This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

Digital Object Identifier

Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract

SATPAL SINGH KUSHWAHA[1], SANDEEP JOSHI[1],(Member, IEEE), DILBAG SINGH[2],(Member, IEEE), MANJIT KAUR[3],(Member, IEEE), and HEUNG-NO LEE[2], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Manipal University Jaipur, Jaipur-Ajmer Express Highway, Jaipur, India.
[2]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea.
[3]School of Engineering and Applied Sciences, Bennett University, Greater Noida, India.

Corresponding author: Heung-No Lee (e-mail: heungno@gist.ac.kr).

This work was partly supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2021-0-01835) supervised by the IITP(Institute for Information Communications Technology Planning Evaluation) and the National Research Foundation of Korea (NRF) Grant funded by the Korean government (MSIP) (NRF-2021R1A2B5B03002118)

ABSTRACT Blockchain is a revolutionary technology that enables users to communicate in a trust-less manner. It revolutionizes the modes of business between organizations without the need for a trusted third party. It is a distributed ledger technology based on a decentralized peer-to-peer (P2P) network. It enables users to store data globally on thousands of computers in an immutable format and empowers users to deploy small pieces of programs known as smart contracts. The blockchain-based smart contract enables auto enforcement of the agreed terms between two untrusted parties. There are several security vulnerabilities in Ethereum blockchain-based smart contracts, due to which sometimes it does not behave as intended. Because a smart contract can hold millions of dollars as cryptocurrency, so these security vulnerabilities can lead to disastrous losses. In this paper, a systematic review of the security vulnerabilities in the Ethereum blockchain is presented. The main objective is to discuss Ethereum smart contract security vulnerabilities, detection tools, real life attacks and preventive mechanisms. Comparisons are drawn among the Ethereum smart contract analysis tools by considering various features. From the extensive depth review, various issues associated with the Ethereum blockchain-based smart contract are highlighted. Finally, various future directions are also discussed in the field of the Ethereum blockchain-based smart contract that can help the researchers to set the directions for future research in this domain.

INDEX TERMS Blockchain, Smart Contract, Decentralized, Ethereum, Vulnerabilities, Security analysis tool.

I. INTRODUCTION

BLOCKCHAIN technology can be defined as an immutable, shared distributed ledger, spread over thousands of computer systems. Blockchain technology [1] gained implausible attention after the publication of Satoshi Nakamoto's [2] white paper in 2008. In the white paper, Satoshi gave the solution to the double-spending problem for digital currency in a decentralized P2P [3] network. Dubai appointed a minister in charge of Artificial Intelligence with a vision of becoming the world's first blockchain-powered government [4].

Saudi Arabia, one of the Middle Eastern countries, recently announced to use blockchain technology [5] for cred-

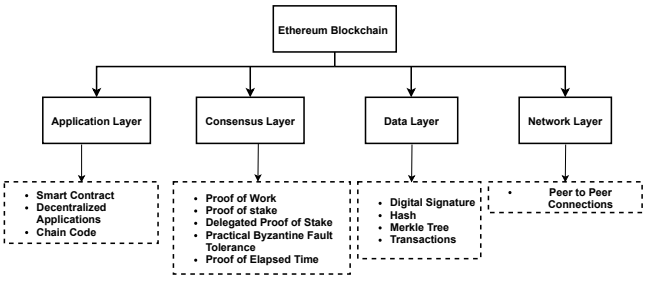

FIGURE 1. Layered structure of Ethereum blockchain.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

iting a part of the liquidity to be injected into the banking sector. Distributed consensus building is replacing the role of a trusted third party in decentralized P2P networks [6]. Ethereum is a most generally used Turing Complete [5] blockchain platform, which allows developers to write a smart contract with their own random rules for ownership, transaction format, and state transition functions. Anyone can run an Ethereum node [6] [7] on their machine to participate in the Ethereum blockchain network. Figure 1 describes the layered architecture of the Ethereum blockchain [8].

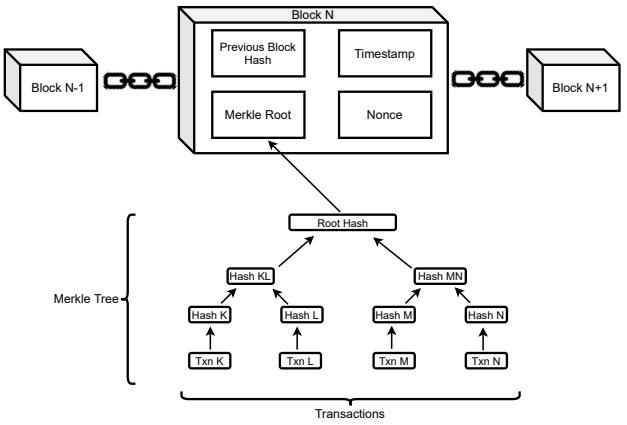

FIGURE 2. Structure of an Ethereum blockchain node.

Blockchain is a chain of immutable [9] blocks and this feature is increasing its popularity worldwide. Figure 2 explains the structure of an Ethereum blockchain node. Each node in the chain is linked to the other using the hash of the previous node. Transactions are hashed in the form of a Merkle tree, which is an important part of the blockchain. It is used for efficient and secure verification of consistency and integrity of large sets of transactional data. The use of Merkle trees in blockchain was introduced by W. Scott Stornetta and Stuart Haber in 1993 [10], but this data structure had already been patented by Ralph Merkle in 1979 [11].

A. MOTIVATION

The motivation for this survey is to serve developers, students, and researchers in the field of Ethereum smart contract security vulnerabilities. The existing surveys did not provide a systematic understanding of Ethereum security issues and their root causes. So, there is a strong requirement for a concise source of systematized information and best practices for understanding Ethereum smart contract security vulnerabilities, their root causes, sub-causes, detection tools, and preventive mechanism.

B. CONTRIBUTIONS

We perform a systematic survey of Ethereum smart contract security vulnerabilities, detection tools, real-life attacks, and preventive mechanisms over the period 2016 to 2021. Our work contributes to a detailed understanding of the Ethereum blockchain smart contract vulnerabilities with real-life attacks/examples. Our key contributions are as follows:

1) Ethereum smart contract vulnerabilities are categorized into three main root causes and seventeen sub-causes categories.
2) A deep insight into twenty-four Ethereum smart contract vulnerabilities with their preventive methods, detection, and analysis tools is provided under three root causes and seventeen sub-causes.
3) A brief comparison of detection and analysis tools for these vulnerabilities based on five parameters such as type of tool, input to the tool, type of analysis, implementation language, and availability of the tool to use is provided. This survey cannot be considered complete since Ethereum blockchain smart contract technology is constantly expanding at a very fast speed.

C. RELATED WORK

Recently, many researchers have published review papers in this field with different point of view. Li et al. [127] surveyed 20 types of vulnerabilities with attacks and defense mechanisms, without marking a difference in the type of blockchains like Bitcoin or Ethereum. Zhu et al. [128] reviewed 11 security vulnerabilities with attacks and defenses but their survey was limited to the bitcoin blockchain platform only. Saad et al. [129] surveyed about attacks and defenses but not about the vulnerabilities. They discussed attacks on the blockchain and their prevention mechanism. Harz et al. [130] discussed smart contract programming languages and their verification tools and methods but did not discuss more about security vulnerabilities. Angelo et al. [54] discussed about smart contract security analysis tools irrespective of their provenance. They discussed 27 tools for analyzing Ethereum smart contracts. Luu et al. [10] studied security vulnerabilities but did not discuss their detection and defense mechanisms. They presented a security analysis tool, named Oyente, which is based on formal verification. They analyzed Ethereum smart contracts for 4 to 5 security vulnerabilities using this tool. Atzei et al. [48] presented some of the security vulnerabilities and their related real-world attacks. They discussed the Ethereum smart contract security vulnerabilities in the context of common programming issues. Tang et al. [144] reviewed Ethereum smart contract vulnerabilities detection tools in three categories such as static analysis, dynamic analysis, and formal analysis. They considered 15 different security vulnerabilities and presented related detection tools. They suggested to use machine learning methods to analyze smart contracts. They discussed only 15 security vulnerabilities and missed several other important vulnerabilities. H. Chen et al. [145] discussed Ethereum smart contract vulnerabilities, their defenses, and attacks in relation with Ethereum smart contract architecture layers. They presented a good survey by giving good insight of each vulnerability but they did not cover the detection tool. T. Durieux et al. [145] discussed evaluation of 9 automated

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

analysis tools on 47587 Ethereum smart contracts. In their work they mainly discuss about tools comparison only.

The above-mentioned surveys did not cover all Ethereum smart contract security vulnerabilities and also some of the surveys missing details about detection tools and defense mechanisms. A survey can not be considered complete until it covers all the problems of their area. To fill this research gap, we present 24 security vulnerabilities with their detection tools, defense mechanisms, and real-time attacks.

D. PAPER OUTLINE

The remainder of this paper is structured as follows: Section II discusses the survey methodology, Section III reviews the key concepts, functionality, and structure of the Ethereum blockchain smart contract. Section IV presents a review of vulnerabilities in the Ethereum smart contract, categorized under three main root causes named Solidity programming language, EVM, and Ethereum blockchain design.These are further categorized into 17 sub-causes. This section briefly describes vulnerabilities with their preventive methods and detection tools against some well-known attacks. Section V gives a summary of Ethereum smart contract analysis tools to analyze the vulnerabilities based on the type of tools, input to the tool, type of analysis, implementation language, and availability of the tool to use. Section VI describes challenges and outlines future directions. Finally, Section VII concludes this survey by presenting a summary of the contributions.

II. SURVEY METHODOLOGY

In this section, various research questions and selection criteria of related articles are presented.

A. RESEARCH QUESTIONS

Several blockchain smart contract development platforms are available with different-different characteristics, opportunities, and challenges. Developers can select blockchain platforms as per their convenience from Ethereum [3], Hyperledger [12], Cardano, EOS [13], TRON [14], Steller, NEO, Waves, Tezos [15], NEM, etc. Ethereum is the most used prominent blockchain platform, so in this paper, we surveyed the vulnerabilities present in the Ethereum smart contract. The literature deficiencies are a definite and organized review of the current Ethereum blockchain smart contract vulnerabilities. It was the main reason for conducting this research. Systematic study methods of Kitchenham et al. [16] and Peterson et al. [17] are followed to define the following research questions:

- RQ1: What are the existing vulnerabilities in Ethereum Smart Contracts?
- RQ2: What are the main root causes of vulnerabilities in Ethereum Smart Contracts?
- RQ3: What are the sub-causes of vulnerabilities in Ethereum Smart Contracts?
- RQ4: What are the detection tools of vulnerabilities in Ethereum Smart Contracts?

B. PROCESS ADOPTED FOR THE SELECTION OF RELATED ARTICLES

To address these questions, 454 research articles are identified from peer-reviewed scientific research databases. Out of these, 335 articles are excluded and 119 articles are included for the final survey based on the exclusion and inclusion criteria mentioned in Table 1.

TABLE 1. Literature articles inclusion/exclusion criteria

Exclusion criteria	Inclusion Criteria
Studies not associated with Ethereum	Studies associated with Ethereum blockchain
Studies not associated with smart contract	Studies associated with Ethereum smart contract
Studies authored non-English	Studies related to Ethereum smart contract vulnerabilities
Doctrate thesis	Studies related to Ethereum smart contract vulnerability detection tools
Related to another smart contract platform	Studies related to Ethereum smart contract analyzers
Keynote articles	Studies related to Ethereum smart contract security, reliability, and optimization

We applied keyword searching to find Ethereum smart contract security vulnerability-related articles for our initial search. We performed this search in five peer-reviewed scientific databases like Elsevier Science Direct, IEEE Xplore Digital Library, Springer Online Library, ACM Digital Library, and Google Scholar. Our main focus is on Ethereum smart contracts, so we selected articles containing Ethereum in their abstracts. Literature articles identification, exclusion, eligibility, and inclusion methodology are shown in Figure 3.

We selected articles from conference proceedings, journal articles, repositories, and others like book chapters, etc. Selected articles bifurcation is shown in Figure 4 as per year of publication and scientific database.

III. ETHEREUM BLOCKCHAIN SMART CONTRACT

Smart Contract [18] is one of the use cases of blockchain technology. It was introduced by Nick Szabo [19] in 1997. It is a small piece of self-executable [20] program code. It is deployed on the blockchain and used for auto enforcement of terms and conditions between two untrusted parties. Consensus protocols [5] [21] are used for the precise implementation of smart contracts, otherwise, the effect of the same will be nullified. Figure 5 elaborates the structure of the Ethereum Smart Contract [12] [24].

Ethereum smart contracts' main components are functions, events, and state variables written in solidity [3]. Solidity programming language's Turing completeness [5] makes it perfect for writing a smart contract. After compilation, the smart contract code is converted into EVM byte code [23] [24] and stored on Ethereum blockchain by a contract creation transaction. After a successful contract creation transaction, the smart contract is recognized by a unique contract address.

An Ethereum Smart Contract account consists of [25] its executable code, contract address, a state consisting of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

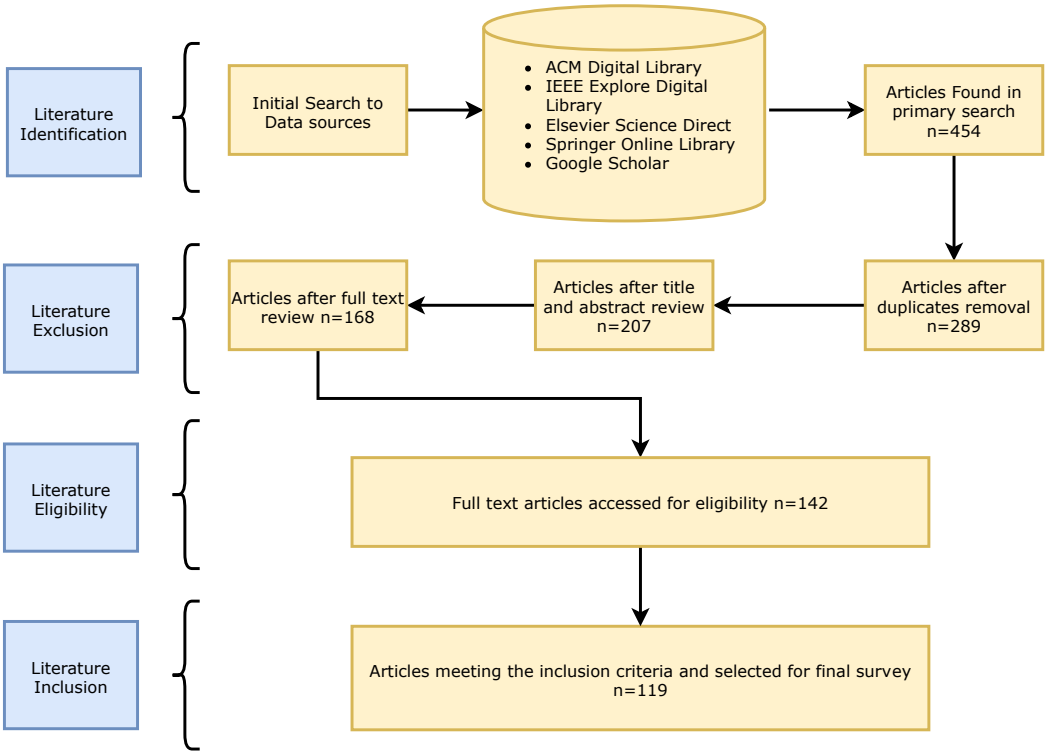

FIGURE 3. Literature articles identification, exclusion, eligibility, and inclusion methodology..

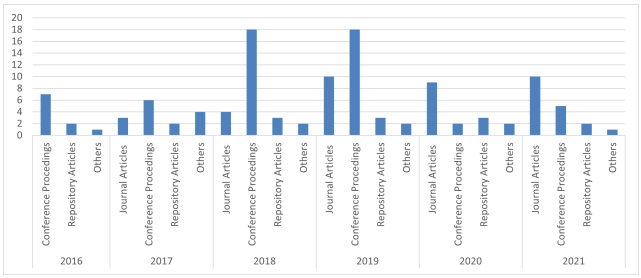

FIGURE 4. Selected research articles bifurcation according to the year of publication and scientific database.

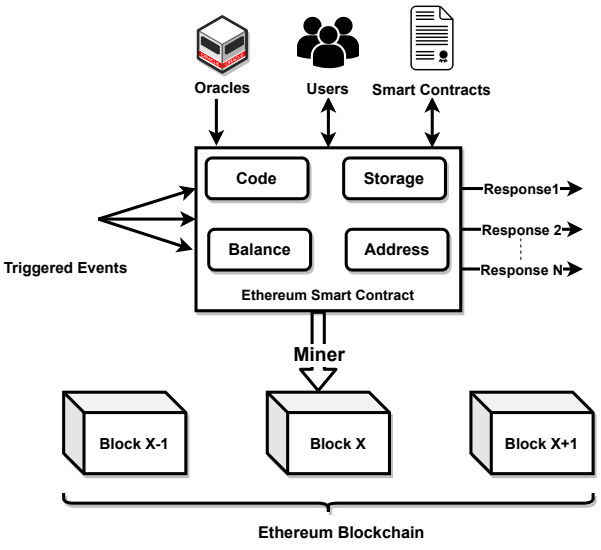

FIGURE 5. Structure of Ethereum smart contract.

private storage, and balance in terms of virtual coins (Ether). A smart contract can be invoked using a contract invoking transactions to its inimitable address, with some parameters like invocation data and payment in terms of ether as transaction fees (Gas) [26]. Ethereum Virtual Machine or EVM [27] is a Turing complete stack-based virtual machine. It provides a run-time environment that is isolated from the network to execute the smart contract code. Computation in EVM [28] is essentially bounded by transaction fees. Smart contracts contain ethers (a type of cryptocurrency) as their balance. Ethers can be sent to other contracts for executing the smart contract. It is a very crucial task because any type of vulnerability can be the reason for the loss of millions of Ether [28]. A smart contract cannot be changed after the deployment [29] on the blockchain due to the immutable [30] nature of the blockchain. Therefore, security vulnerabilities and best practices should be considered by the smart contract developers during the coding of smart contracts.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

IV. ETHEREUM SMART CONTRACT VULNERABILITIES AND PREVENTIVE METHODS

In this section, we have reviewed Ethereum Smart Contract vulnerabilities [9], [31]–[45], [122], [123], [131], [132], [138]–[142] with some well-known attacks/example [32], [46]–[49], detection tools [30], [37], [50]–[58], [125], [133]–[137], [143], and their suggested preventive methods [30], [47], [59]–[63], [124], [126]. Researchers classified these vulnerabilities based on different criteria such as seriousness, root cause, flaws in solidity, security flaws, privacy flaws, performance flaws, flaws in EVM [27] byte code, and blockchain [64] characteristics. We have divided these vulnerabilities under the following three main root causes:

- Root Cause 1: Solidity Programming Language (SPL)
- Root Cause 2: Features of Ethereum Virtual Machine (EVM)
- Root Cause 3: Design features of Ethereum Blockchain (EBD)

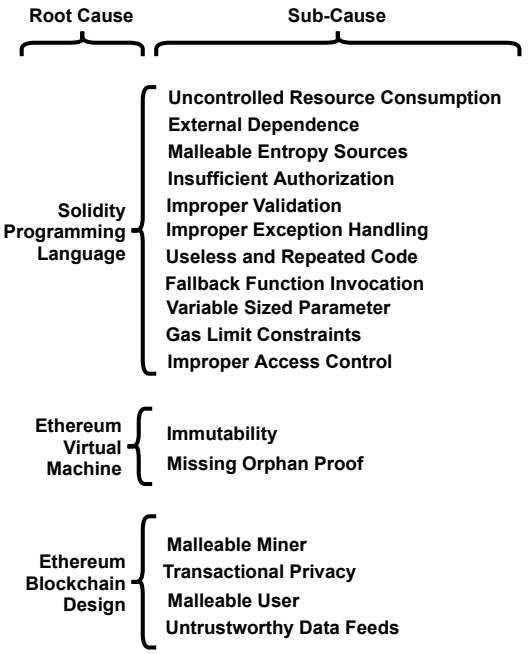

FIGURE 6. Root cause and sub-cause categorization of Ethereum smart contract vulnerabilities.

We further categorize these vulnerabilities in sub-cause categories, which are highlighted in Figure 6.

For ease of use and proper understanding, Figure 7 explains the nomenclature used to denote the vulnerabilities. Vulnerabilities in each root cause category are numbered from 1 to N (N is the total number of vulnerabilities in that category), for example, SPL - V1 is the first vulnerability in Root Cause "Solidity programming language" and so on.

Figure 8 elaborates the Ethereum smart contract vulnerabilities, numbered starting from 1 to the total number of vulnerabilities in the respective root cause category.

In the following section, we present each vulnerability in detail according to its root cause, sub-cause, detection

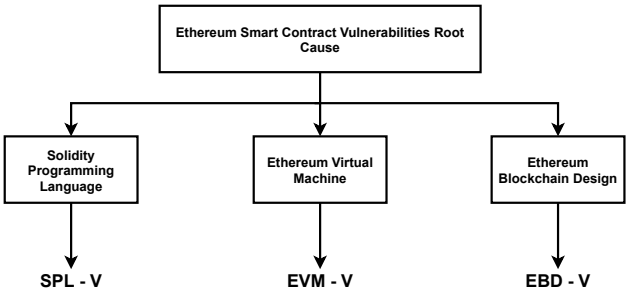

FIGURE 7. Nomenclature to denote vulnerabilities.

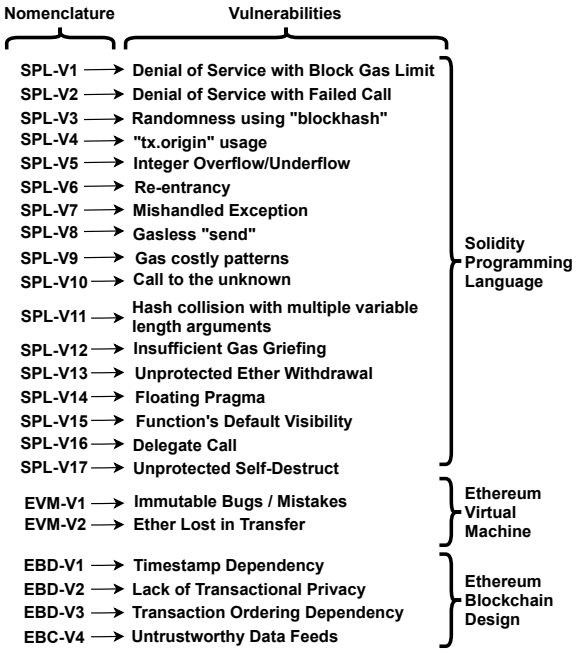

FIGURE 8. Classification of Ethereum smart contract vulnerabilities and their root causes

tool, and its prevention mechanism. Tables 2 to 24 highlights summarized details like sub-cause, attacks, detection tools related to each vulnerability as mentioned in Figure 8.

SPL -V1 (Denial of service with block gas limit) [26]: Each block in the Ethereum smart contract has an upper limit on the amount of gas (10,000,000 gas [67]) that can be spent for computation. Thus, if the gas spent on any transaction exceeds this block gas limit, then this leads to denial of service and the transaction will fail. This is the case with the unknown size of arrays which grow over time.

Preventive method: Rather than modifying these arrays completely at once in a single block, several blocks should be taken. Because taking several blocks will break a single transaction into multiple transactions, hence it reduces the gas required for each transaction. This will reduce the risk of exceeding the block gas limit, and hence prevent denial of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 2. SPL-V1 Sub-cause, attack, and detection tools

Sub-Cause	Uncontrolled Resource Consumption
Attack/Example	Block Stuffing Attack [68]
Detection Tool	SmartCheck [69]

TABLE 3. SPL-V2 Sub-cause, attack, and detection tools

Sub-Cause	External Dependence [70]
Attack/Example	Ponzi GovernMental Jackpot [70]
Detection Tool	SmartCheck [69]

service.

SPL -V2 (Denial of service with failed call) [70]: External calls in a smart contract can fail accidentally (due to programming errors) or intentionally (by an attacker). In a situation when an attacker combines several calls in a single transaction and executes in a loop, then this can prevent other smart contracts nodes to interact with it. This scenario is shown in the Figure 9 solidity code.

Preventive method: The contract logic to handle failed calls should be executed in such a way that several calls of Ether transfer must not be combined in a single transaction with the assumption that the external calls will always fail to choose 'pull' over 'push' for external calls. It should implement contract logic to handle failed calls.

SPL-V3 (Randomness using 'block hash') [9], [46]: Sometimes randomness is required in operations and block hash can be used for this purpose. However, it can be manipulated like timestamp because the same can be predictable by the miners. The use of 'block hash' as a source of randomness is shown in Figure 10 solidity code.

Preventive method: The block hash can be read by any other transaction within the same mined block. If the attacker is also a miner, then it can be manipulated and worse can happen. Therefore, the external sources of randomness should be used, for example, Oracle, RNG [61], RANDAO, etc.

DOS_with_Failed_Call.sol

```
contract auction {
  address highestBidder;
  uint highestBid;

  function bid() payable {
     require(msg.value >=highestBid);

     if(highestBidder != address(0)) {
        (bool success, ) = highestBidder.call.value(highestBid)("");
        require(success); // if this call consistently fails, no one else can bid,
                          // hence results in DOS_with_Failed_call
     }
     highestBidder = msg.sender;
     highestBid = msg.value;
  }
}
```

FIGURE 9. Denial of service for failed call.

TABLE 4. SPL-V3 Sub-cause, attack, and detection tools

Sub-Cause	Malleable Entropy Sources
Attack/Example	Gambling or Lottery Contracts [8]
Detection Tool	Oyente [10], Securify [71], SmartCheck [69]

Blockhash.sol

```
function randomNumber() internal view returns (uint)
{
   return uint(blockhas());
}
```

FIGURE 10. 'Block hash' as a source of randomness.

SPL -V4 ('tx.origin' usage) [9]: This global variable of solidity returns the address of the transaction initiator, which in turn attracts the adversaries to imitate the initiator that makes the contract vulnerable. The victims of this attack are wallet-type smart contracts.

Preventive method: For security reasons, the origin address should not be passed as a parameter, because an attacker can easily figure out the right address to perform spoofing. Therefore, it is suggested to use 'msg.sender' in place of 'tx.origin' to check the authorization of ownership, because tx.origin is the original sender of the transaction but msg.sender is the immediate sender. One more point to consider is to not use address.call.value(amount)(); instead, use address.transfer(). Because address.transfer throws an exception in case of failure and stipend requirement for executing address. The transfer is 2300 ether, which means assaulting contracts would not have enough gas for additional calculation.

SPL -V5 (Integer overflow/underflow) [47] [72]: This kind of situation will occur when the value in a calculation exceeds the lower or upper range of the variable size type, so cannot be expressed by that type. This can be the case with smaller data types. If the balance will be at the upper limit uint value (2^{256}), then it will reset again to zero. An unknown hacker drained off 2000 ether (cryptocurrency coin of the Ethereum blockchain) which was worth $2.3 million using this vulnerability from "Proof of Weak Hands Coin" (PoWHC), which was a legit Ponzi [70] scheme smart contract.

Preventive method: To tackle integer overflow and underflow issues, arithmetic operations should be implemented very carefully by comparing the operators and operands before the operation. It is suggested to use assert(), require()

TABLE 5. SPL-V4 Sub-cause, attack, and detection tools

Sub-Cause	Insufficient Authorization
Attack/Example	Unknown
Detection Tool	Oyente, Securify, SmartCheck, Remix, Mythril [72]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 6. SPL-V5 Sub-cause, attack, and detection tools

Sub-Cause	Improper Validation [66]
Attack/Example	Proof of Weak Hand Coin (PoWHC) [74]
Detection Tool	SmartCheck [69], EASYFLOW [73] [72]

TABLE 7. SPL-V6 Sub-cause, attack, and detection tools

Sub-Cause	External Dependence
Attack/Example	DAO Attack [79]
Detection Tool	Oyente, Securify, SmartCheck, Remix, Mythril

functions, and 'SafeMath.sol' [38] library for arithmetic functions.

SPL-V6 (Re-entrancy) [74], [76], [77]: It is also known as a recursive call attack. In re-entrance vulnerability, a malicious contract calls back into the calling contract before the first invocation of the function is finished. Therefore, due to this recursive nature of the call, the malicious user can do multiple repetitive withdrawals without affecting his balance. An attacker stole 60 million US dollars by utilizing this vulnerability in the Decentralized Autonomous Organizations (DAO).

Preventive method: For protecting against re-entrancy attacks, a reliable method can be used such as the "Check-Effect-Interaction pattern". This defines the order in which we should structure our functions to assure all internal state changes before the next call. After resolving all state changes, the function should be allowed to interact with other contracts. To prevent cross-function re-entrance attacks, mutex locks are suggested to use. Any contract's state can be locked using a mutex lock and can only be modified by the owner of the lock. It prevents the recursive call of 'withdraw function' by an attacker. OpenZeppelin has its mutex implementation called as Re-entrance guard which acts as a re-entrancy lock.

SPL -V7 (Mishandled exception) [79]: When a contract is called by another contract and an exception or error is raised in the Calle contract. But if the same might not be reported to the Calle contract, then it might lead to threats. Therefore, it can attract the attackers to execute the malicious code in the contract.

Preventive method: An exception in the callee may or may not be propagated to the caller depending on how the function was called. Exception handling operators should be used to avoid such types of situations. The return value of functions must always be checked and an exception should be thrown.

SPL -V8 (Gasless 'send') [26]: Gas is required in Ethereum smart contract as an execution fee. When ethers are transferred to a smart contract via send and if the fallback

TABLE 8. SPL-V7 Sub-cause, attack, and detection tools

Sub-Cause	Improper Exception Handling
Attack/Example	DAO Attack [79]
Detection Tool	Oyente, Securify, SmartCheck, Remix

TABLE 9. SPL-V8 Sub-cause, attack, and detection tools

Sub-Cause	Improper Exception Handling
Attack/Example	King of the Ether Throne [82]
Detection Tool	VULTRON [83]

TABLE 10. SPL-V9 Sub-cause, attack, and detection tools

Sub-Cause	Useless and Repeated Code
Attack/Example	FirstContract, Ballot
Detection Tool	Oyente, Remix, SmartCheck, Gasper

function in the Calle contract has the maximum gas limit [80] of twenty-three hundred units, then an out-of-gas exception can be thrown. If this exception is not handled properly, then malicious users can take benefit from this vulnerability by keeping non-transferred ethers wrongfully even though it was assumed to give them away. This scenario can be understood in Figure 11. If the pay function of contract A is called with the address of contract B. Then contract A is supposed to send 'X' wei (the smallest denomination of ether) to contract B. If the fallback function of contract B revert due to insufficient gas and the contract behaves maliciously then it will keep those 'X' wei to itself.

```
contract A {
   function pay(uint n, address B) {

      B.send(X);
  }
 }


 contract B {
    uint public count = 0;

     function() {
      count++;
  }
 }
```

FIGURE 11. An Example of Gasless 'send'

Preventive method: Gas consumption limits change regularly because of the change in the transaction fees. Therefore, an exception should be thrown properly if failure happens due to gas consumption. The fallback functions should be designed in such a way that they do not require too much gas for execution or are less expensive to execute. It will mitigate the purpose of failing and the cost of executing the contract.

SPL -V9 (Gas costly pattern) [79] [83]: Every instruction in Ethereum smart contract requires gas (Ethers) as execution fee. Under-optimized [40] smart contracts can contain unnecessary and irrelevant code or expensive patterns, which can cost a lot of execution fees. Chen et al. [42] identified seven programming patterns and developed a tool 'GASPER' to determine such types of patterns.

Preventive method: Chen et. al [42] developed a tool

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 11. SPL-V10 Sub-cause, attack, and detection tools

Sub-Cause	Fallback function invocations
Attack/Example	The DAO
Detection Tool	Mythril [72]

TABLE 12. Sol-V11 Sub-cause, attack, and detection tools

Sub-Cause	Variable Sized Parameters
Attack/Example	Hash Collision Attack
Detection Tool	VerX [85]

TABLE 13. SPL-V12 Sub-cause, attack, and detection tools

Sub-Cause	Gas Limit Constraints [81]
Attack/Example	Griefing Attack [88]
Detection Tool	MadMax [89]

TABLE 14. SPL-V13 Sub-cause, attack, and detection tools

Sub-Cause	Improper Access Controls
Attack/Example	Improper Access Controls
Detection Tool	MythX [91], Mythril [72]

named 'GASPER', which performs the following operations to detect gas costly patterns:

- Do a depth-first search and detect the dead code by scanning the control flow graph.
- Detects the opaque predicates by executing the smart contract symbolically.
- Detects expensive loop operations like SLOAD, SSTORE, and BALANCE.

These types of gas costly patterns should be identified and should be replaced by gas efficient patterns for optimizing the execution cost.

SPL -V10 (Call to the unknown) [54]: The cause of this vulnerability is the execution of the fallback function under certain conditions. The function with the given signature does not exist. This can happen due to the invocation of certain functions of Calle's contract or the transfer of ether to another smart contract.

Preventive method: Call to unknown contracts can be very risky, so the following points should be considered:

- Be vigilant while making external calls
- External calls to unknown addresses should be performed as the last operation in a localized function or piece of code execution.
- Unknown contracts should be marked unsafe
- Use checks effect interaction pattern to avoid state changes after an external call

SPL -V11 (Hash collision with multiple variable-length arguments): ' abi.encodePacked()' is a nonstandard packed mode that is mainly used for indexed packed parameters. 'abi.encodePacked()' can lead to a hash collision in some specific situations due to multiple variable-sized parameters. 'abi.encodePacked()' packs elements in order regardless of their original order in the array, which can be exploited by the attacker in signature verification by changing the position of the elements.

Preventive method: The following points should be considered to prevent this vulnerability:

- Rather than passing arrays as parameters, pass a single value.
- Array's positions are modifiable, so allow arrays of fixed length only.
- Use abi.encode() instead of abi.encodePacked()

SPL -V12 (Insufficient gas griefing) [9]: Smart contracts that accept data from external sources and use the same in a subcall to another contract, are vulnerable to insufficient gas griefing attacks. Because in case of subcall failure [85], the whole transaction can be reverted. The attacker can effectively censor transactions by using just enough gas [86] to execute the transaction, but not enough for the sub-call to succeed. This scenario is shown in Figure 12 solidity code.

Preventive method: To prevent this vulnerability, trusted users should be allowed to relay transactions. A type of logic should be applied to check whether a sufficient amount of gas is provided or not. Relayers can be encouraged by a reward to give the right amount of gas for a successful transaction.

Insufficient_Gas_Griefing.sol

```
contract broadcast  {

  mapping(bytes => bool) executed;


  function repeat(bytes_data) public {

     //Repeat protection, do not call the same transaction twice

     require (external[_data] == 0,"Repeated Call");

     executed[_data] == true;

     innerContract.call(bytes4(keccak256("execute(bytes)")),_data);

}
}
```

FIGURE 12. Insufficient gas griefing.

SPL -V13 (Unprotected ether withdrawal): Ether can be withdrawn by attackers due to missing or inadequate access control. Improper naming of functions intended to be a constructor and constructor code can be accessed by malicious users to reinitialize the contract.

Preventive method: Access control should be implemented in such a way that only authorized users can access the same. The function's naming should be done very carefully so that the function's name should not be intended to be a constructor. Because the constructor code, which will be ended up in the run time, byte code can be called by anyone to initialize the contract again.

SPL-V14 (Floating pragma) [91]: Different compiler versions can be used to compile the solidity smart contract. But compilation with an outdated compiler might introduce a bug and can affect the smart contract system. At the time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 15. SPL-V14 Sub-cause, attack, and detection tools

Sub-Cause	Improper Access Controls
Attack/Example	Unknown
Detection Tool	SolidityCheck [11]

TABLE 16. SPL-V15 Sub-cause, attack, and detection tools

Sub-Cause	Improper Access Control
Attack/Example	Parity Multi-signature Parity attack [48]
Detection Tool	ContractGuard [53]

TABLE 17. SPL-V16 Sub-cause, attack, and detection tools

Sub-Cause	External Dependence
Attack/Example	Parity Multi-signature Parity attack [48]
Detection Tool	ContractFuzzer [63]

TABLE 18. SPL-V17 Sub-cause, attack, and detection tools

Sub-Cause	Improper Access Control
Attack/Example	Parity Multi-signature Parity attack
Detection Tool	MythX [91]

of deployment, the compiler version of the smart contract should be the same as that testing time.

Preventive method: Smart contracts can be compiled using different compiler versions. But if an older version of the solidity compiler is used to compile the smart contract, leads to errors in the form of bugs. A smart contract must be compiled on the same solidity compiler on which it has already been tested during the creation phase. Before deployment on the blockchain, the compiler version mentioned in the solid source code of the smart contract must be ensured for similarity and as a preventive technique Pragma version should be locked before deployment

SPL -V15 (Function's default visibility) [9], [46]: In solidity, visibility specifiers are used to make functions external, public, internal, or private. Incorrect use of these visibility specifiers can make smart contracts vulnerable and attract attackers.

Preventive Method: It is an acceptable practice to consistently determine the visibility of all functions in a contract, regardless of whether they are intentionally public. Recent versions of Solidity presently show warnings during the compilation of functions that have no explicit visibility set, to help empower this training. So as a preventive method, special attention is required while using them.

SPL-V16 (Delegate Call) [92]: For calling another contract's function, the target function's ABI (Application Binary Interface - the standard way of interaction with smart contracts in the Ethereum blockchain environment) is required. If ABI is known then the target function's signature can be used directly for calling purposes. But what if we don't know the target function's ABI, then we can use the "DELEGATECALL". For example, suppose there is contract A and contract B. The contract A delegate call to contract B's function. If the user calls contract A then contract A delegate call to contract B and the function would be executed. But in this scenario, all the state changes will be reflected in contract A, not in contract B. The context preserving nature of 'DELEGATECALL' can create vulnerabilities when it runs in the context of other applications. It can lead to unexpected execution such as self-destruction and balance loss of contracts.

Preventive method: "Library" keyword is provided in the solidity programming language to execute library contracts. This guarantees that the library contract is stateless and non-self-destructible. Stateless libraries help in mitigating several issues related to storage context. Stateless libraries additionally forestall assaults whereby aggressors change the condition of the library straightforwardly to influence the contracts that rely upon the library's code. So as a preventive method, more attention must be paid while utilizing "DELEGATECALL" especially in the calling context of calling contract and library contract. For ensuring the safety of storage context, Delegate calls must be used to call trusted contracts only.

SPL-V17 Unprotected 'self-destruct': This vulnerability is named because of the self-destruct opcode. Before deleting the contract permanently, this opcode can transfer the ethers of the contract to a predefined contract. If the access control is not given properly, the malicious user can destroy the contract to snip all ethers of the contract.

Preventive method: This vulnerability regularly emerges from the misuse of 'this.balance'. Whenever we send ether to any contract then called contracts fallback function is called. But the fallback function is not called if the ether transfer is initiated using "selfdestruct()" function call. Self-destruct functionality should be removed from the smart contract unless it is compulsorily required. And if there is a need to use the functionality of "selfdestruct()" then it must be used in the multi-signature scheme environment. Because in multi-signature environment, the execution of "selfdestruct()" must be approved by multiple parties.

EVM-V1 (Immutable bugs or mistakes) [9] [46]: It is related to the immutability of the Ethereum blockchain. Because once a contract is deployed on the blockchain, it cannot be altered due to the immutability feature of blockchain. It is against the legal law that allows being modified and terminated. However, it may lead to problems if the deployed smart contract contains bugs because after deployment it is impossible to alter.

Preventive Method: Immutability of Ethereum smart contracts creates several issues. Marino et al. [94] tackled this vulnerability by presenting a set of standards taken from legal contracts and redefined to be fit in the context of Ethereum smart contracts. This set of standards enable Ethereum smart

TABLE 19. EVM-V1 Sub-cause, attack, and detection tools

Sub-Cause	Immutability [32]
Attack/Example	The DAO attack
Detection Tool	Securify [71], EVMFuzzer [94]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 20. EVM-V2 Sub-cause, attack, and detection tools

Sub-Cause	Missing Orphan Proof
Attack/Example	Unknown
Detection Tool	Securify, SmartCheck, EVMFuzzer [94]

TABLE 21. EBD-V1 Sub-cause, attack, and detection tools

Sub-Cause	Malleable Miner
Attack/Example	GovernMental Attack
Detection Tool	Oyente, Remix, SmartCheck, Mythril

contracts to be changed or terminated whenever required as per the need. Marino et al. demonstrated their success by applying this set of standards in the scenario of the Ethereum smart contract.

EVM-V2 (Ether lost in transfer) [31]: Ethers can be transferred to a contract to its unique address, but the contract must not be orphan, because in that case the transferred ether will be lost forever. There is a mechanism to revert the ether transfer.

Preventive method: Since the lost ether cannot be recuperated, developers need to physically guarantee the rightness of the recipient addresses. The address of the recipient contract must be verified manually to avoid the loss of ether in the transfer.

EBD-V1 (Timestamp dependency) [9]: It is classified as a security vulnerability by Maher and Alharby [96]. It leads to a vulnerable situation when the triggered condition to execute the transaction is the block timestamp, because the dishonest miners can utilize the block timestamp value in an unethical way.

Preventive method: Luu et al. [8] suggested to use block number instead of block timestamp. Because the block number cannot be altered by the malicious miner. Therefore, it is suggested not to assign a block timestamp to a variable in the smart contract code.

EBD-V2 (Lack of transactional privacy) [96] [97]: Transactions balance details of the users are publicly available. But, users want that their financial details and transactions should not be visible to others. It may limit the users of smart contracts since attackers can monitor the transaction-related details of users. Attackers can use this information for various kinds of unethical uses.

Preventive method: Watanabe et al. [21] suggested the encryption of the smart contracts before deploying them on the blockchain. Kosba et al. [97] developed a tool to create a privacy-preserving contract. They included the important features of privacy protection not only in Ethereum blockchain-based smart contract applications but also for all types of blockchains. As a preventive method to this issue,

TABLE 22. EBD-V2 Sub-cause, attack, and detection tools

Sub-Cause	Transactional Privacy
Attack/Example	Zerocash Cryptocurrency [99]
Detection Tool	Hawk [96]

TABLE 23. EBD-V3 Sub-cause, attack, and detection tools

Sub-Cause	Malleable User
Attack/Example	GovernMental Attack
Detection Tool	Oyente, Securify, Mythril

TABLE 24. EBD-V4 Sub-cause, attack, and detection tools

Sub-Cause	Untrustworthy Data Sources
Attack/Example	Adversaries attack through Oracle
Detection Tool	Town Crier [101]

they implemented a decentralized smart contract framework, Hawk, which does not store financial transactions in the blockchain and spares the developers from implementing any cryptographic function.

EBD-V3 (Transaction ordering dependency) [54]: This vulnerability is related to the execution order of two dependant transactions that are invoking the same smart contract. A malevolent user can utilize this vulnerability to attack if the transactions are not executed in the proper order. The order of transaction execution is decided by the miners, but if the adversary is the miner itself, then this will be a very disastrous situation.

Preventive method: Natoli et al. [100] used Ethereum-based functions (e.g., SendIfReceived) to enforce the order of transactions. Luu et al. [10] recommended a guard condition such that “a contract code either returns the expected output or fails". To defend against Transaction Ordering Dependancy attacks, Shuai Wang et al. [56] suggested a pre-commit scheme

EBD-V4 (Untrustworthy data feeds) [99]: Some smart contracts need data feeds from outside the blockchain, but there is no guarantee that the external data source is trustworthy. So, in the case when an attacker is intentionally sending wrong information to fail the smart contract operation then it will be a hazardous situation.

Preventive Method Zhang et al. [99] developed a tool named Town Crier (TC). TC works as a trustworthy more mediate in between the external source and a smart contract. In other words, it acts as an authenticator and provides data feed privacy using encryption. Town Crier tool comprises a Town Crier smart contract that dwells on the blockchain, and a Town Crier worker which lives exterior the blockchain. The data feed request can be sent to the Town Crier contract by the client contract, which finally will send to the Town Crier worker. The Town Crier worker at that point speaks to outside information sources using HTTPS to get the information. After getting the necessary information, the worker will advance those feed requests to the Town Crier contract, which finally will ssend to the client contract.

Figure 13 depicts the timeline discovery of famous attacks on Ethereum blockchain-based smart contracts. The figure depicts the attack’s sub-cause, its related vulnerability, any financial loss, and the resolution and suggestions.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

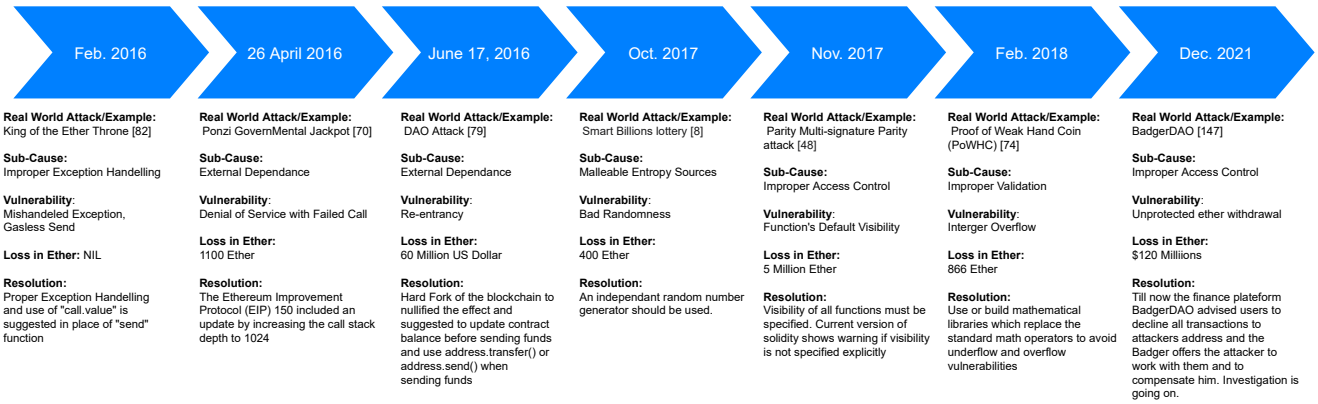

FIGURE 13. Discovey Timeline of Famous Attacks on Ethereum Smart Contract

V. ETHEREUM SMART CONTRACT ANALYSIS TOOLS

Smart contracts need to perform as intended because bugs may lead to terrific losses. Security analysis tool [30] [37] [92] [53] [100] are necessary to analyze the security vulnerabilities of the smart contract because after the deployment of the blockchain we are unable to alter the smart contract due to the immutability feature of blockchain.

We have already described an analysis tool for each vulnerability in Section III. Several surveys are presented about tools to analyze Ethereum smart contracts based on different parameters. Table 25 highlights a summary based on following parameters:

1) **Input to the tool byte code or solidity code**: Some of the tools directly work with solidity code, but due to the incompatibility of solidity code on different platforms, some tools use EVM byte code [54] for analysis.
2) **Type of analysis (static or dynamic):** Some of the tools analyze the code dynamically during execution, while some analyze the code without executing the code.
3) **Type of tool (academic or company):** Some of the tools are designed for academic purposes. Some of them are developed by private companies while some of the tools like Remix and Oyente are community tools. We have mentioned the respective company name if the tool is company-made.
4) **Implementation language:** The majority of the tools are implemented in Python. Some of them in JAVA, C++, etc.
5) **Public Availability of the tool:** Not all tools are publicly available. Some of them are private that developed by companies. Some of them are not released by the developers, only the methodology is shown in their research papers.

VI. DISCUSSION

In this systematic review, we have discussed Ethereum smart contract security vulnerabilities, detection tools, real-life attacks, and preventive mechanisms. For this purpose, we have referred 119 research articles and some online resources over the period 2016 to 2021. The comparison among the related works are then presented to evaluate various shortcomings with the existing techniques. Finally, various future directions are also discussed in the field of the Ethereum blockchain-based smart contract that can help the researchers to set the directions for future research in this domain.

A. COMPARISON WITH RELATED WORK

This paper is an extension of the existing surveys [10], [48], [54], [127]–[130], [144]–[146] on the Ethereum smart contract security vulnerabilities, well-known attacks, detection tools, and their suggested preventive methods. This work encompasses more detailed and systematic aspects for filling the research gaps related to systematic information source for Ethereum smart contract security vulnerabilities. We categorized the security vulnerabilities into three main root cause and seventeen sub-causes. This paper presents a deep insight into each vulnerability as well as its detection tools, real life attacks and prevention mechanisms. Table 26 highlights the comparisons between the present survey with the existing surveys on Ethereum smart contract vulnerabilities. It is found that the present survey covers more features as compared to the existing survey articles. Additionally, we have considered more recent literature to highlight the future research directions.

B. LIMITATIONS

Mostly our discussion is related to Ethereum blockchain-based smart contract security vulnerabilities. As research on the Ethereum blockchain contributes most of the research literature but other blockchain platforms like Solana, EOS, and Hyperledger Fabric, etc. are also having important concerns. We have not discussed security vulnerabilities related to blockchain platforms other than Ethereum. We leave the same for future work.

C. ETHEREUM BLOCKCHAIN DESIGN CHALLENGES

Blockchain is having an image problem because it is mostly connected with cryptocurrency and assumed as a world of frauds and attackers. Smart contract technology is growing

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3140091, IEEE Access

TABLE 25. Summary of Ethereum smart contract analysis tools

Tool	Byte Code	Solidity Code	Static Analysis	Dynamic Analysis	Type of Tool	Implementation Language	Publicly Available
ContractLarva [103]	✗	✗	✗	✗	Academic	Haskell	✓
E-EVM [25]	✗	✗	✗	✗	Academic	Python	✓
EtherTrust [104]	✓	✗	✓	✗	Academic	java	✓
EthIR [105]	✓	✗	✓	✗	Academic	Python	✓
FSolidM [106]	✗	✗	✓	✗	Academic	Java Script	✓
Gasper [42]	✓	✗	✓	✗	*SDSLabs	Go	✓
KEVM [107]	✓	✗	✓	✗	Academic	Python	✓
MAIAN [59]	✓	✗	✓	✗	Academic	Python	✓
Manticore [108]	✓	✗	✓	✓	*Trail of Bits	Python	✓
Mythril [72]	✓	✗	✓	✗	*ConsenSys	Python	✓
Osiris [61]	✓	✗	✓	✗	Academic	Python	✓
Oyente [10]	✓	✗	✓	✗	Community	Python	✓
Porosity [109]	✓	✗	✓	✗	*Comae Technologies	C++	✓
Rattle [110]	✓	✗	✓	✗	*Trail of Bits	Python	✓
ReGaurd [76]	✓	✓	✗	✗	*Chieftin Lab	C++	✗
Remix-IDE [111]	✗	✓	✓	✓	Community	Java Script	✓
SASC [112]	✗	✓	✓	✗	*Fujitsu	Python	✗
sCompile [113]	✗	✓	✓	✗	Academic	C++	✗
Securify [71]	✓	✗	✓	✗	Academic	Java	✓
SmartCheck [69]	✗	✓	✓	✗	Academic	Java	✓
Solgraph [114]	✗	✓	✓	✗	*Raine Revere	Java Script	✓
SolMet [93]	✗	✓	✓	✗	Academic	Java	✓
teEther [26]	✓	✗	✓	✗	Academic	Python	✗
Vandal [115]	✓	✗	✓	✓	Academic	Python	✓
Zeus [116]	✗	✓	✓	✗	*IBM Research India	C++	✗

* Tool development company name

TABLE 26. Comparison between the existing and present survey articles on Ethereum smart contract vulnerabilities

Survey	Security Vulnerability	Detection Tools	Attacks	Prevention Mechanism	Ethereum Blockchain Specific
Present Survey	✓	✓	✓	✓	✓
Li et al. [127]	✓	✗	✓	✓	✗
Zhu et al. [128]	✓	✗	✓	✓	✗
Saad et al. [129]	✗	✗	✓	✓	✓
Harz et al. [130]	✗	✓	✗	✗	✓
Angelo et al. [54]	✗	✓	✗	✗	✓
Chen et al. [145]	✓	✗	✓	✓	✓
Luu et al. [10]	✓	✗	✓	✗	✓
Atzei et al. [48]	✓	✗	✗	✓	✗
Tang et al. [144]	✓	✓	✗	✗	✗
Durieux et al. [145]	✗	✓	✗	✗	✗

at a very fast speed. Ethereum blockchain design and EVM features pose several security challenges [38], [117]–[121]. However, the following are some of the security challenges due to Ethereum's blockchain design.

1) **Verifier's nobility:** In the transaction verification process, if the majority of verifiers are dishonest, then invalid transactions can be approved and added to the chain of the block.
2) **Selfish mining:** Miner's honesty plays an important role in the mining process. An ill-intended miner can work as an adversary and affect the honest mining process.
3) **Immutability:** It does not allow editing a buggy smart contract after deployment on the blockchain. There should be some mechanism to alter the buggy or faulty smart contracts.
4) **Transaction processing time:** Transaction processing time in the Ethereum blockchain makes smart contract execution very slow, which affects the efficiency.
5) **Lack of standards:** Limited number of standards related to smart contracts increases the difficulty in the maintenance process of smart contracts.
6) **Lack of Concurrency:** Concurrency is not supported by Ethereum. Thus, to ensure security in the blockchain construction process, each blockchain node has to store Ethereum's current state and the whole transaction history. By this process, fifteen transactions per second are supported by Ethereum, which leads to serious scalability problems.

D. EVM FEATURES CHALLENGES

EVM, that is, Ethereum Virtual Machine, provides a virtual environment for the smart contract execution.

1) **EVM stack size:** Maximum stack size in EVM is up to 1024 items with 256 bits for each. It increases the chances of vulnerability. The restricted stack size can without much of a stretch, lead to weaknesses and increase the trouble of creating complex applications.
2) **Gas consumption Limit:** As the number of users increases in large-scale projects, the number of transactions will also increase accordingly. This will increase the gas consumption of the contract. Because the gas limit cannot be changed, then there are more chances of "out-of–of-gas error".

Above mentioned open issues should be investigated to develop safe smart contracts.

E. FUTURE RESEARCH DIRECTIONS

In the future, more features and functionalities will be added in Ethereum blockchain smart contracts that may lead to more security flaws. Therefore, the present vulnerability detection tools should be enhanced and new detection tools should be developed for the detection and mitigation of new security flaws. Further work could be led to finding the unknown vulnerabilities and how we can forestall them. For example, have a self-protection (revocable) mechanism for the smart contract to consequently play it safe if an attack happens. This is essential because, after the deployment on the blockchain, the smart contract is immutable. As the size of Ethereum projects increases over time, then it will pose scalability and new security issues. Thus, these challenges should be considered in the near future on the smart contract.

VII. CONCLUSION

We have presented a systematic review of the Ethereum smart contract security vulnerabilities including well-known security attacks with their preventive methods. We discussed the vulnerabilities according to their root cause, sub causes. We systematically presented the well-known attacks and their detection and analysis tools. Ethereum smart contract analysis tools are highlighted based on five parameters such as type of tool, input to the tool, type of analysis, implementation language, and availability of the tool to use. It has been found that due to the security flaws in Ethereum smart contracts, it is vulnerable to attack from adversaries. Smart contract deals in digital assets, so a single error or security flaw can cause a loss in millions. Therefore, the smart contract developers must consider such vulnerabilities and use best practices and principles while creating safe smart contracts. We have also provided insights into security challenges and future research directions for developing more robust advanced and robust vulnerabilities detection tools.

REFERENCES

[1] A. Averin and O. Averina, "Review of Blockchain Technology Vulnerabilities and Blockchain-System Attacks," 2019, doi: 10.1109/FarEastCon.2019.8934243.

[2] C. S. Wright, "Bitcoin: A Peer-to-Peer Electronic Cash System," SSRN Electronic Journal. 2019, doi: 10.2139/ssrn.3440802.

[3] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in Proceedings - IEEE 1st International Workshops on Foundations and Applications of Self-Systems, FAS-W 2016, 2016, pp. 210–215.

[4] M. In and F. Of, "Dubai Aims to Be a City Built on Blockchain Where Financial Regulation Goes in a Republican Era," Wall Str. J., pp. 3–6, 2017, [Online]. Available: https://www.wsj.com/articles/dubai-aims-to-be-a-city-built-on-blockchain-1493086080.

[5] S. Kim and G. C. Deka, Eds., Advanced Applications of Blockchain Technology, vol. 60. Singapore: Springer Singapore, 2020.

[6] M. Singh and S. Kim, "Blockchain technology for decentralized autonomous organizations," in Advances in Computers, vol. 115, 2019, pp. 115–140.

[7] C. E. Brown, O. Kunčar, and J. Urban, "Formal Verification of Smart Contracts," in Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, 2017, pp. 91–96, [Online]. Available: http://proofmarket.org.

[8] 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.

[9] M. Wohrer and U. Zdun, "Design Patterns for Smart Contracts in the Ethereum Ecosystem," in Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green [2] and Communications, Cyber, Physical and Social [2], Smart Data, Blockchain, Computer and Information Technology, iThings/Gree, 2018, pp. 1513–1520, doi: 10.1109/Cybermatics_2018.2018.00255.

[10] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the ACM Conference on Computer and Communications Security, 2016, vol. 24-28-Octo, pp. 254–269, doi: 10.1145/2976749.2978309.

[11] A. Mense and M. Flatscher, "Security vulnerabilities in ethereum smart contracts," in ACM International Conference Proceeding Series, 2018, pp. 375–380, doi: 10.1145/3282373.3282419.

[12] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the Efficiency and Reliability of Digital Time-Stamping," Seq. II, pp. 329–334, 1993, doi: 10.1007/978-1-4613-9323_24.

[13] R. Merkel, "Secrecy, authentication and public key systems." Stanford, CA, USA, p. 193, 1979.

[14] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F. Y. Wang, "An Overview of Smart Contract: Architecture, Applications, and Future Trends," in IEEE Intelligent Vehicles Symposium, Proceedings, 2018, vol. 2018-June, pp. 108–113, doi: 10.1109/IVS.2018.8500488.

[15] "EOS," 2018. https://eos.io/.

[16] "TRON," 2017. https://tron.network/.

[17] B. Canou et al., "Tezos: the OCaml Crypto-Ledger," in OCaml 2017 - OCaml Users and Developers Workshop, 2017, pp. 1–2.

[18] B. A. Kitchenham, D. Budgen, and O. Pearl Brereton, "Using mapping studies as the basis for further research - A participant-observer case study," Inf. Softw. Technol., vol. 53, no. 6, pp. 638–651, 2011, doi: 10.1016/j.infsof.2010.12.011.

[19] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," 12th Int. Conf. Eval. Assess. Softw. Eng. EASE 2008, 2008, doi: 10.14236/ewic/ease2008.8.

[20] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart Contract Templates: foundations, design landscape and research directions," pp. 1–15, Aug. 2016, [Online]. Available: http://arxiv.org/abs/1608.00771.

[21] N. Szabo, "Formalizing and securing relationships on public networks," First Monday, vol. 2, no. 9, 1997, doi: 10.5210/fm.v2i9.548.

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

[23] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. J. Kishigami, "Blockchain contract: A complete consensus using blockchain," 2015 IEEE 4th Glob. Conf. Consum. Electron. GCCE 2015, pp. 577–578, 2016, doi: 10.1109/GCCE.2015.7398721.

[24] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, "A Massive Analysis of Ethereum Smart Contracts Empirical Study and Code Metrics," IEEE Access, vol. 7, pp. 78194–78213, 2019, doi: 10.1109/ACCESS.2019.2921936.

[25] R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, "Visual emulation for Ethereum's virtual machine," in IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018, 2018, pp. 1–4, doi: 10.1109/NOMS.2018.8406332.

[26] J. Krupp and C. Rossow, "TEETHER: Gnawing at ethereum to automatically exploit smart contracts," in Proceedings of the 27th USENIX Security Symposium, 2018, pp. 1317–1333.

[27] G. A. Oliva, A. E. Hassan, and Z. M. (Jack) Jiang, "An exploratory study of smart contracts in the Ethereum blockchain platform," Empir. Softw. Eng., vol. 25, no. 3, pp. 1864–1904, 2020, doi: 10.1007/s10664-019-09796-5.

[28] R. Yang, T. Murray, P. Rimba, and U. Parampalli, "Empirically Analyzing Ethereum's Gas Mechanism," in Proceedings - 4th IEEE European Symposium on Security and Privacy Workshops, EUROS and PW 2019, 2019, pp. 310–319, doi: 10.1109/EuroSPW.2019.00041.

[29] F. Ma et al., "EVM: From Offline Detection to Online Reinforcement for Ethereum Virtual Machine," in SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering, 2019, pp. 554–558, doi: 10.1109/SANER.2019.8668038.

[30] T. Chen et al., "An adaptive gas cost mechanism for ethereum to defend against under-priced DoS attacks," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and

Lecture Notes in Bioinformatics), 2017, vol. 10701 LNCS, pp. 3–24, doi: 10.1007/978-3-319-72359-4_1.

[31] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, "VeriSolid: Correct-by-Design Smart Contracts for Ethereum," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2019, vol. 11598 LNCS, pp. 446–465, doi: 10.1007/978-3-030-32101-7_27.

[32] 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.

[33] M. Demir, M. Alalfi, O. Turetken, and A. Ferworn, "Security Smells in Smart Contracts," in Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019, 2019, pp. 442–449, doi: 10.1109/QRS-C.2019.00086.

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

[35] W. Dingman et al., "Defects and vulnerabilities in smart contracts, a classification using the NIST bugs framework," Int. J. Networked Distrib. Comput., vol. 7, no. 3, pp. 121–132, 2019, doi: 10.2991/ijndc.k.190710.003.

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

[37] J. Schütte et al., "Blockchain and Smart Contracts: Technologies, research issues and applications," Fraunhofer-Gesellschaft, Position Pap., vol. 4801, no. June, pp. 1–33, 2018, [Online]. Available: http://publica.fraunhofer.de/dokumente/N-497216.html.

[38] A. Singh, R. M. Parizi, Q. Zhang, K. K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities," Comput. Secur., vol. 88, 2020, doi: 10.1016/j.cose.2019.101654.

[39] P. Tantikul and S. Ngamsuriyaroj, "Exploring vulnerabilities in solidity smart contract," in ICISSP 2020 - Proceedings of the 6th International Conference on Information Systems Security and Privacy, 2020, pp. 317–324, doi: 10.5220/0008909803170324.

[40] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons, "Smart contracts vulnerabilities: A call for blockchain software engineering," 2018 IEEE 1st Int. Work. Blockchain Oriented Softw. Eng. IWBOSE 2018 - Proc., vol. 2018-Janua, no. March, pp. 19–25, 2018, doi: 10.1109/IWBOSE.2018.8327567.

[41] J. Song, H. He, Z. Lv, C. Su, G. Xu, and W. Wang, "An Efficient Vulnerability Detection Model for Ethereum Smart Contracts," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11928 LNCS, pp. 433–442, 2019, doi: 10.1007/978-3-030-36938-5_26.

[42] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," SANER 2017 - 24th IEEE Int. Conf. Softw. Anal. Evol. Reengineering, pp. 442–446, 2017, doi: 10.1109/SANER.2017.7884650.

[43] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A Survey on the Scalability of Blockchain Systems," IEEE Netw., vol. 33, no. 5, pp. 166–173, 2019, doi: 10.1109/MNET.001.1800290.

[44] A. Juels, A. Kosba, and E. Shi, "The ring of gyges: Investigating the future of criminal smart contracts," in Proceedings of the ACM Conference on Computer and Communications Security, 2016, vol. 24-28-Octo, pp. 283–295, doi: 10.1145/2976749.2978362.

[45] Z. Zheng, S. Xie, H. N. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," Int. J. Web Grid Serv., vol. 14, no. 4, pp. 352–375, 2018, doi: 10.1504/IJWGS.2018.095647.

[46] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," Int. J. Netw. Secur., vol. 19, no. 5, pp. 653–659, 2017, doi: 10.6633/IJNS.201709.19(5).01.

[47] E. Mik, "Smart contracts: terminology, technical limitations and real world complexity," Law, Innov. Technol., vol. 9, no. 2, pp. 269–300, 2017, doi: 10.1080/17579961.2017.1378468.

[48] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10204 LNCS, no. July, pp. 164–186, 2017, doi: 10.1007/978-3-662-54455-6_8.

[49] G. Ayoade, E. Bauman, L. Khan, and K. Hamlen, "Smart contract defense through bytecode rewriting," in Proceedings - 2019 2nd IEEE International Conference on Blockchain, Blockchain 2019, 2019, pp. 384–389, doi: 10.1109/Blockchain.2019.00059.

[50] I. Grishchenko, M. Maffei, and C. Schneidewind, A semantic framework for the security analysis of ethereum smart contracts, vol. 10804 LNCS. Principles of Security and Trust, 2018.

[51] W. J.-W. Tann, X. J. Han, S. Sen Gupta, and Y.-S. Ong, "Towards Safer Smart Contracts: A Sequence Learning Approach to Detecting Security Threats," ArXiv Cryptogr. Secur., 2018, [Online]. Available: http://arxiv.org/abs/1811.06632.

[52] J. Ye, M. Ma, T. Peng, and Y. Xue, "A Software Analysis Based Vulnerability Detection System For Smart Contracts," Stud. Comput. Intell., vol. 851, pp. 69–81, 2020, doi: 10.1007/978-3-030-26574-8_6.

[53] X. Wang, J. He, Z. Xie, G. Zhao, and S. C. Cheung, "ContractGuard: Defend Ethereum Smart Contracts with Embedded Intrusion Detection," IEEE Trans. Serv. Comput., vol. 13, no. 2, pp. 314–328, 2020, doi: 10.1109/TSC.2019.2949561.

[54] M. Di Angelo and G. Salzer, "A survey of tools for analyzing ethereum smart contracts," Proc. - 2019 IEEE Int. Conf. Decentralized Appl. Infrastructures, DAPPCON 2019, pp. 69–78, 2019, doi: 10.1109/DAPPCON.2019.00018.

[55] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in Proceedings - 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2019, 2019, pp. 8–15.

[56] S. Wang, C. Zhang, and Z. Su, "Detecting nondeterministic payment bugs in Ethereum smart contracts," in Proceedings of the ACM on Programming Languages, 2019, vol. 3, no. OOPSLA, pp. 1–29, doi: 10.1145/3360615.

[57] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: Automated Vulnerability Detection Models for Ethereum Smart Contracts," IEEE Trans. Netw. Sci. Eng., pp. 1–1, 2020, doi: 10.1109/tnse.2020.2968505.

[58] S. Zhang and J. H. Lee, "Smart Contract-Based Secure Model for Miner Registration and Block Validation," IEEE Access, vol. 7, pp. 132087–132094, 2019, doi: 10.1109/ACCESS.2019.2940551.

[59] S. Akca, A. Rajan, and C. Peng, "SolAnalyser: A Framework for Analysing and Testing Smart Contracts," Proc. - Asia-Pacific Softw. Eng. Conf. APSEC, vol. 2019-Decem, pp. 482–489, 2019, doi: 10.1109/APSEC48747.2019.00071.

[60] X. Yang, "Formal Verification for Ethereum Smart Contract Using Coq." World Academy of Science, Engineering and Technology," Int. J. Electr. Comput. Energ. Electron. Commun. Eng., vol. 5, no. 6, 2018.

[61] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in ACM International Conference Proceeding Series, 2018, pp. 664–676, doi: 10.1145/3274694.3274737.

[62] M. Wohrer and U. Zdun, "Smart contracts: Security patterns in the ethereum ecosystem and solidity," in 2018 IEEE 1st International Workshop on Blockchain Oriented Software Engineering, IWBOSE 2018 - Proceedings, vol. 2018-Janua, 2018, pp. 2–8.

[63] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 259–269, doi: 10.1145/3238147.3238177.

[64] W. C. Yang and J. Peng, "Research on EVM-Based Smart Contract Runtime Self-protection Technology Framework," in Advances in Intelligent Systems and [2], 2020, vol. 1150 AISC, pp. 617–627, doi: 10.1007/978-3-030-44038-1_57.

[65] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 9604 LNCS, 2016, pp. 79–94.

[66] D. Magazzeni, P. Mcburney, and W. Nash, "Validation and verification of smart contracts: A research agenda," Computer (Long. Beach. Calif)., vol. 50, no. 9, pp. 50–57, 2017, doi: 10.1109/MC.2017.3571045.

[67] S. O'Neal, "Ethereum Miners Vote to Increase Gas Limit, Causing Community Debate," 2020. https://cointelegraph.com/news/ethereum-miners-vote-to-increase-gas-limit-causing-community-debate (accessed Jul. 30, 2020).

[68] O. Solmaz, "The Anatomy of a Block Stuffing AttackNo Title," 2018. https://solmaz.io/2018/10/18/anatomy-block-stuffing/ (accessed Jul. 26, 2020).

[69] "SmartDec: Smartcheck." https://github.com/smartdec/smartcheck.

[70] W. Chen, Z. Zheng, E. C. H. Ngai, P. Zheng, and Y. Zhou, "Exploiting Blockchain Data to Detect Smart Ponzi Schemes on Ethereum," IEEE Access, vol. 7, pp. 37575–37586, 2019, doi: 10.1109/ACCESS.2019.2905769.

[71] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in Proceedings of the ACM Conference on Computer and Communications Security, 2018, no. June, pp. 67–82, doi: 10.1145/3243734.3243780.

[72] "Mythril." https://github.com/ConsenSys/mythril.

[73] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "EASYFLOW: Keep ethereum away from overflow," in Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019, 2019, pp. 23–26, doi: 10.1109/ICSE-Companion.2019.00029.

[74] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain games: A survey," IEEE Conf. Comput. Intell. Games, CIG, vol. 2019-Augus, pp. 1–8, 2019, doi: 10.1109/CIG.2019.8848111.

[75] N. Fatima Samreen and M. H. Alalfi, "Reentrancy Vulnerability Identification in Ethereum Smart Contracts," IWBOSE 2020 - Proc. 2020 IEEE 3rd Int. Work. Blockchain Oriented Softw. Eng., pp. 22–29, 2020, doi: 10.1109/IWBOSE50093.2020.9050260.

[76] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," in Proceedings - International Conference on Software Engineering, 2018, pp. 65–68, doi: 10.1145/3183440.3183495.

[77] J. W. Liao, T. T. Tsai, C. K. He, and C. W. Tien, "SoliAudit: Smart Contract Vulnerability Assessment Based on Machine Learning and Fuzz Testing," in 2019 6th International Conference on Internet of Things: Systems, Management and Security, IOTSMS 2019, 2019, pp. 458–465, doi: 10.1109/IOTSMS48152.2019.8939256.

[78] M. Rodler, W. Li, G. O. Karame, and L. Davi, Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. NDSS, 2019.

[79] B. Ghaleb, A. Al-Dubai, E. Ekonomou, M. Qasem, I. Romdhani, and L. Mackenzie, "Addressing the DAO Insider Attack in RPL's Internet of Things Networks," IEEE Commun. Lett., vol. 23, no. 1, pp. 68–71, 2019, doi: 10.1109/LCOMM.2018.2878151.

[80] I. Ashraf, X. Ma, B. Jiang, and W. K. Chan, "GasFuzzer: Fuzzing Ethereum Smart Contract Binaries to Expose Gas-Oriented Exception Security Vulnerabilities," IEEE Access, vol. 8, pp. 99552–99564, 2020, doi: 10.1109/ACCESS.2020.2995183.

[81] E. Albert, P. Gordillo, A. Rubio, and I. Sergey, "Running on fumes: Preventing out-of-gas vulnerabilities in ethereum smart contracts using static resource analysis," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 11847 LNCS. pp. 63–78, 2019, doi: 10.1007/978-3-030-35092-5_5.

[82] "King of the Ether: Post-Mortem Investigation." https://www.kingoftheether.com/thrones/kingoftheether/index.html (accessed Jul. 26, 2020).

[83] H. Wang, Y. Li, S. W. Lin, L. Ma, and Y. Liu, "VULTRON: Catching vulnerable smart contracts once and for all," Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. New Ideas Emerg. Results, ICSENIER 2019, pp. 1–4, 2019, doi: 10.1109/ICSE-NIER.2019.00009.

[84] X. Wang, H. Wu, W. Sun, and Y. Zhao, "Towards Generating Cost-Effective Test-Suite for Ethereum Smart Contract," in SANER 2019 - Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering, 2019, pp. 549–553, doi: 10.1109/SANER.2019.8668020.

[85] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "VerX: Safety Verification of Smart Contracts," Oakland, pp. 1661–1677, 2020, doi: 10.1109/SP40000.2020.00024.

[86] S. Jumnongsaksub and K. Sripanidkulchai, "Reducing Smart Contract Runtime Errors on Ethereum," IEEE Softw., pp. 0–0, 2020, doi: 10.1109/MS.2020.2993882.

[87] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, and D. Tigano, "Design Patterns for Gas Optimization in Ethereum," in IWBOSE 2020 - Proceedings of the 2020 IEEE 3rd International Workshop on Blockchain Oriented Software Engineering, 2020, pp. 9–15.

[88] S. Eskandari, S. Moosavi, and J. Clark, "SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11599 LNCS, pp. 170–189, 2020, doi: 10.1007/978-3-030-43725-1_13.

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

[90] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A Survey on Ethereum Systems Security," ACM Comput. Surv., vol. 53, no. 3, pp. 1–43, 2020, doi: 10.1145/3391195.

[91] D. Kirillov, O. Iakushkin, V. Korkhov, and V. Petrunin, "Evaluation of Tools for Analyzing Smart Contracts in Distributed Ledger Technologies," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11620 LNCS, pp. 522–536, 2019, doi: 10.1007/978-3-030-24296-1_41.

[92] B. C. Gupta and S. K. Shukla, "A Study of Inequality in the Ethereum Smart Contract Ecosystem," 2019 6th Int. Conf. Internet Things Syst. Manag. Secur. IOTSMS 2019, pp. 441-449, 2019, doi: 10.1109/IOTSMS48152.2019.8939257.

[93] P. Hegeds, "Towards analyzing the complexity landscape of solidity based ethereum smart contracts," Proc. - Int. Conf. Softw. Eng., vol. 7, no. 1, pp. 35–39, 2018, doi: 10.1145/3194113.3194119.

[94] Y. Fu et al., "EVMFuzzer: Detect EVM vulnerabilities via fuzz testing," ESEC/FSE 2019 - Proc. 2019 27th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., pp. 1110-1114, 2019, doi: 10.1145/3338906.3341175.

[95] B. Marino and A. Juels, "Setting standards for altering and undoing smart contracts," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016, vol. 9718, pp. 151–166, doi: 10.1007/978-3-319-42019-6_10.

[96] M. Alharby and A. van Moorsel, "Blockchain Based Smart Contracts: A Systematic Mapping Study," in 3rd International Conference on Artificial Intelligence and Soft [2], 2017, pp. 125-140, doi: 10.5121/csit.2017.71011.

[97] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," in Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016, 2016, pp. 839–858, doi: 10.1109/SP.2016.55.

[98] A. Unterweger, F. Knirsch, C. Leixnering, and D. Engel, "Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum," in 2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018 - Proceedings, 2018, vol. 2018-Janua, pp. 1–5, doi: 10.1109/NTMS.2018.8328739.

[99] E. Ben-Sasson et al., "Zerocash: Decentralized anonymous payments from bitcoin," Proc. - IEEE Symp. Secur. Priv., pp. 459–474, 2014, doi: 10.1109/SP.2014.36.

[100] C. Natoli and V. Gramoli, "The Blockchain Anomaly," in Proceedings - 2016 IEEE 15th International Symposium on Network [2] and Applications, NCA 2016, 2016, pp. 310–317, doi: 10.1109/NCA.2016.7778635.

[101] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in Proceedings of the ACM Conference on Computer and Communications Security, 2016, vol. 24-28-Octo, pp. 270–282, doi: 10.1145/2976749.2978326.

[102] A. Ghaleb and K. Pattabiraman, "How Effective are Smart Contract Analysis Tools Evaluating Smart Contract Static Analysis Tools Using Bug Injection," 2020, doi: 10.1145/3395363.3397385.

[103] S. Azzopardi, J. Ellul, and G. J. Pace, "Monitoring smart contracts: Contractlarva and open challenges beyond," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11237, pp. 113–137, 2019, doi: 10.1007/978-3-030-03769-7_8.

[104] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and tools for the static analysis of ethereum smart contracts," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10981 LNCS, pp. 51–78, 2018, doi: 10.1007/978-3-319-96145-3_4.

[105] E. Albert, P. Gordillo, B. Livshits, A. Rubio, and I. Sergey, "EthIR: A framework for high-level analysis of ethereum bytecode," arXiv, 2018.

[106] A. Mavridou and A. Laszka, "Tool Demonstration: FSolidM for designing secure ethereum smart contracts," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2018, vol. 10804 LNCS, pp. 270–277, doi: 10.1007/978-3-319-89722-6_11.

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

[108] Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G., Feist, J., Brunson, T., and Dinaburg, A. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. Proceedings - 2019 34th IEEE/ACM International Confer-

ence on Automated Software Engineering, ASE 2019, 1186–1189. https://doi.org/10.1109/ASE.2019.00133

[109] M. Suiche, "Porosity: A Decompiler For Blockchain-Based Smart Contracts Bytecode," Def Con, vol. 25, p. 30, 2017, [Online]. Available: https://www.comae.com/reports/dc25-msuiche-Porosity-Decompiling-Ethereum-Smart-Contracts-wp.pdf.

[110] "Rattle" GitHub - crytic/rattle: evm binary static analysis.

[111] A. Dika and M. Nowostawski, "Security Vulnerabilities in Ethereum Smart Contracts," Proc. - IEEE 2018 Int. Congr. Cybermatics 2018 IEEE Conf. Internet Things, Green Comput. Commun. Cyber, Phys. Soc. Comput. Smart Data, Blockchain, Comput. Inf. Technol. iThings/Gree, no. December, pp. 955–962, 2018, doi: 10.1109/Cybermatics_2018.2018.00182.

[112] E. Zhou et al., "Security Assurance for Smart Contract," 2018 9th IFIP Int. Conf. New Technol. Mobil. Secur. NTMS 2018 - Proc., vol. 2018-Janua, pp. 1–5, 2018, doi: 10.1109/NTMS.2018.8328743.

[113] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, "sCompile: Critical Path Identification and Analysis for Smart Contracts," Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 11852 LNCS, pp. 286–304, 2019, doi: 10.1007/978-3-030-32409-4_18.

[114] "Solgraph," [Online]. Available: https://github.com/raineorshine/solgraph.

[115] L. Brent et al., "Vandal: A Scalable Security Analysis Framework for Smart Contracts." 2018, [Online]. Available: http://arxiv.org/abs/1809.03981.

[116] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing Safety of Smart Contracts," NDSS, 2018, doi: 10.14722/ndss.2018.23082.

[117] Z. Zheng et al., "An overview on smart contracts: Challenges, advances and platforms," arXiv, 2019.

[118] S. Makridakis and K. Christodoulou, "Blockchain: Current Challenges and Future Prospects/Applications," Futur. Internet, vol. 11, no. 12, p. 258, Dec. 2019, doi: 10.3390/fi11120258.

[119] J. Kolb, M. AbdelBaky, R. H. Katz, and D. E. Culler, "Core Concepts, Challenges, and Future Directions in Blockchain," ACM Comput. Surv., vol. 53, no. 1, pp. 1–39, May 2020, doi: 10.1145/3366370.

[120] J. Chen, X. Xia, D. Lo, J. Grundy, and X. Yang, "Maintaining Smart Contracts on Ethereum: Issues, Techniques, and Future Challenges," arXiv, 2020.

[121] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," J. Netw. Comput. Appl., p. 102857, Nov. 2020, doi: 10.1016/j.jnca.2020.102857.

[122] Krupa, T., Ries, M., Kotuliak, I., Košál, K., and Bencel, R. (2021). Security issues of smart contracts in ethereum platforms. Conference of Open Innovation Association, FRUCT, 2021-January(January). https://doi.org/10.23919/FRUCT50888.2021.9347617

[123] Noor Aidee, N. A., Johar, M. G. M., Alkawaz, M. H., Iqbal Hajamydeen, A., and Hamoud Al-Tamimi, M. S. (2021). Vulnerability Assessment on Ethereum Based Smart Contract Applications. 2021 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), 13–18. https://doi.org/10.1109/I2CACIS52118.2021.9495892

[124] Alkhalifah, A., Ng, A., Watters, P. A., and Kayes, A. S. M. (2021). A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks. Frontiers in Computer Science, 3(February), 1–15. https://doi.org/10.3389/fcomp.2021.598780

[125] Moona, A., and Mathew, R. (2021). Review of Tools for Analyzing Security Vulnerabilities in Ethereum based Smart Contracts. SSRN Electronic Journal, Icicnis, 524–533. https://doi.org/10.2139/ssrn.3769774

[126] Ali, A., Abideen, Z. U., and Ullah, K. (2021). SESCon: Secure Ethereum Smart Contracts by Vulnerable Patterns' Detection. Security and Communication Networks, 2021, 1–14. https://doi.org/10.1155/2021/2897565

[127] X. Li, P. Jiang, T. Chen, X. Luo, en Q. Wen. (2020). "A survey on the security of blockchain systems", Future Generation Computer Systems, vol 107, bll 841–853, 2020. https://doi.org/10.1016/j.future.2017.08.020

[128] L. Zhu et al. (2018). "Research on the Security of Blockchain Data: A Survey", CoRR, vol abs/1812.02009, 2018. https://doi.org/10.1007/s11390-020-9638-7

[129] M. Saad et al.(2019). "Exploring the Attack Surface of Blockchain: A Systematic Overview", CoRR, vol abs/1904.03487, 2019.

[130] D. Harz en W. J. Knottenbelt (2018). "Towards Safer Smart Contracts: A Survey of Languages and Verification Methods", CoRR, vol abs/1809.09805, 2018.

[131] Khan, S.N., Loukil, F., Ghedira-Guegan, C. et al. "Blockchain smart contracts: Applications, challenges, and future trends", Peer-to-Peer Netw. Appl. 14, 2901–2925 (2021). https://doi.org/10.1007/s12083-021-01127-0

[132] ang, Z., Jin, H., Dai, W. et al. "Ethereum smart contract security research: survey and future research opportunities", Front. Comput. Sci. 15, 152802 (2021). https://doi.org/10.1007/s11704-020-9284-9

[133] Teng Hu, Xiaolei Liu, Ting Chen, Xiaosong Zhang, Xiaoming Huang, Weina Niu, Jiazhong Lu, Kun Zhou, Yuan Liu, "Transaction-based classification and detection approach for Ethereum smart contract",Information Processing Management, Volume 58, Issue 2, 2021, 102462, ISSN 0306-4573, https://doi.org/10.1016/j.ipm.2020.102462.

[134] Bhardwaj, A., Shah, S.B.H., Shankar, A. et al. "Penetration testing framework for smart contract Blockchain", Peer-to-Peer Netw. Appl. 14, 2635–2650 (2021). https://doi.org/10.1007/s12083-020-00991-6

[135] Anna Vacca, Andrea Di Sorbo, Corrado A. Visaggio, Gerardo Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges", Journal of Systems and Software, Volume 174, 2021, 110891, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2020.110891.

[136] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu and X. Wang, "Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection," in IEEE Transactions on Knowledge and Data Engineering, doi: 10.1109/TKDE.2021.3095196.

[137] Rameder, H. (2021). "Systematic Review of Ethereum Smart Contract Security Vulnerabilities, Analysis Methods and Tools" [Diploma Thesis, Technische Universität Wien]. reposiTUm. https://doi.org/10.34726/hss.2021.86784

[138] Chen, J., Xia, X., Lo, D. et al. "Maintenance-related concerns for post-deployed Ethereum smart contract development: issues, techniques, and future challenges" Empir Software Eng 26, 117 (2021). https://doi.org/10.1007/s10664-021-10018-0

[139] Z. Wan, X. Xia, D. Lo, J. Chen, X. Luo and X. Yang, "Smart Contract Security: A Practitioners' Perspective," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 1410-1422, doi: 10.1109/ICSE43902.2021.00127.

[140] Palina Tolmach, Yi Li, Shang-Wei Lin, Yang Liu, and Zengxiang Li. 2021. "A Survey of Smart Contract Formal Specification and Verification", ACM Comput. Surv. 54, 7, Article 148 , 38 pages. DOI:https://doi.org/10.1145/3464421

[141] S. Ji, D. Kim and H. Im, "Evaluating Countermeasures for Verifying the Integrity of Ethereum Smart Contract Applications," in IEEE Access, vol. 9, pp. 90029-90042, 2021, doi: 10.1109/ACCESS.2021.3091317.

[142] N. A. Noor Aidee, M. G. M. Johar, M. H. Alkawaz, A. Iqbal Hajamydeen and M. S. Hamoud Al-Tamimi, "Vulnerability Assessment on Ethereum Based Smart Contract Applications," 2021 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), 2021, pp. 13-18, doi: 10.1109/I2CACIS52118.2021.9495892.

[143] Rajesh Gupta, Mohil Maheshkumar Patel, Arpit Shukla, Sudeep Tanwar, "Deep learning-based malicious smart contract detection scheme for internet of things environment", Computers Electrical Engineering, 2021, 107583, ISSN 0045-7906, doi: 10.1016/j.compeleceng.2021.107583.

[144] Tang X., Zhou K., Cheng J., Li H., Yuan Y. (2021) The Vulnerabilities in Smart Contracts: A Survey. In: Sun X., Zhang X., Xia Z., Bertino E. (eds) Advances in Artificial Intelligence and Security. ICAIS 2021. Communications in Computer and Information Science, vol 1424. Springer, Cham. doi: 10.1007/978-3-030-78621-2-14

[145] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A Survey on Ethereum Systems Security," ACM Computing Surveys, vol. 53, no. 3, pp. 1–43, 2020.doi: 10.1145/3391195

[146] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020. doi: 10.1145/3377811.3380364

[147] R. Lawler, "Someone stole $ 120 million in crypto by hacking a DeFi website," The Verge, 03-Dec-2021. [Online]. Available: https://www.theverge.com, [Accessed: 11-Dec-2021].