked the smart contract DAO [38] by exploiting a recursive calling vulnerability, and stole around 60 million dollars.

We divide the common blockchain risks into nine categories, as shown in Table figure 5, and detail the causes and possible consequence of each risk. The risks described in Serial no 3.1 exist in blockchain 1.0 and 2.0, and their causes are mostly related to the blockchain operation mechanism. By contrast, the risks introduced in Serial no 3.2 are unique to blockchain 2.0, and are usually resulted from the development, deployment, and execution of smart contracts. We have put a special focus in this paper on serial no 3.2 vulnerabilities because they are dealing with our Research Questions (RQ) to analyse BC-API vulnerabilities.

Blockchain APIs serve as the primary interface for interacting with blockchain networks, facilitating data querying, transaction execution, through smart contract interactions. In order to guarantee the security of blockchain network, smart contracts are generally executed in SEE (e.g., EVM in Ethereum and Docker container in Hyperledger Fabric), which is not allowed to import external information. Hence, smart contract needs trusted data feeds (Oracles) to provide external states about the real world in the form of a transaction (because any information that is not generated by a transaction has to be introduced as data attached to a transaction [11]) in a secure and trusted manner, thereby ensuring the deterministic contract execution results [46]. Although APIs have added benefits for the performance, flexibility of Blockchains, they can quickly turn from being a convenient asset to a liability thereby posing vulnerabilities to business. For instance, high-profile DAO attack on Ethereum in 2016 siphoning billions of dollars of investor money.

These vulnerabilities can be present in any part of the API, from the design phase to the deployment stage that can result in severe consequences, such as data breaches, unauthorized access, and even system crashes creating a threat to confidentiality, integrity and authorization of the blockchain system. Despite the significant role that APIs play in software applications dAPPs, they are often left unprotected due to misconfigurations or lack of security measures.

Lately, plenty of research has taken place regarding API vulnerability misuse detection in software as a whole. However, a significant gap exists on our knowledge about the Blockchain API security misuse in specific, especially given the fact that blockchain exchange is still unfolding in API usage. Therefore, in this paper, we intend to explore the common Blockchain API misuses vulnerability and its detection using an open source prominent detection tool OYENTE [46] used widely in the industry nowadays for bug detection in BCs.

2 Research Questions:

Like any other API, the APIs in BC-Apps need to be tested for any potential flaws like unauthorized access, encrypted data in transit, and cross site request forgery, automatic call over a large number of transactions. Such an error in the API deployment could be very costly, especially in BC networks where transaction processing has a cost (e.g., gas in Ethereum BC). The API tests need to check that all the interactions between applications/users and the backend (BC network) in the BC-App are as per the predefined specifications, and the performance of the interaction is correct and smooth.

RQ (1) Analyzing Ethereum – API vulnerabilities using open-source OYENTE Testing Tool while classifying vulnerabilities under a novel classification taxonomy.

Motivation:

• Unique Interface between BC and APIs: The BC-API testing procedure is unique (than general API testing) and therefore a subject of specific interest for this research proposal. In a practical BC- App scenario, there exist two types of interfaces. First is between the BC and the application in which API is being integrated, and second is between the various components of a BC-based system. The former is more challenging to test in an efficient manner, as it requires the significant knowledge of both application domain, BC platform and specialized testing tools that can work with both these domains. For instance, usually the users of DApp interact with the BCs using a web application browser. In the existing test scenarios, the methods that test the web applications only consider browser-side code (Java script code), while the SC analysis tools consider only the SC side code (Ruby or Solidify code). The approach of independent testing makes it challenging to use these techniques as they are in the DApp setting. Hence, our research motivation considers the fact that the testers need to work together to understand the interfacing between BC Platform and applications.

• Oracle problem: Connectability to off-chain real world data for smart contracts in blockchain has vulnerabilities. E.g. BC's block Hash verification process in JSON files (CIA security Principles). All the APIs would require thorough testing to ensure that there are no security issues and that the service integration works seamlessly. For instance, Testing APIs helps in BC's block hash verification process. Each block information has a unique hash that changes when there is any change performed in the block. Hence, the APIs that fetch and verify these hashes can be validated through API testing.

• Previous research has created several classification taxonomies related to vulnerabilities of SCs to categorize the vulnerabilities based upon EVM, BlockChain that are broader classes and give less information to the readers about the code's internal drawback. [44] [45] A good understanding of vulnerabilities requires coding examples along with description providing coding examples.

• The Solidity programming language has undergone major changes. In this survey paper, we replicate vulnerabilities in Solidity code using the latest "solc" compiler version 5.0.

RQ 2: How effectively the Open-Source test tool OYENTE (2016) works to identify Ethereum - API misuses

Motivation:

• API testing performance in Ethereum is largely unexplored in past SC analysis, while at the same time the API-BC integration is exponentially increasing with added functionalities in dAPPs. OYENTE has been largely explored in context of broader SC vulnerability Analysis and not in the specific context of Oracle vulnerability tests.

• Detection tools designed to identify blockchain API misuses are instrumental in proactively mitigating security risks and ensuring the integrity of blockchain ecosystems. By evaluating and comparing performance of these tools on various Benchmark parameters, we aim to provide insights into their capabilities, limitations, and real-world applicability, thereby informing practitioners and researchers about the most suitable tools for securing blockchain applications.

• The growing demand for robust security solutions in the blockchain industry (CIA in Blockchains) motivates researchers to explore and improve the effectiveness of detection tools. [29].

• Blockchain technology testers should be able to select and prioritize the tool usage by considering the features and capabilities of each detection tool, such as static analysis, dynamic analysis, and machine learning algorithms. [44].

To Summarize, the key contributions of this paper are:

• A classification of the SC security vulnerabilities according to the domain specific classification taxonomy.

• A description of SC API vulnerabilities through updated sample code.

• A discussion of performance parameters of open source OYENTE vulnerability detection tool.

The remainder of this paper is organized as follows: section III presents our classification of SC's vulnerabilities. Section IV presents the description of SC API vulnerabilities with respect to domain knowledge along with some sample code. Section V lists the testing tools developed for SCs. Section VI concludes the paper and highlights the future research directions.

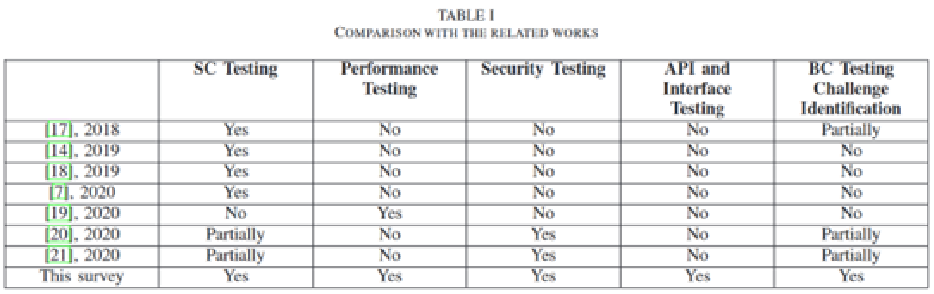
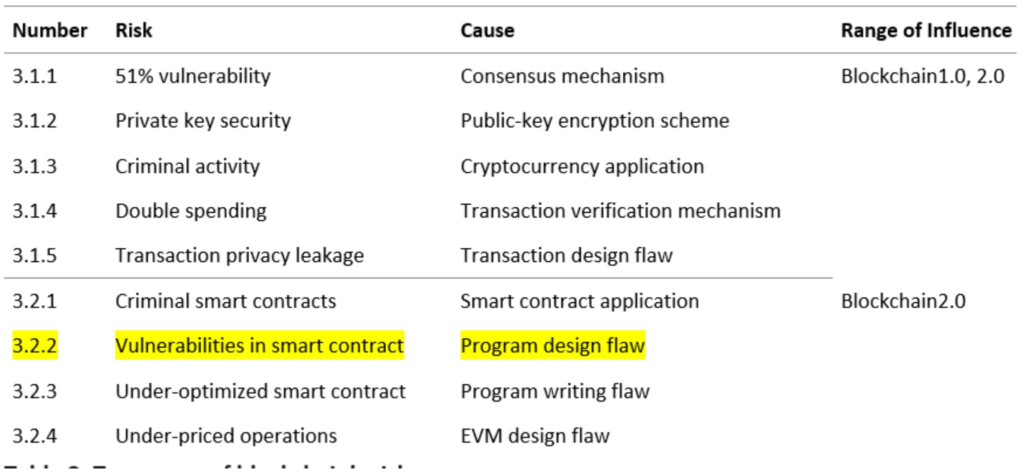


TABLE I
COMPARISON WITH THE RELATED WORKS

	SC Testing	Performance Testing	Security Testing	API and Interface Testing	BC Testing Challenge Identification
[17], 2018	Yes	No	No	No	Partially
[14], 2019	Yes	No	No	No	No
[18], 2019	Yes	No	No	No	No
[7], 2020	Yes	No	No	No	No
[19], 2020	No	Yes	No	No	No
[20], 2020	Partially	No	Yes	No	Partially
[21], 2020	Partially	No	Yes	No	Partially
This survey	Yes	Yes	Yes	Yes	Yes

Figure 4: Comparison of Past works on BC-API Testing



Number	Risk	Cause	Range of Influence
3.1.1	51% vulnerability	Consensus mechanism	Blockchain1.0, 2.0
3.1.2	Private key security	Public-key encryption scheme	
3.1.3	Criminal activity	Cryptocurrency application	
3.1.4	Double spending	Transaction verification mechanism	
3.1.5	Transaction privacy leakage	Transaction design flaw	
3.2.1	Criminal smart contracts	Smart contract application	Blockchain2.0
3.2.2	Vulnerabilities in smart contract	Program design flaw	
3.2.3	Under-optimized smart contract	Program writing flaw	
3.2.4	Under-priced operations	EVM design flaw	

Figure 5: Taxonomy of BC Risk

3 Literature Review:

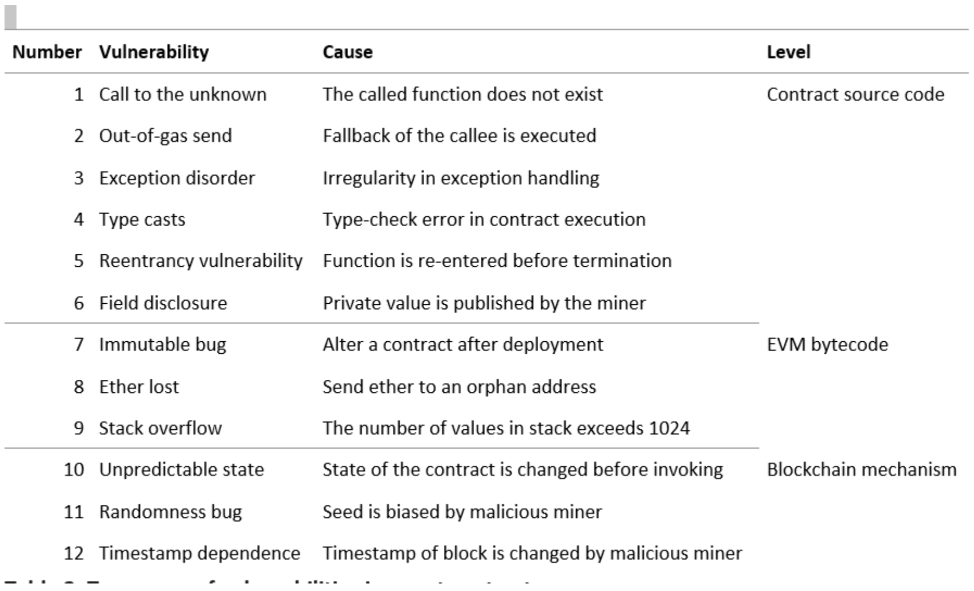
Literature review with respect to Risks/Vulnerabilities to SCs:

As a whole, we divide the common blockchain risks into nine categories, as shown in Table 2, and detail the causes and possible consequence of each risk. [47]. In this paper we research the specific Vulnerability found in Ethereum Blockchain SCs related to API calling.

Vulnerabilities in smart contract

As programs running in the blockchain, smart contracts may have security vulnerabilities caused by program defects when API calls take place. Nicola et al. [43] conduct a systematic investigation of 12 types of vulnerabilities in smart contract, as shown in 3.

Previous research work has created several taxonomies related to vulnera-



Number	Vulnerability	Cause	Level
1	Call to the unknown	The called function does not exist	Contract source code
2	Out-of-gas send	Fallback of the callee is executed	
3	Exception disorder	Irregularity in exception handling	
4	Type casts	Type-check error in contract execution	
5	Reentrancy vulnerability	Function is re-entered before termination	
6	Field disclosure	Private value is published by the miner	
7	Immutable bug	Alter a contract after deployment	EVM bytecode
8	Ether lost	Send ether to an orphan address	
9	Stack overflow	The number of values in stack exceeds 1024	
10	Unpredictable state	State of the contract is changed before invoking	Blockchain mechanism
11	Randomness bug	Seed is biased by malicious miner	
12	Timestamp dependence	Timestamp of block is changed by malicious miner	

Figure 6: Taxonomies of Vulnerabilities in a Smart Contract

bilities of SCs. The typical approach is to categorize the vulnerabilities based upon EVM, BlockChain, and Solidity associated issues as discussed in [51]. The above three are broader classes and give less information to the readers about the code's internal drawback.

Luu… [44] proposes to give the 4 kinds of potential security bugs classification in context of larger Smart Contract Code analysis in his research. We are classifying BC-API Security vulnerabilities based on the same classification taxonomy to use it in our Code analysis specific to our research of Ethereum API/Oracle testing as it gives an in depth evaluation of BC Bugs at code level unlike other studies that classify bugs based on a broader tool level. For instance, the survey in [52] classifies the SC vulnerabilities in the context of EVM and Solidity. Other significant survey grouped the vulnerabilities based upon layering [53] like application layer, data layer, and consensus layer. Furthermore, the work in [54] used NIST Bugs Framework for the classification.

As SC introduced new kinds of vulnerabilities not common in traditional programming paradigms, a good understanding of vulnerabilities requires coding examples along with description [56]. Work in [52], [53] also provided coding examples, but we address a different set of vulnerabilities. It is worth mentioning that the most relevant survey on the SC attacks with vulnerable SC code is available in [55], published in 2017. However, since then, the Solidity programming language has undergone major changes. In this survey paper, we replicate vulnerabilities in Solidity code using the "solc" compiler version 5.0.

Luu..[44] propose a symbolic execution tool called OYENTE to find 4 kinds of potential security bugs from the above 12 bugs in SC Code. In their sample they discovered that 8833 out of 19,366 Ethereum smart contracts are vulnerable. We shall be using the same taxonomy classification of SC Vulnerability Code to analyse Ethereum API vulnerabilities in our study as it gives an in depth understanding of the code. The details of these 4 bugs are as follows:

(1) Transaction-ordering dependence. Valid transactions can change the state of Ethereum blockchain. In every epoch, each miner proposes their own block to update the blockchain. Since a block may contain multiple transactions, blockchain state may change multiple times within an epoch. When a new block contains two transactions which invoke the same smart contract, it may trigger this vulnerability. Because the execution of the smart contract is associated with state, the execution order affects the ultimate state.

(2) Timestamp dependence. In the blockchain, every block has a timestamp. Some smart contracts' trigger conditions depend on timestamp, which is set by the miner according to its local system time. If an attacker can modify it, timestamp-dependent contracts are vulnerable.

(3) Mishandled exceptions. This category of vulnerability may occur when different smart contracts are called from each other. When contract A calls contract B, if B runs abnormally, B will stop running and return false. In some invocations, contract A must explicitly check the return value to verify if the call has been executed properly. If A does not correctly check the exception information, it may be vulnerable.

(4) Reentrancy vulnerability. During the invocation of the smart contract, the actual state of the contract account is changed after the call is completed. An attacker can use the intermediate state to make repeated calls to the smart contract. If the invoked contract involves Ether transaction, it may result in illegal Ether stealing.

In this paper, we shall analyse the above SC Code vulnerabilities specific to the condition when API calling fails or returns errors in execution as this given area has been largely unexplored in past SC analysis, while at the same time the API-BC integration is exponentially increasing with added functionalities in BC dAPPs.

Literature review of Blockchain (SC) Vulnerability testing Tools:

Since programming bugs are universal in software application systems [21] [22], it advances the requirement for enhanced testing and debugging suites that are distinct to these languages. Several programming languages like Go, Solidity, and Ruby are increasingly used in BC based projects, require new gen testing tools that have not yet been standardized completely as yet. In contrast, undoubtedly, Java testing tools (i.e., provision for testing Java code via libraries and built-in tools) have experienced much more testing than Go lang, Ruby or Solidity.

A BC-App consists of various component layers, such as Smart Contracts (SCs), peer-to-peer networks, distributed systems, and consensus protocols. It will need a set of specialized testing tools to test each of these components individually. Also, as BC-App are distributed by definition, testing in insulation would need tools that suitably produce mocking objects proficiently simulating the BC. Therefore, some latest research in the field has provided hope with various open-source detection tools, such as Hyperledger caliper, OYENTE, EthereumTester, Exonum Testkit, and Embark, can be used for BC-Apps testing. The choice of tool(s) heavily depends on the target application, component and BC platform it uses. [25]

OYENTE Loi et al. [44] propose OYENTE to detect bugs in Ethereum smart ontracts. OYENTE leverages symbolic execution to analyze the bytecode of smart contracts and it follows the execution model of EVM. Since Ethereum stores the bytecode of smart contracts in its blockchain, OYENTE can be used to detect bugs in deployed contracts. Fig 4 shows OYENTE's architecture and execution process. It takes the smart contract's bytecode and Ethereum global state as inputs. Firstly, based on the bytecode, CFG BUILDER will statically build CFG (Control Flow Graph) of smart contract. Then, according to Ethereum state and CFG information, EXPLORER conducts simulated execution of smart contract leveraging static symbolic execution. In this process, CFG will be further enriched and improved because some jump targets are not constants; instead, they should be computed during symbolic execution. The CORE ANALYSIS module uses the related analysis algorithms to detect four different vulnerabilities (described earlier). The VALIDATOR module validates the detected vulnerabilities and vulnerable paths. Confirmed vulnerability and CFG information will finally be output to the VISUALIZER module, which can be employed by users to carry out debugging and program analysis. Currently, OYENTE is open source for public use [45].

Literature review of Testing Methods: Several theoretical frameworks un- derpin research on the automatic detection of API blockchain misuses. One prominent theory is the concept of behavioral analysis, which focuses on analyzing patterns and deviations in API usage to identify potential misuses (Chen et al., 2019). Another theory is based on anomaly detection, which involves detecting unusual or unexpected behaviors in API interactions that may indicate misuse or security threats (Basu et al., 2020).

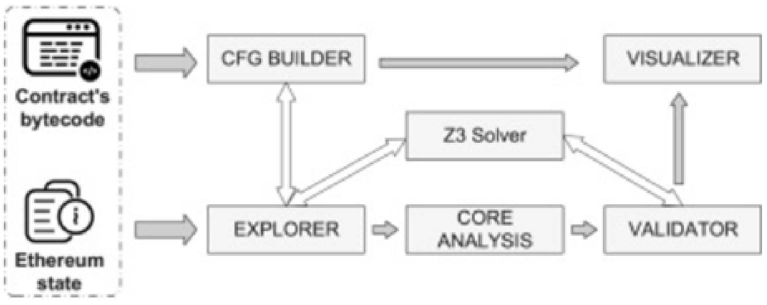


Figure 7: Enter Caption

Additionally, machine learning algorithms, such as supervised and unsupervised learning, are often employed to classify API calls and distinguish between normal and malicious behaviour. Static analysis techniques involve examining code artifacts, such as smart contracts and transaction data, to identify potential vulnerabilities or security loopholes (Conti et al., 2018). Dynamic analysis, on the other hand, involves monitoring API interactions in real-time to detect suspicious activities or deviations from expected behaviour (Swan, 2015). Hybrid approaches that combine static and dynamic analysis techniques offer a comprehensive solution by leveraging the benefits of both methods (Liu et al., 2021).

4 Research Methodology:

Applying Smart Contract Data-Sets to API Vulnerability Analysis

While direct datasets analysing and detecting for blockchain API vulnerabilities might be limited, smart contract datasets offer a valuable resource for understanding and analysing security issues in the blockchain space. By leveraging these SC datasets, one can draw parallels and develop effective methods for detecting and mitigating API vulnerabilities.

• **Overlapping Vulnerabilities:** Many security vulnerabilities in smart contracts also apply to blockchain APIs. For instance, Issues like re-entrancy, denial of service, and integer overflows are common to both. By studying how these vulnerabilities are exploited in smart contracts, you can gain insights into similar weaknesses in blockchain APIs.

• **Real-world Data**: Smart contract datasets are derived from real-world deployments, ensuring that the data is applicable and practical. This real-world relevance is crucial for developing robust vulnerability detection tools and methodologies.

• **Detailed Annotations**: These datasets often come with detailed annotations and categorizations of different vulnerabilities. This can help in training machine learning models and developing heuristic-based detection methods that could be adapted to blockchain APIs.

We thus analyse the common vulnerability patterns in smart contracts and adapt these patterns to the context of blockchain APIs. For instance, re-entrancy vulnerabilities in smart contracts might translate to improper handling of nested API calls.

XBlock-ETH: XBlock-ETH Dataset is an up-to-date on-chain data from Ethereum, which is one of the most popular permission-less blockchains and consists of transactions, smart contracts, and cryptocurrencies (i.e., tokens) and is itself derived from opensource Ethereum community platform available publicly. [50]

Why we chose the Xblock-ETH Dataset ?

• Rich Dataset extract the raw data consisting of 8,100,000 blocks from Ethereum.

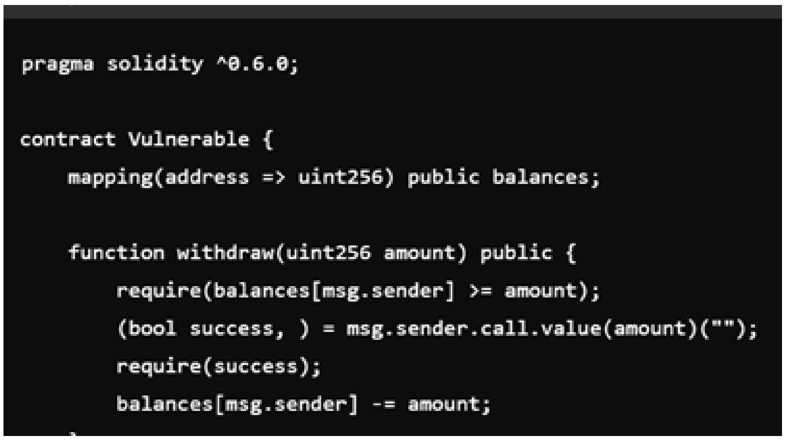
• Bulky size of Community Data: Difficulty in data synchronization at Blockchain peer due to the bulky size of the open source blockchain data from Ethereum community portal, takes a long period to fully synchronize entire blockchain data at a node newly connected with the BC. For example, it takes more than one week and over 500 GB storage space to fully synchronize the entire Ethereum at a peer. The high expenditure of massive storage space and network bandwidth due to blockchain data synchronization impedes the analysis of blockchain data.

• Heterogeneous and Complex Community Data: Open source Blockchain data is stored at clients in heterogeneous and complex data structures, which cannot be directly analyzed. Also, the underlying blockchain data is either binary or encrypted. Thus, it is a necessity to extract and process binary and encrypted blockchain data so as to obtain valuable information. However, it is non-trivial to process heterogeneous blockchain data since conventional data analytic methods may not work for this type of data. Contrarily, the XBlock – ETH has simplified and homogenized the community data by processing it manifold. [50]

Common API Vulnerabilities in Ethereum Smart Contracts

1. Reentrancy Attack A re-entrancy attack occurs when an external contract makes a recursive call back into the calling contract before the first invocation is complete. This can lead to multiple withdrawals before the contract state is updated.

Explanation: In this example, the contract sends Ether to the caller before updating the balance, allowing a reentrant call to withdraw funds multiple

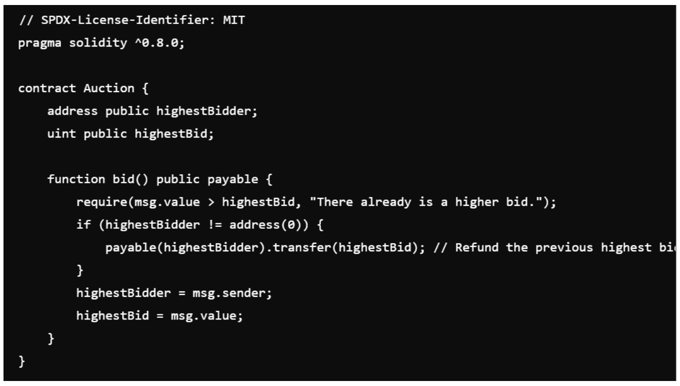


```
pragma solidity ^0.6.0;

contract Vulnerable {
    mapping(address => uint256) public balances;

    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount);
        (bool success, ) = msg.sender.call.value(amount)("");
        require(success);
        balances[msg.sender] -= amount;
```

Figure 8: Re-entrancy



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Auction {
    address public highestBidder;
    uint public highestBid;

    function bid() public payable {
        require(msg.value > highestBid, "There already is a higher bid.");
        if (highestBidder != address(0)) {
            payable(highestBidder).transfer(highestBid); // Refund the previous highest bi
        }
        highestBidder = msg.sender;
        highestBid = msg.value;
    }
}
```

Figure 9: Transaction Order Dependence

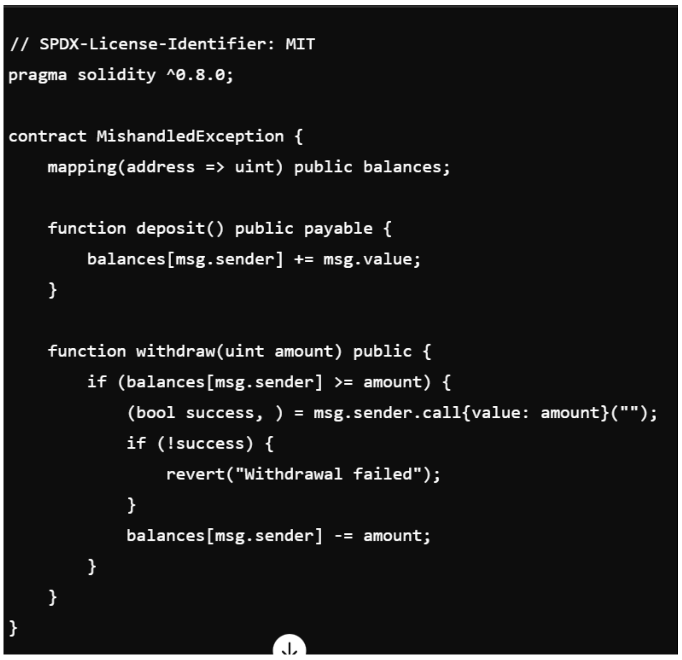
times.

2. Transaction Order Dependence rises when the outcome of a transaction depends on the order in which it is executed relative to other transactions. Attackers can exploit this to manipulate the sequence of transactions in their favor, leading to various types of malicious activities

3. Mishandled Exceptions may occur when different smart contracts are called from each other.

4. Timestamp Dependence Smart contracts that use block timestamps for critical operations can be manipulated by miners, leading to unexpected behaviors or exploits.

Explanation: The now keyword can be influenced by miners to extend or reduce the deadline, potentially leading to misuse.



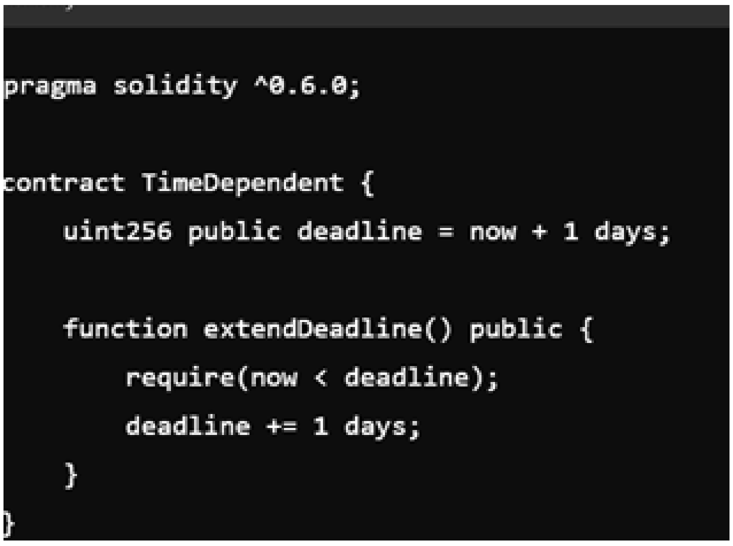
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MishandledException {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw(uint amount) public {
        if (balances[msg.sender] >= amount) {
            (bool success, ) = msg.sender.call{value: amount}("");
            if (!success) {
                revert("Withdrawal failed");
            }
            balances[msg.sender] -= amount;
        }
    }
}
```

Figure 10: Mishandled Exception



```
pragma solidity ^0.6.0;

contract TimeDependent {
    uint256 public deadline = now + 1 days;

    function extendDeadline() public {
        require(now < deadline);
        deadline += 1 days;
    }
}
```

Figure 11: Timestamp Dependence

4.1 Setting up Test Environment:

In this paper, we do not aim to provide a survey on functional and non-functional testing techniques of generic software systems that are not based on BC technologies, as there already exist many such studies done earlier. [17] [18] [19]. This section presents a detailed discussion on data collection by testing vital API Misuse components that are being used in different BC-Apps.

First, we start with the survey of existing tools and techniques available online or in market to be possibly used for discovering bugs and vulnerabilities in BC APIs. We searched in the online digital libraries (Git, ACM, etc) for state-of- the-art detection tools and zeroed in on open source Oyente that is compatible to test on Ethereum simulator platforms. (docker pull luongnguyen/oyente). They help to design and execute test scenario-cases with relevant test data and parameters. OYENTE would mimic the behavior and functionality of a real blockchain network, allowing API testing to be performed in a controlled and isolated environment without affecting the actual blockchain or consuming real resources. These tools work on the premise by typically starting with certain representations of misuse patterns, adopt different techniques to scan programs for pattern matches, and generate reports when matches are found. Additionally, blockchain test nodes Geth and Parity frameworks and libraries like **web3.js, Ether.js, Web3.py, solc 5.0** are installed and configured to create an automated test environment. This list was found when we searched for literatures published with keywords "se- cure API misuse Blockchain" in the ACM digital library. We attain a Benchmark Dataset, that label programs based on their cor- rect or incorrect usage of security APIs. They gather the vulnerabilities reported and provide an evaluation dataset for Application Security Testing tools. We may use XBlock-ETH data for this purpose that contains processed data from Ethereum platform dataset [5].

4.2 Factors to be considered during data Collection:

Implementing BC-based systems without paying special care to immutability carries a significant asset risk in research because BC transactions are irreversible, and not having adequate controls to avoid redundancy. The users must be aware of the possible output(s) of a transaction that they are performing. The SC developers could also place the required checks that invalidate a transaction if they try to invoke a wrong function in the SCs. To this end, we need to ensure that SCs are checked for a redundant transaction validation. Immutability is an inherent property of BC that ensures its data integrity and auditability. Moreover, it supports BC's secure and transparent nature. Immutability guarantees that the data stored on the BC ledger are tamper- proof (i.e., it cannot be removed or modified). Although this property is highly desirable for several applications and repudiates many privacy requirements and data protection rights when personal data is involved as a BC asset. Therefore, we as testers need to have adequate knowledge about the various data protection acts, and compliance testing needs to be carried out to ensure that the proposed BC implementation does not breach any of the privacy rights acts. This implies that immutability calls for an interdisciplinary approach to testing.

4.3 Project Timeline

It is imperative to have a carefully drafted plan and a dedicated strategy for testing with BC technolog because BC ecosystem's complexity can dynamically expand the scope of minimal set of tests (i.e., sanity check) to a significant margin.

- April 23 - May 7 (2 weeks)

  Research on the functioning of open source detector tools like MythX, Oyente, etc, and the functioning of Simulator/emulators for Blockchain-API technology to investigate the following:

    Compatibility of detector tools with the local API Blockchain Environment like JSON RPC, EOSIO SDK, etc. while checking for dependencies if any.

    JSON RPC, EOSIO SDKs, etc are Blockchain APIs used at industry level for third party tool integrations with blockchain technology.

    Check the Compatibility of the above detector tool to read successfully the misuse Data set.

    Identify the types of vulnerabilities to be considered (e.g., smart contract vulnerabilities, API misuses).

- May 8 - 15 (1 week)

  Continue writing and updating research proposal under supervisor's guidance with focus on detailed explanations.

- May 16 - 22 (1 weeks)

  Troubleshoot and Debug any dependencies that arise during the process which may include:

  > Creating needed libraries on Blockchain - API platform. Identify gaps in existing research related to automatic detection of vulnerabilities in blockchain APIs.

- May 22 - May 31 (1 week)

  Obtain a benchmark dataset like MUBench so that effective comparison can be made in future. This may take a buffer extra week as the API Blockchain field is relatively new and data procurement may be complex and diversely sourced.

- June 1 - June 21 (3 weeks)

  > Experiment Design: Design a series of experiments to evaluate the performance of selected tools in detecting vulnerabilities.
  >
  > Define the test scenarios, including various types of vulnerabilities, blockchain platforms, and API configurations.

- June 22 - 30 (1 week)

  Start Data testing and studying different types of vulnerabilities that occur in API Integration with blockchains like access point vulnerabilities, etc.

- July 1 - 21 (3 weeks)

  Data testing and Analysis:

  > Test the API Vulnerabilities using automated tools and Analyze the experimental results to evaluate the effectiveness and efficiency of the selected tools such as calculation of P Score, R value, etc. Identify trends, patterns, and correlations in the data. Interpret findings in relation to research objectives, benchmark data set and existing literature.

- July 22 - August 22 (4 weeks)

  > Conduct an online survey on Monkeysurvey.com using remote random survey samples in Information Blockchain industry experts after taking their due ethics consent. Budgetary details shall be finalised based on sample strength depending on respondents availibilitiy.

Report writing and presenting content conclusions/ generalisations in an interactive style that is standardised and ready for peer review.

- August 23 - September 14 (3 weeks)

  Prepare a detailed presentation summarizing the research study and its findings. Create visual aids, slides, and diagrams to enhance understanding and clarity. Practice delivering the presentation to ensure effective communication.

- Septembeer 15 - 30 (2 weeks)

  Release the initial report draft for peer review and take feedbacks so as to Address any feedback or revisions and an effective thesis defense is prepared.

5 Evaluation

5.1 Benchmarking:

We begin by defining clear metrics such as detection rate, false positive rate, and execution time. After we run the detector tools against a set of known vulnerabilities in controlled environments, we compare the results against industry standards or previous benchmarks to gauge the effectiveness of the tools. Benchmarking involves a structured approach to assess performance and efficacy of results. Finally, we document findings comprehensively, including any limitations or challenges encountered during the benchmarking process. Continuous refinement of benchmarks will aid in enhancing the reliability and relevance of vulnerability detection in blockchain APIs.

5.2 Metrics:

The effectiveness of detection tools in identifying and mitigating blockchain API misuses should be evaluated based on performance metrics such as precision, recall, and false positive rate. We used below metrics to measure tool effectiveness: precision, recall, F-score as shown below:

• Precision (P) measures among all reported misuses, how many of them are actual misuses (i.e., true positives). P = of true misuses detected / Total of detected misuses:

• Recall (R) measures among all known API misuses in benchmarks, how many of them are detected by a tool. R = of true misuses detected Total/ of known true misuses.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)}+\text{False Positives (FP)}}$$

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)}+\text{False Negatives (FN)}}$$

Figure 12:

• F-score (F) is the harmonic mean of P and R, to reflect a trade-off between the two metrics.

6 Conclusion:

Present possible directions to the OYENTE Solidity Community with a detailed empirical analysis of the tool with regard to Ethereum API-misuse detection so as to improve precision and recall values in future. This will not only pave way for tool upgradation but also enable SC developers to design robust and resilient BC API integration in future. Although the scope of the study is limited to specific code related API vulnerabilities in SCs, its results and approach can be adopted by future researchers to expand and cover other BC-API vulnerabilities as well.

7 References

1. https://academy.moralis.io/blog/what-is-a-blockchain-api

2. https://online.rmit.edu.au/blog/how-will-blockchain-affect-traditional-industries-australia

3. https://academy.moralis.io/blog/what-is-a-blockchain-api

4. https://owasp.org/www-project-top-ten/

5. "XBlock-ETH: Extracting and Exploring Blockchain Data From Ethereum," in IEEE Open Journal of the Computer Society, vol. 1, pp. 95-106, 2020, doi: 10.1109/OJCS.2020.2990458

6. Yuheng Huang, Haoyu Wang, Lei Wu, Gareth Tyson, Xiapu Luo, Run Zhang, Xuanzhe Liu, Gang Huang, and Xuxian Jiang. 2020. Understanding (Mis)Behavior on the EOSIO Blockchain. Proc. ACM Meas. Anal. Comput. Syst. 4, 2, Article 37 (June 2020), 28 pages. https://doi.org/10.1145/3392155.

7. P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, "Security analysis methods on ethereum smart contract vulnerabilities: A survey," 2019.

8. S. Amann, H. Nguyen, S. Nadi, T. Nguyen and M. Mezini, "A Systematic Evaluation of Static API-Misuse Detectors" in IEEE Transactions on Software Engineering, vol. 45, no. 12, pp. 1170-1188, 2019.doi: 10.1109/TSE.2018.2827384

9. Y. Zhang, M. M. A. Kabir, Y. Xiao, D. Yao and N. Meng, "Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?," in IEEE Transactions on Software Engineering, vol. 49, no. 1, pp. 288-303, 1 Jan. 2023, doi: 10.1109/TSE.2022.3150302.

10. Piccolboni, L., Guglielmo, G.D., Carloni, L.P., Sethumadhavan, S. (2020). CRYLOGGER: Detecting Crypto Misuses Dynamically. 2021 IEEE Symposium on Security and Privacy (SP), 1972-1989.

11. Arthur Gervais, Ghassan O. Karame, Karl Wu¨st, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). Association for Computing Machinery, New York, NY, USA, 3–16. https://doi.org/10.1145/29767

12. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204. Springer-Verlag, Berlin, Heidelberg, 164–186. https://doi.org/10.1007/978-3-662-54455-68

13. .Cao, X., Zhang, J., Wu, X. et al. A survey on security in consensus and smart contracts. Peer-to-Peer Netw. Appl. 15, 1008–1028 (2022). https://doi.org/10.1007/s12083-021-01268-

14. J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," IEEE Access, vol. 7, pp. 77 894–77 904, 2019

15. Cao, X., Zhang, J., Wu, X. et al. A survey on security in consensus and smart contracts. Peer-to-Peer Netw. Appl. 15, 1008–1028 (2022). https://doi.org/10.1007/s12083-021-01268-2

16. Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A Survey of Attacks on Ethereum Smart Contracts SoK. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204. Springer-Verlag, Berlin, Heidelberg, 164–186. https://doi.org/10.1007/978-3-662-54455-68

17. G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, andR. Hierons, "Smart contracts vulnerabilities: a call for blockchain software engineering?" in 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, pp. 19–25.

18. M. di Angelo and G. Salzer, "A survey of tools for analyzing Ethereum smart contracts," in 2019 IEEE Int. Conf. on Decentralized Applications and Infrastructures (DAPPCON), 2019, pp. 69–78.

19. C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, “Performance evaluation of blockchain systems: A systematic survey,” IEEE Access, vol. 8, pp. 126 927–126 950, 2020

20. Z. M. Jiang and A. E. Hassan, “A survey on load testing of large-scale software systems,” IEEE Transactions on Software Eng., vol. 41, no. 11, pp. 1091–1118, 2015.

21. J. Leng, M. Zhou, L. J. Zhao, Y. Huang, and Y. Bian, “Blockchain security: A survey of techniques and research directions,” IEEE Transactions on Services Computing, pp. 1–1, 2020.

22. I. Homoliak, S. Venugopalan, D. Reijsbergen, Q. Hum, R. Schumi, and P. Szalachowski, “The security reference architecture for blockchains: Towards a standardized model for studying vulnerabilities, threats, and defenses,” IEEE Communications Surveys Tutorials, 2020

23. Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” IEEE Transactions on Software Eng., vol. 37, no. 5, pp. 649–678, 2011.

24. A. A. Omar and F. A. Mohammed, “A survey of software functional testing methods,” SIGSOFT Softw. Eng. Notes, vol. 16, no. 2, p. 75–82, Apr. 1991.

25. Blockchain Testing: Challenges, Techniques, and Research Directions, Chhagan Lal and Dusica Marijan, 2021, eprint 2103.10074, arXi

26. H. Zhang, “On the distribution of software faults,” IEEE Transactions on Software Eng., vol. 34, no. 2, pp. 301–302, 2008.

27. G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, “On the distribution of bugs in the eclipse system,” IEEE Transactions on Software Eng., vol. 37, no. 6, pp. 872–877, 2011. Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, ”An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 2017, pp. 557-564, doi: 10.1109/BigDataCongress.2017.85.

28. Xiao Yi, Daoyuan Wu, Lingxiao Jiang, Yuzhou Fang, Kehuan Zhang, and Wei Zhang. 2022. An empirical study of blockchain system vulnerabilities: modules, types, and patterns. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022). Association for Computing Machinery, New York, NY, USA, 709–721. https://doi.org/10.1145/3540250.3549105

29. Christidis, K., Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. IEEE Access, 4, 2292-2303.

30. Conti, M., Kumar, E., Lal, C., Ruj, S., Vaudenay, S. (2018). A Survey on Security and Privacy Issues of Bitcoin. IEEE Communications Surveys Tutorials, 20(4), 3416-3452.

31. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P. (2016). A Secure Sharding Protocol for Open Blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 17-30).

32. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., Felten, E. W. (2015). Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. IEEE Symposium on Security and Privacy.

33. Zyskind, G., Nathan, O., Pentland, A. (2015). Decentralizing Privacy: Using Blockchain to Protect Personal Data. In Proceedings of the 2015 IEEE Security and Privacy Workshops (SPW) (pp. 180-184).

34. Sindre, G., Opdahl, A. L. (2011). Eliciting Security Requirements with misuse Cases. Requirements Engineering, 16(1), 47-70.

35. https://www.surveymonkey.com/?utsource = homepageutsource3 =heade

36. https://www.grayscale.com/research/reports/ethereums-coming-of-age-dencun-and-eth-2.0

37. Dean, 51

38. H. Mayer, Ecdsa security in bitcoin and ethereum: a research survey, 2016. URL http://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf.

39. Alharby, M., van Moorsel, A., 2017. Blockchain-based Smart Contracts: A Systematic Mapping Study. arXiv preprint...

40. Swan, 2015 M. Swan Blockchain: Blueprint for a New Economy, O'Reilly Media, Inc (2015)

41. Zheng Z., Xie S., Dai H.-N., Wang H. Blockchain challenges and opportunities: A survey Internat. J. Web Grid Serv. (2016)

42. L. Luu, Y. Velner, J. Teutsch, P. Saxena, Smart pool: Practical decentralized pooled mining, in: USENIX Security Symposium, 2017.

43. N. Atzei, M. Bartoletti, T. Cimoli, A survey of attacks on ethereum smart contracts (sok), in: International Conference on Principles of Security and Trust, 2017, pp. 164–186.

44. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: The 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254–269.

45. L. Luu, D. Chu, H. Olickel, P. Saxena, A. Hobor, Oyente: An analysis tool for smart contracts, 2016. URL https://www.comp.nus.edu.sg/ loiluu/oyente.html.

46. https://ieeexplore.ieee.org/abstract/document/8643084?casa$_token = TQrS3ZUTPLYAAAAA : Ck4N7lgZXpEvGPKRPfFOIe46kr8b5We75nvyhHExEZaKlRI$ $zd - n75TZ4$

47. https://www.sciencedirect.com/science/article/pii/S0167739X17318332?via=ihub

50. P. Zheng, Z. Zheng, J. Wu and H. -N. Dai, "XBlock-ETH: Extracting and Exploring Blockchain Data From Ethereum," in IEEE Open Journal of

the Computer Society, vol. 1, pp. 95-106, 2020, doi: 10.1109/OJCS.2020.2990458

51. H. Hasanova, U.-j. Baek, M.-g. Shin, K. Cho and M.-S. Kim, "A survey on blockchain cybersecurity vulnerabilities and possible countermeasures", International Journal of Network Management, vol. 29, pp. e2060, 2019.

52. A. K. J. C. M. A. A. Alkhalifah, A. Ng and P. Watters, "A taxonomy of blockchain threats and vulnerabilities", 2019.

53. H. Chen, M. Pendleton, L. Njilla and S. Xu, "A survey on ethereum systems security: Vulnerabilities attacks and defenses", ACM Comput. Surv., vol. 53, 2020.

54. N. F. A. L. P. J. P. E. B. W. Dingman, A. Cohen and L. Deng, "Defects and vulnerabilities in smart contracts a classification using the nist bugs framework", International Journal of Networked and Distributed Computing, vol. 7, pp. 121-132, 2019.

55. N. Atzei, M. Bartoletti and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)", Proceedings of the 6th International Conference on Principles of Security and Trust - Volume=10204, pp. 164-186, 2017.

56. Z. A. Khan and A. Siami Namin, "Ethereum Smart Contracts: Vulnerabilities and their Classifications," 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 2020, pp. 1-10, doi: 10.1109/BigData50022.2020.9439088.