EasyChair Preprint

№ 6761

Security Vulnerabilities of Blockchain-Based Smart Contracts and Countermeasures: a Survey

Epherem Merete Sifra and Gaofei Wu

October 3, 2021

Security Vulnerabilities of Blockchain-Based Smart Contracts and Countermeasures: A Survey

Epherem Merete Sifra and Gaofei Wu

Xidian University, Xi 'an, Shaanxi Province 710071 China
wmerete@gmail.com, wugf@nipc.org.cn

Abstract. A smart contract is a computer program that defines an agreement between multiple parties to follow specific logic and agreement and executes automatically when certain conditions are met. Recently, blockchain-based smart contracts have become popular due to their immunity, decentralization, and security insurance in a trust-less environment. However, they are extremely susceptible to various security risks, so it is necessary to conduct a systematic investigation of existing security countermeasures and use unified evaluation criteria. Although there have been some security investigations on blockchain-based smart contracts, they have failed to conduct a reasonable analysis of existing security solutions with a unified criterion. Therefore, in this article, we conduct a comprehensive investigation of the latest work on security in blockchain-based smart contracts. We proposed a system and security model representing smart contracts based on blockchain. On this basis, we analyzed the security requirements of blockchain and smart contracts, and we use these requirements as evaluation criteria to analyze the works under investigation. Based on the results of the analysis, we have identified a series of open research questions and future directions to stimulate research work on protecting blockchain-based smart contracts.

Keywords: Blockchain, Smart Contracts, Security, Privacy, Cryptocurrency, Vulnerability.

1 Introduction

Smart contracts are a computer program proposed by Nick Szabo in 1996, which defines smart contracts as a computer program designed to facilitate, verify, and enforce legal contracts and negotiations in the digital world. A traditional smart contract was contingent on a centralized trusted party; thus, it confronts a single point of failure problem. Also, an untrusted centralized party will cause huge benefit losses to users. Due to this, cooperation among multiple smart contracts platforms is not practical. A centralized smart contracts platform confronts severe security and trust problems; hence, ensuring trust is not feasible. To tackle these problems, some researchers proposed to used blockchain to build a decentralized smart contracts platform. Blockchain is a decentralized ledger that stores data in a decentralized and append-only way. A bitcoin was the first blockchain introduced by Satoshi to implement decentralized cryptocurrency and quickly attracts considerable attention in academia and industry, due to its decentralization, immunity, consistency, and trust characteristics. These advantages make it an ideal platform for deploying smart contracts. A blockchain-based smart contract is a collection of computer codes stored on a blockchain, and these codes are only valid when specific terms and conditions are met. The first blockchain-based decentralized smart contract platform is Ethereum. Contrary to the centralized smart contracts platforms, blockchain-based smart contracts support the trusted execution without any trusted third party (TTP), and it avoids security risks raised by untrusted or insecure centralized parties. Despite the popularity of blockchain-based smart contracts, they are highly vulnerable to various security risks. For instance, the adversary can take advantage of code bugs in smart contracts to obtain benefits illegally. Also, they can harm the trusted execution of contracts by breaking the underlying consensus mechanisms for blockchain. The attacks that target the peer-to-peer network can cause inconsistent contracts execution results. Several real-world attacks are aiming at blockchain-based smart contracts. Such as the attacks in Ethereum Decentralized Autonomous Organization (DAO) in 2016 [3] and the Parity Wallet attacks in 2017. These attacks have resulted in significant losses. For example, the DAO attack caused a loss of $ 60 million worth of ether coins. The cause of this attack was the gap in the recursive call bug on the smart contract. This attack also forced the Ethereum blockchain to apply a hard fork on its chain. Because of this hard fork, the Ethereum blockchain split into two blockchains, i.e., Ethereum and Ethereum classic. Therefore, exploring the security and privacy issues and their countermeasures in blockchain-based smart contracts in a systematic way is vital to alleviate the risks.

There are several surveys work aiming at blockchain security and privacy [4] [5] [6] [7] [8]. However, they have failed to conduct a systematic investigation into blockchain-based Smart contracts and possible attacks and security issues. Some works tried to investigate smart contracts code security. They analyzed the security vulnerabilities of Smart contracts and outlined common program flaws that can lead to vulnerabilities. However, they

fail to consider other security issues. To better illustrate the superiority of our work, we have summarized the comparison of our work with existing surveys in Table 1.

Contributions: This paper surveys the potential attacks and their security countermeasures that target blockchain-based smart contracts. We summarized the existing attacks on blockchain smart contracts, and we proposed security requirements for blockchain-based smart contracts. We have utilized these requirements as criteria to evaluate the security solutions for blockchain smart contracts. Finally, we point out a series of open issues about the security of blockchain-based smart contracts and discuss the future research direction. The remainders of our paper are organized as follows. In Section II, we define a system model for blockchain-based smart contract platforms and analyze its unique characteristics. In Section III, we propose a security model and summarize the potential attacks. Based on the security model, we summarize the security requirements. Section IV presents a detailed analysis of existing security countermeasures. In Section V, based on the analysis and comparison, we figure out a series of open research issues and future directions. Finally, Section VI the whole paper concludes here.

Table 1. . Comparison of Our Survey with Other Existing Surveys

Topic	[4]	[5]	[6]	[7]	[8]	Our work
Summarize smart contracts attack	Yes	No	Yes	Yes	No	Yes
Propose a set of Evaluation criteria	No	No	Yes	Yes	No	Yes
Analyze security requirement	No	No	No	No	No	Yes
Analyze security solution	Yes	No	Yes	Yes	Yes	Yes
Propose open issue	No	Yes	No	Yes	Yes	Yes
Propose a taxonomy	Yes	Yes	Yes	Yes	Yes	Yes

Yes; denotes a corresponding topic is satisfied,

No; denotes a corresponding topic is not satisfied

2 System Model and Unique Characteristics

In this section, we define the system model and analyze the unique characteristics of blockchain-based smart contract platforms.

2.1 System Model

The blockchain contains a summary of all the transactions and smart contracts in the blocks. It is composed of several blocks linked together to form a chain with a hashed previous address. Data recorded in blockchain cannot be deleted or modified, which ensures the integrity and trust of the data. When someone wants to write a record (usually called a transaction) to a blockchain, the miner or validator in the blockchain will verify the validity of the transaction based on consensus. A typical blockchain consists of multiple computing devices, each of which is connected through a peer-to-peer (P2P) network. There is no centralized party for node management and network maintenance. In order to better analyze its characteristics, we strive to use a multi-layered model to represent the conceptual model of the blockchain. The multi-layer blockchain model consists of six layers, i.e., Data Layer, Physical Layer, Consensus Layer, Network layer, Presentation layer, and Application Layer, as shown in Figure 1.

Application Layer: The application layer serves as an interface for users and applications to access blockchain blocks. The application layer itself is not an application, but it provides functions for running applications and creating transactions. The transaction is an important part of the blockchain ecosystem. In this layer, there is two-party that have a power to create a transaction, a person who owns a token and smart contract. A smart contract can create a transaction to a specific address when a certain conditions set are met. An event that will update the state of the blockchain is considered as transaction. In bitcoin, transaction is a transfer of the locked Unspent

Transaction Output (UTXO) or simply bitcoin to the new address. However, in the case of Ethereum, transaction is the transfer of ETH from account or contract to another account or contract. Meanwhile, deploying a contract to the blockchain is also considered a transaction, but the contract is not transferred to a specific address, instead, it will be deployed without the recipient address. One of the functions that have been implemented in Ethereum blockchain through this layer is decentralized applications (Dapp). Through this Dapp, many blockchain-based applications have been implemented in different fields, such as the Internet of Things (IoT) [23], supply chain management systems [26], vehicular ad hoc network (VANET) [24], healthcare [27], Real state [25], and voting [22].

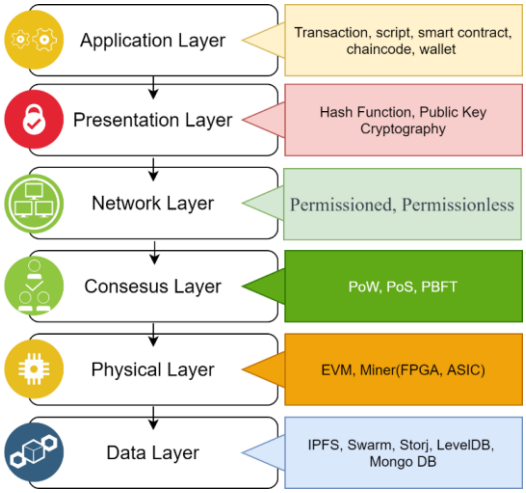

Fig. 1. Multi-layer Blockchain System Model

Network Layer: Nodes in the blockchain are identified as computing devices Interconnect with neighbor nodes to establish a peer-to-peer (P2P) network. There are many P2P protocols that can facilitate the propagation of data between the sender and the receiver. Kadmila [11] P2P protocol is the basis protocol for node lookup process in most blockchains ecosystems. The responsibility of this layer is to propagate the transactions and new executed transactions between P2P nodes by finding the best route without any central management. Generally, blockchain network can be categorized into two group, i.e., public and private blockchains. In the public blockchain network, all records are publicly visible. Anyone who wants to join the network can join without permission and can also participate in the consensus process. On the contrary, the private blockchain record is invisible to the public, and it can only be accessed by nodes with access rights. Table 2 summarized the current blockchains platform and their futures. Figure 2 (a) shows a permissionless blockchain network model and Figure 2 (b) show a permissioned blockchain network model. In this model, different types of devices are interconnected through the use of a P2P connection protocol.

Consensus Layer: Consensus is a set of rules approved by blockchain members to verify transactions in the blockchain. Nowadays, different blockchain consensus mechanisms have been proposed and applied. For example, Bitcoin chose to apply Proof of Work (POW) for its cryptocurrency [2]. There are other forms of consensus protocols applying the concept of Proof of Stake (PoS) [17], Proof of Elapsed Time (PoET) [18], Delegated Proof of Stake (DPoS) [19], Proof of Activity (a hybrid of proof of work and proof of stake) [20], Proof of Importance [21], proof of storage, etc. The process of verifying and approving transactions in the blockchain is called mining, and this process may require high computing power and electricity in case of PoW consensus. Another popular consensus algorithm proposed by Ethereum is PoS. PoS provides block verification capabilities for nodes with more stake in the blockchain. Nodes with high computing power or more stake perform block verification before being added to the chain. A miner or validator is a computing device that has the power to verify transactions or execute smart contracts. In the PoW consensus, miners use its computing power to obtain the ability to create the next block. On the contrary, in order for miners to mine the next block in PoS, they must deposit a certain amount of stake on theirs address, and then, according to the algorithm, others miner will vote to delegate to the next miner.

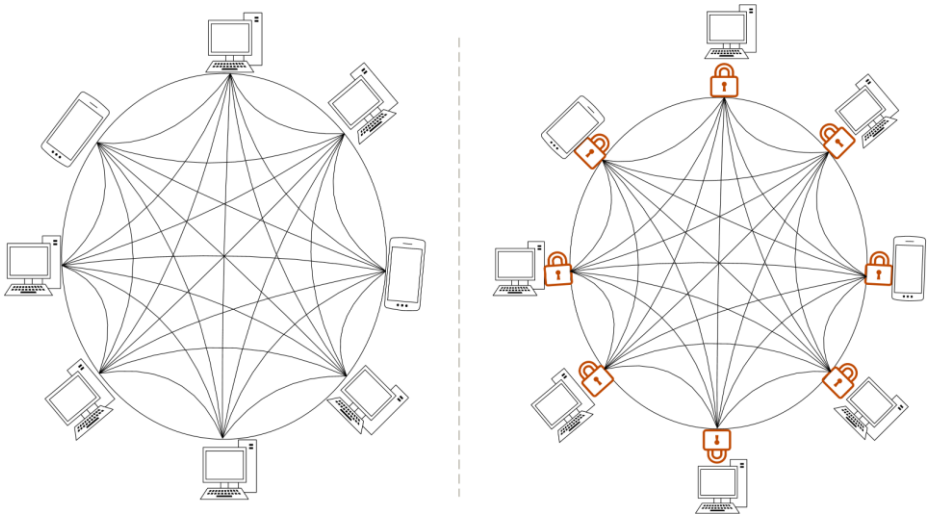

Fig. 2. (a) Public Blockchain (b) Private Blockchain

Physical Layer: The physical layer in this model is responsible for facilitating the execution of any type of transaction on heterogeneous hardware. The smart contract is deployed in the blockchain, and there are different types of computers in the P2P network. These computers are called nodes. Most nodes have different working environments. Executing smart contracts in this environment will make smart contracts vulnerable to serious security attacks. To alleviate this situation, the first smart contract implementation was developed using client software with a virtual machine. There are different types of virtual machines in the blockchain ecosystem, which are mainly used to execute transactions (smart contracts and chain codes). The Ethereum Virtual Machine (EVM) is responsible for running smart contracts written by Solidity and Vyper on the Ethereum blockchain.

Table 2. Summary of Blockchain Platforms

Platforms	Execution Environment	Smart contracts Language	Type	Consensus
Ethereum[12]	EVM	Solidity, Vyper	Public	PoW(PoS Expected)
Hyperledger Fabric [13]	Docker	JavaScript, Go	Private	BFT
Cardano[14]	KEVM	Haskell	Public	PoS(ouroboros)
NEO [15]	NeoVM	Java, C#, GO, JS	Public	dBFT

Data Layer: The data layer contains the data structure of the blockchain. A blockchain is a chain of blocks, and each block contains a list of valid transactions recorded in the ledger within a given period of time. A block is the basic unit of the blockchain [16]. Each block consists of two parts, namely the block header and the block body. The block header consists of the block version, random number, timestamp, previous block hash, and Merkel tree root hash. The other part of the block consists of the transaction counter and the transaction history of the block. One valid block contains previous block hash to link with the previous block. The root of the Merkle tree retains the hash value of each transaction to ensure the immunity of the transactions. The structure of blocks is not similar on all blockchain platforms, but most of the block structure consist of the above two main parts and previous block hash; therefore, each block linked together and its chain immutable. Figure 3 show the basic block structure. To retrieve the created block and transaction each node stores the valid block index and location in their local database. Ethereum uses a database based on Log Structured Merge Tree (LSM) to store an index of valid transactions.

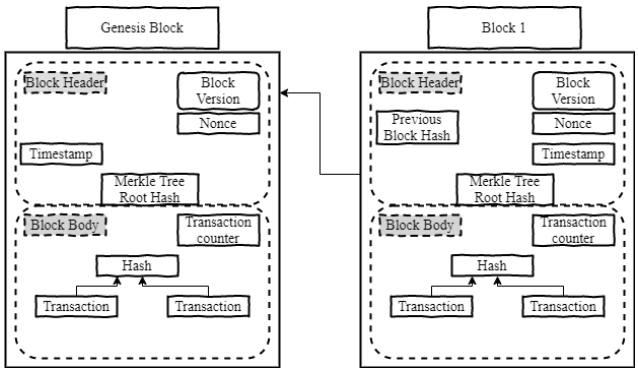

Fig. 3. Basic block structure

3 System Requirement

In this section, we define the security model and summarize the potential attacks on blockchain-based smart contracts. Then, we propose a series of security requirements to ensure the security of blockchain-based smart contracts.

3.1 Security Model

In our security model, there is no fully-trusted centralized entity responsible for system management. All nodes are rational and benefit-driven; thus, their trust cannot be ensured. Besides, blockchain nodes do not trust each other, and all nodes make decisions based on their benefits and the information recorded in the blockchain. Also, an adversary may control a proportion of blockchain node. Furthermore, it can broadcast numerous illegal transactions or block over the network. The smart contract built by the adversary can trigger attacks against smart contracts by exploit their code weaknesses. Finally, we consider that the adversary can conduct a series of attacks, summarized in the next subsection.

Potential Attacks: In our security model, we consider an adversary who can conduct a series of attacks. In order to better demonstrate, we divide potential attacks into several categories, namely, attacks on the blockchain network, attacks on the consensus mechanism, and attacks on smart contracts.

Attacks to blockchain network*:* Blockchain is based on P2P networking, which lacks a logically centralized trusted authority for network maintenance. Therefore, the adversary can take advantage of the weaknesses of the P2P network and conduct several attacks, as summarized below:

- *Denial of Service (DoS)*: DoS aim to make the system unobtainable, which is implemented by either forcing a victim computer to reset or consuming its resource, e.g., CPU cycles, memory or network bandwidth [29]. DoS can be conducted with distributed nodes, which is called Distributed DoS (DDoS).
- *Eclipse Attacks:* Eclipse attack refers to that an adversary isolates a blockchain miner from all its incoming and outgoing neighbors, thus separating it from other peers in the network [28]. The adversary uses many nodes or botnets to exploit the victim's nodes. Once the adversary effectively isolates the victims from their peer network, they can send obsolete/fake information to the victim. Besides, the adversary can use the victims compute power for their purpose.
- *Sybil Attack:* A Sybil attack is an effort of an adversary to deceive other peers as many distinct identities [30]. The adversary creates multiple fake identities for various intentions, e.g., 51% attack.

Attacks to Consensus Mechanism*:* The security of blockchain highly relies on the underlying consensus mechanism. Especially, in most public blockchain systems, miners can obtain rewards by generating a new block. Therefore, an adversary has strong incentives to harm consensus mechanisms for more profits. To better analyze the security of blockchain, we discuss the attacks on the consensus mechanism in this section.

- *Selfish Mining:* The adversary can conduct selfish mining attack by holding the created blocks instead of broadcasting to the chain, which causes honest miners to waste their computing resources. In this way, adversary can gain an advantage over other miners with less computing resources. Selfish mining [31] can effectively reduce the rewards of honest miners. It can also cause other problems, such as forks after withholding,

honest miners always broadcast the mined blocks on time, but adversary build the longest chain privately. Many consensus mechanisms accept that the longest chain is legal. When the adversary broadcasts the hidden chain to the blockchain network, the consensus will accept it as a valid chain.

- *Majority Attack:* Majority attack refers that an adversary controls a proportion of miners. so that, it can control the blockchain system. For example, when an adversary controls over 50% of the computational power in Bitcoin system, it can reverse the transactions and interfere with the process of storing new block [32]. Several attacks can be performed based on majority attack, e.g., double spending, selfish mining.
- *Double-Spending Attack:* Double-spending is a vulnerability in cryptocurrencies or other digital financial systems in which the same single digital token can often be used [33]. Because of this, two or more conflicting blocks with the same height are created, it is possible to cause this kind of attack, resulting in inconsistency, called derivation. Therefore, some encryption tokens may be temporarily used in two conflicting blocks, and then only one block is included in the honest chain.

***Attacks to Smart Contract code**:* Attacks on smart contracts code directly relate to the bugs in contracts, most of which are caused by programming language weaknesses [34]. There are many serious attacks on smart contacts code in the real world, resulting in huge losses to Ethereum. Therefore, we summarize the typical attacks of this type as below:

- *Reentrancy:* Ethereum smart contracts can call and utilize the codes of other external contracts [35]. Contracts regularly handle Ether and other tokens. Based on the conditions set to the contract, these Ether and tokens will be sent to various external or contracts addresses. To send Ethers or tokens to another address, the contract must submit external calls to another contract. An adversary can intercept these external calls. Then can force the contract to execute further code through a fallback function including calls to itself. The notorious DAO hack was one of the victims of this type of attack. A reentrancy attack may occur to a function that makes an external call to another untrusted contract before resolving the vulnerability. An adversary who gains control over the contract can steal Ether from another contract by using vulnerable function calls. Listing 1 shows a code snippet that can be used to perform a reentrancy attack. This code snippet shows the reentrancy callback function vulnerability in line 5 and the attacker's contract starting from line 9.

```
1   contract DAO {
2       fuction withdraw(uint withdrawAmount) public
3       {
4           require(withdrawAmount <= balances[msg.sender]);
5           msg.sender.call.value(withdrawAmount)("");
6           balances[msg.sender] -= withdrawAmount;
7       }
8   }
9   contract Attacker {
10    function attack()public payable{
11          DAO.withdraw(amount);
12          function() public payable {
13              if(address(DAO).balance>=amount)
14              DAO.withdraw(amount);
15          }
16      }
17  }
```

Listing 1. Contract with simple reentrancy vulnerability and attack

- *Call Depth Attack:* Due to call depth attacks, even completely trusted and accurate calls may not be executed. This is because the depth of the "call stack" is limited. If the attacker repeatedly makes several calls and brings the depth of the key to 1023, they can call your function and immediately disable all sub-calls.

```
1  contract auction {
2      mapping(address => uint) refunds;

3      function withdrawRefund(address recipient){
4          uint refund = refunds[recipient];
5          refunds[recipient] = 0;
6          recipient.send(refund);
7      }
8  }
```

Listing 2. contract with call depth attack bug

- *Transaction-Ordering Dependence (TOD):* A transaction will store in the mempool for a while. Miners can know what will happen in each transaction before being included in a block [37]. This can be dangerous for things like decentralized markets. Transactions to purchase certain tokens can be seen by the public, and market orders can be modified and executed before other transactions are included. Preventing this situation is difficult since every transaction will come down to the specific contract itself.
- *DoS (Denial of Service):* Malicious nodes can trigger DoS attacks on smart contracts. For example, malicious leaders will always fail by refunding their addresses, and they can prevent other nodes from calling the bid () function. Listing 3 shows a code snippet that can be used to perform a DoS attack. Under normal circumstances, the node with the highest bid will become the leader, and the previous leader value will be automatically refunded. On the contrary, if the current leader is malicious, it can intercept the refund to his address. Therefore, the next leader will not be assigned to the bid on time.

```
1  contract Auction {
2      address currentLeader;
3      uint highestBid;

4      function bid() {
5          if (msg.value <= highestBid) { throw;
6          }

7          if (!currentLeader.send(highestBid)) {throw; }
       // Refund the old leader, and throw if it fails

8          currentLeader = msg.sender;
9          highestBid = msg.value;
10     }
11 }
```

Listing 3. contract with DoS vulnerability

- *Timestamp Dependence:* The average timestamp and block number can be used to estimate time. But The timestamp can be exploited by the miner, so it should not be used in the key parts of the contract. Listing 4 show simple timestamp dependence bug, in line 2 the now (the current time) can be manipulated by the miner.

```
1 uint startTime = SOME_START_TIME;

2 if (now > startTime + 1 week)
3 { // the now can be manipulated by the miner
4 }
```

Listing 4. contract with timestamp dependence vulnerability

- *Short Addresses:* Short address attacks are a side effect of the EVM itself accepting incorrectly padded arguments [36]. Attackers can exploit this by using specially-crafted addresses to make poorly coded clients encode arguments incorrectly before including them in transactions.
- *Delegate call:* Delegate call is a special variant of message call in the contract. It is the same as message call except that the code at the target address is executed in the context of the calling contract. This means that the contract can dynamically load code from different addresses at runtime. The storage, current address and balance still refer to the calling contract, but the code is taken from the called address.
- Mishandled Exception: Some operations will not throw an exception, but returns a Boolean value. If this return value is not checked, the contract will continue to execute even if the payment fails, which can easily lead to inconsistencies.

3.2 Security Requirements

We explored the potential security risks in blockchain-based smart contracts and summarized the security requirements to mitigate these risks.

Integrity (In): Integrity ensures that the block content in the chain is in the correct format when it is first created, also, it is a way to protect the unauthorized tampering of information. Attacks like 51% and Sybil attacks will challenge the integrity of Block in the blockchain. Malicious nodes try to modify the block, and if they successfully modify the block, the integrity will be compromised. In order to avoid such attacks, nodes should support the integrity of each block at any time.

Availability (Av): Any node and block token in the blockchain should always be available even the hostile node triggered the DoS or DDoS attacks over the network. If the node availability is not protected, the adversaries can use them for their purpose.

Non-repudiation (Nr): The adversary node in the blockchain tries to put its illegal block into the legal chain by cheating or attacking the honest node. Every node participating in a legal or illegal transactions cannot deny the transaction in the blockchain. Non-repudiation is about collecting, maintaining, and providing undeniable evidence about every transaction from sender to receiver. This helps to easily find the adversary's actions.

Confidentiality (Cf): Confidentiality refers to hiding transaction information, smart contract data, and preventing it from being exposed to unauthorized parties.

Correctness (Co): The smart contract code should always be correct, otherwise small errors in the code may cause significant losses.

Robustness: The security solution should be able to resist various security attacks against blockchain-based smart contracts.

4 Countermeasures

In this section, we analyze existing solutions for security vulnerabilities using the proposed security requirement as criteria. We searched for relevant papers using blockchain, Smart Contracts Security, Cryptocurrency, and Vulnerability as keywords in four databases, i.e., ACM Digital Library, IEEE Library, Elsevier Library, and Springer Library. These countermeasures can be roughly divided into three categories according to their design goals, Network-Based Attack Solution, Consensus-Based Attacks Solution, and Smart Contracts Based Attacks Solution, TABLE 2 summarizes the existing works into three taxonomies. TABLE 3 summarizes and compares the existing works based on the proposed security requirements.

4.1 Network Based Attacks Solutions

Muhammad et al. [38] proposed two solutions to mitigate Bitcoin DDoS mempool attack, i.e., (fee-based mempool and Age-based mempool). The main idea is to leverage Lyapunov Optimization to control mempool size by taking mining fee, relay fee, and age of the transaction as a criterion to detect malicious transactions. Specifically, the age-based mempool can filter malicious transactions with a high true positive (TP) rate, which makes the miner available in the DDoS attacks environment. The fee-based mempool can only filter legitimate transactions from a few numbers of malicious transactions. When the scheme is applied in an idle state, the legitimate transaction will be removed from the mempool to fulfill the algorithm requirements. This can be a reason for inconsistency problems in the blockchain. This solution can resist DDoS attacks and improve the availability of blockchain nodes. Hence, it can meet the Av requirements. However, it does not consider other security requirements.

Wust and Gervais. [39] Launched an eclipse attack using the Ethereum block propagation algorithm over the main net active nodes. The node that misses a block in propagation time will start synchronizing the missed block from the peer node. In the attack, the attacker sends the forged blockchain to the victim and further uses the weakness in the block propagation algorithm to prevent the victim from establishing contact with others. Instead of synchronizing the missing block and receiving a new block from one node, this scheme allows the victim node to request from many nodes. Also, receiving the missing nodes and new nodes from a different node can reduce

eclipse attacks. Besides, DDoS and double-spend attacks are also able to mitigate. Generally, this countermeasure satisfies Av security requirements by using many nodes to broadcast a block.

Aiming at resisting the three types of eclipse attacks in Ethereum, Marcus et al. [40] proposed a scheme that exploits the Kademlia neighbor finding protocols vulnerability. It aims to overcome the risk of establishing max-peers with the victim node's TCP connection. To overcome this attack, the upper bound of incoming connections is limited. For eclipse by a table poisoning attack, this scheme comes up with five different countermeasures. One-to-one mapping is one of the countermeasures used to mitigate eclipse by table poisoning attacks. In this countermeasure, the Internet Protocol (IP) address of the node will be mapped with the node ID generated using the ECDSA algorithm. This one-to-one scheme can mitigate Sybil and eclipse attacks and it satisfies the Av and In requirement at the network layer. The authors also recommend running a seed process to protect victim nodes, regardless of whether the node table is empty or full. Timestamps in UDP packets are also used to trigger an eclipse attack called manipulating time attack over a blockchain. To alleviate this, the scheme used nonce instead of timestamps. These countermeasures have been evaluated using the proposed security requirements and satisfy the Availability requirement by protecting the nodes from eclipse attacks and Nonrepudiation requirement by mapping the node's IP address with a node's ID.

Swathi et al. [41] proposed a distributed Sybil attack detection strategy. The scheme tracks the behavior of each miner in the transaction mining process and stores it in a public table. For each transaction, the scheme includes the address of the miner who confirmed the transaction in the block. The behavior table will be updated continuously when a new block is generated. This provides clear information about the responsible nodes for each transaction. This scheme can mitigate Sybil attacks by checking the transaction history of nodes from the publicly available behavior table. In addition, it can also mitigate DDoS and Eclipse attacks triggered by Sybil nodes. However, If the attacker successfully simulates trusted nodes, the trusted node identity will be used by the hostile nodes for illegal activity. In addition, the blockchain will be vulnerable to 51% and selfish mining attacks. Because the attacker deceives other nodes by using the trusted node's identity to modify the behavior table entries of trusted nodes by releasing blocks. This will trick other nodes to reject the honest node's transactions. The scheme does not provide any node identity authentication mechanisms. Therefore, this will impair the availability of the node. The behavior monitoring table does not have a consensus, encryption, or tamper-proof verification mechanism. In general, this solution only meets the security requirements of *Nr*.

Table 3.
The Classifications of Potential Attack and Existing Solution Summary

Attack's target	Potential attacks	Solutions	Publication Year
Network Based Attacks	DDoS Attack	[38], [39]	2019,16
	Eclipse Attack	[39], [40]	2016,19
	Sybil Attack	[41], [43]	2019,20
Consensus Based Attacks	Selfish Mining	[42], [44], [45]	2014,16
	Double-spend Attack	[39], [43]	2016,20
	51%(Majority) Attack	[46], [47], [43], [45]	2018,19,20,19
Smart Contracts Based Attacks	Reentrancy	[48], [49], [50], [51], [52], [53], [54], [55]	2018,18,18,18,16,18,18,16
	Call Depth Attack	[49], [50], [51], [52], [53], [55]	2018,18,18,16,18,16
	DoS(Denial of Service)	[54], [55]	2018,16
	Timestamp Dependence	[49], [50], [52], [53], [54]	2018,16,18,18,18
	Transaction Ordering Dependence (TOD)	[51], [52], [53]	2018,16,18
	Mishandled Exception	[49], [50], [51], [52], [53], [55]	2018,18,18,16,18,16

4.2 Consensus Based Attacks Solutions

Heilman. [42] proposed a new scheme against selfish mining called" Freshness Preferred" (FP), which uses random beacons to generate unforgeable timestamps. This unforgeable timestamp is used to punish miners who detain mined blocks and reduce the profit of selfish mining cartels. The proposed scheme increases the threshold of the mining power of the selfish cartel to launch a selfish mining attack from 25% to 32%. Due to the existence of thresholds and unforgeable timestamps, the FP scheme can ensure honest nodes to mine transactions without worrying about forks related to selfish mining. Therefore, the FP scheme protects the In at the consensus level. This scheme can minimize selfish mining activities between miners and timestamp dependence attacks on block timestamps. Meanwhile, this scheme uses non-forgeable timestamps to meet the security requirements of blockchain, In, and Nr.

Lixiang et al. [43] proposed a consensus that improves the traditional PoS consensus mechanism. This mechanism is based on credit rewards and punishments. In this scheme, each node must have an initial stake value of 50 as a representative for generating the next blocks. The behavior of these representatives has been recorded since they joined the mining process. This record will be broadcasted to all nodes through dynamic updates every time. If the adversary activity reaches the threshold value of the invalid block, the consensus removes this node from the mining process. As well, all nodes must give a vote to the generated block, and the node who could not give a vote in the given time interval more than two times will punish. On the contrary, those nodes that can vote more than twice will be rewarded. In this consensus, a new block will be accepted only when the generated block has more transaction value and votes compared to similar blocks. This scheme can alleviate 51%, double-spend, and Sybil attacks on the PoS consensus mechanism by improving the method of selecting miners. It also meets *Nr* and *In* requirements.

Olat and Potop-Butucaru. [44] proposed a novel zero block algorithm that can prevent selfish mining attacks without using timestamps. The average time of the nonce solution process and the time of information dissemination between miners are used to generate a fixed time interval. Honest nodes reject blocks that appear after a fixed time interval and create a dummy block to notify other nodes. This fixed time interval is used to filter blocks and reject block broadcasts after the virtual blocks are released. Even if a selfish miner controls 49% of the hashing power, the probability of deliberate forks is reduced to 0.04. Compared with Hellman, this scheme can resist a large number of selfish mining pools. Therefore, if a selfish miner fails to tamper with the transaction within a given time interval, the possibility of a double suspend attack will be minimized. Therefore, by limiting the time required for legally mining blocks, the *In* and *Nr* requirements of transactions at the consensus layer are satisfied. Generally speaking, these two schemes are impractical to implement in Bitcoin or Ethereum because it requires a lot of changes to the blockchain.

Kim et al. [45] proposed a consensus mechanism based on the Rock-Scissor-Paper (RSP) algorithm. Like traditional manual games, this consensus has three balance states: Rock, Scissor, and Paper. The specifications of the miner's computing device will use to identify the next block miner. Each node that wants to take part in mining activities will be assigned a unique identifier and request to submit the specifications of the computing device with the RSP combinations. The delegated node uses the user RSP combination to generates random RSP combinations. Any malicious activity will result in a loss of
5% of the shares deposited by miners when they first joined. Attacks such as selfish mining and majority attacks have been well alleviated. Besides, the required computing power and maintenance costs are also reduced. But this consensus is prone to other problems. If many computing devices are identical and choose the same RSP combination, the scheme is incapable to select one miner from the list, and any miners will waste their computing power. Also, the integrity of the block will compromise. This consensus does not meet the majority of proposed requirements.

Jaewon and Lim. [46] proposed a random mining group selection technique. This scheme uses the public key hash value of the miner to find the mining group where it belongs using a hash function. Instead of all miners participating in finding the next block, the scheme distributes the hashing power into different groups and randomly assigns the next block to these groups. This reduces the hashing power of the adversary and they will lose the ability to attack. Besides, the hushing power to participate in mining has diminished effectively. Therefore, this scheme lowers the chance of occurring selfish mining, and 51% of attacks. To check the integrity of blocks, all nodes compare the previous hash of the new block to find the block was mined with the proper mining group. We evaluated the scheme and found that randomly selecting nodes for mining purposes was able to mitigate 51% attack and selfish mining, besides, it satisfies *Nr* and *In* from the proposed security requirements at the consensus layer level.

Yang et al. [47] proposed a new solution to mitigate a 51% attack on PoW by increasing the minimum cost to attack. In this scheme, the miner's block frequency historical weight is used to calculate historical weighted difficulty (HWD). The HWD is used to determine whether a branch switch is required to select miners. In the proposed scheme to prevent non-repudiation, miners always sign the new block using their private key. In terms of

preventing attacks, this work is not efficient in protecting the system, it only delays the attack. However, *Nr* is satisfied but the remaining other is not considered.

Table 4.
Comparison of Existing Work Based On Blockchain Security Requirements

Ref	*In*	*Av*	*Nf*	*Cf*
[38]	×	✓	×	×
[39]	×	✓	×	×
[40]	×	✓	×	×
[41]	×	×	✓	×
[42]	✓	×	✓	✓
[43]	✓	×	✓	×
[44]	✓	×	✓	✓
[45]	×	×	×	×
[46]	✓	×	✓	✓
[47]	×	×	✓	×

✓; denotes a corresponding requirement is satisfied,
×; denotes a corresponding requirement is not satisfied

4.3 Smart Contract-Based Attack Solutions

A lot of work is required to update the source code of the smart contract after deployment. The reason behind this difficulty is the immutable nature of the blockchain. Once we deploy a smart contract with some vulnerabilities on the blockchain, we have no chance to modify the contract. Therefore, to detect vulnerabilities and verify the correctness of smart contracts, most research focuses on detecting vulnerabilities and verifying the correctness of smart contracts before deploying smart contracts to the blockchain. In this section, we will review research papers that mainly use fuzzy methods, symbolic execution methods, and formal verification methods to verify the correctness of smart contracts and detect vulnerabilities. To better show the analysis results in Table 4, we summarize the existing smart contract analysis tools and the vulnerabilities that can be detected using these tools.

Fuzzing Method: Liu et al. [48] introduce a fuzzy-based dynamic analyzer to detect the reentrancy bug in the smart contract. This analyzer translates the smart contract to the C++ equivalent code. The scheme identifies reentrancy vulnerability in smart contracts dynamically by leveraging a fuzzy model and flag bugs in code. The execution traces of code are maintained and the reentrancy automata use it for potential bug identification. The tool can only identify reentrancy errors but does not consider the most common smart contract errors. The correctness of the smart contracts with only identifying one bug is not possible to verify. Therefore, this scheme is unqualified to verify the correctness of smart contracts.

Jiang et al. [49] proposed a new static analysis framework for fuzzers, called Contract Fuzzer, to test security vulnerabilities in Ethereum smart contracts. This scheme uses a fuzzy technique to generate input for vulnerability analysis related to application binary interface (ABI) files of smart contracts to determine whether smart contracts are being executed correctly for testing purposes. For certain types of vulnerabilities (such as timestamp dependency), it is prone to high false-positive rates.

In another study, Xiupei et al. [50] have made progress for this framework by utilizing test oracle. Instead of the Black Box approach, this scheme implements a White box approach and shows a high true positive rate. However, these analyzers identified common Ethereum vulnerabilities, i.e., exception disorder, reentrancy, timestamp dependency, dangerous delegate call, and freezing ether vulnerabilities. This framework notifies potential vulnerabilities in the smart contracts at the pre-deployment stage. However, some vulnerabilities may not be detected by this tool.
Therefore, the correctness of the smart contracts is not verified.

Table 5.
Summary of Smart Contracts Analyzing Tools

Ref	Identified Attack	Method
[48]	Reentrancy	Fuzzing method
[49],[50]	Mishandled Exception, Reentrancy, Timestamp Dependency, Delegate call, freezing Ether	Fuzzing Method
[51]	Transaction Ordering Dependency (TOD), Reentrancy, Mishandled Exception,	Symbolic Method
[52]	Transaction Ordering Dependency (TOD), Timestamp Dependency, Mishandled Exception, Reentrancy	Symbolic Method and Formal Method
[53]	Transaction Ordering Dependency (TOD), Timestamp Dependency, Mishandled Exception, Reentrancy	Symbolic Method
[54]	Timestamp Dependency, DoS, Reentrancy	Symbolic Method
[55]	Mishandled Exception, Out of Gas DoS, Reentrancy	Formal Method

Symbolic execution Method: Tsankov et al. [51] proposed a fully automated smart contract analyzer. The framework ensures that the behavior of the contract is safe or unsafe based on the provided security properties. In this scheme, the semantic information of the given smart contract bytecode or source code dependency graph is symbolically encoded to layered Data log, and it is expressed in a designated domain-specific language (Securify language). Compliance and violation patterns are suitable for checking whether the semantic information of the code is safe or unsafe. These patterns efficiently identify transaction order dependency (TOD), Reentrancy, Handling Exception, and Restricted Transfer. This work guarantees the correctness of the contract to a certain extent.

Luu et al. [52] proposed a dynamic smart contract analyzer, called Oyente. Symbolic execution tools are used to improve the operational semantics of the Ethereum Virtual Machine (EVM) to analyze and identify security vulnerabilities in contracts sources code. This scheme identifies transaction-ordering dependence, timestamp dependence, mishandled exceptions, and reentrancy vulnerabilities. But it is highly prone to false-negative results. This false-negative result indicates that this scheme does not have the power to identify the reentrancy bugs at the expected level. Because of this, this scheme has limitations in verifying the Co of the contract. In another study, Zhou et al. [53] extended the Oyente tool aiming to detect the bugs in smart contracts that are not been well identified before. The scheme utilizes static analysis methods and extra patterns. The scheme generates a topology diagram for smart contracts by analyzing it, this helps the developer to understand their contracts deeply. Besides, it is used to provide visualized information of the detected risks in the smart contract. This tool can guarantee the correctness of the contract to a certain extent.

Tikhomirov et al. [54] proposed a static smart contract analysis tool called SmartCheck. This scheme converts the contract source code into an XML-based parse tree. Besides, the translated code checks against specific XPath patterns to identify vulnerabilities. This scheme detects reentrancy, timestamp dependency, and DOS vulnerabilities. However, this scheme cannot define more complex rules correctly. Therefore, the procedure is susceptible to false-positive rates. By identifying these vulnerabilities in the pre-deployment stage of the contracts, this tool guarantees the correctness of the contract to a certain extent.

Formal verification Method: A formal verification tool proposed by Bhargavan et al. [55] able detects three types of vulnerabilities in Ethereum smart contracts, namely, mishandled exceptions, out-of-gas DoS, and reentrancy. The source code or the bytecode of the smart contracts translate to F* code and the scheme use this code to analyze the vulnerable patterns in F*. The shallow embedding and type checking approaches are used for exploring the formal verification of contracts coded in Solidity and EVM bytecode. The formal verification method in this scheme is exploited to analyze contracts, that notify the above three vulnerabilities. However, this tool only detects three code bugs successfully. Therefore, the correctness of the contract verifies to a certain extent.

5 Open Issues and Future Research Directions

Based on the analyses and comparisons listed in Section IV, we have discovered some open issues related to the security of blockchain-based smart contracts, and proposed a series of future research directions.

5.1 Open Research Issues

In this subsection, we describe four open research issues related to the security of blockchain smart contracts.
First, the countermeasures for the security vulnerabilities on each layer are not comprehensive. The researchers did not conduct a comprehensive study but chose to deal with each potential attack in the same layer separately. This problem requires a framework to resolve similar security holes in the blockchain.
Second, the timestamp in the block can be modified by the miner and this issue sometimes motivates the miner to use this advantage. Third, each transaction in the blockchain is queued in sequence in the miner's memory pool follows the principle of sequential token verification. This method allows the adversary to compromise the proposed security requirements and trigger an attack on the contracts. The parallel execution method will be considered here. Fourth, the smart contract vulnerability analysis tool is not mature enough to detect unknown vulnerabilities. Most smart contract analysis tools convert the bytecode or source code of the contract into a compatible language. The converted code may not be the same as the original smart contract. There are indeed some analysis tools that use the source code of the contract. However, they are still prone to false positives. Therefore, the vulnerability scanning technology needs further research.

5.2 Future Research Directions

According to the open issues, we further suggest three research directions on blockchain and smart contracts security.

Parallel mining: Executing transactions parallel at run-time is a better way to implement block creation and verification in transactions. Parallel execution minimizes the chance of attacker miners destroying tokens queued in the memory pool. Transactions will be executed simultaneously instead of waiting in the memory pool.

Design a new language: the process of finding bugs and validating the code's correctness can be made easier by creating a new programming language. Smart contracts are supposed to always return the same outputs for the same input, however, the present programming language paradigm produces unexpected outcomes. Most of the common vulnerabilities we have seen in our research are originated from programming languages' unexpected behavior. Certain types of bugs can be tolerated in object-oriented programming languages. However, a minor bug in the smart contract could result in a multimillion-dollar loss. Because smart contracts interact with digital currency and tokens more frequently. As a result, smart contracts must verify that each activity is secure. Functional programming languages, which emphasize the use of pure functions, or functions that always return the same result for the same input, could be used to construct smart contracts. Smart contracts written in functional programming languages can be analyzed directly without the need to convert them to another language using formal verification methods. As a result, from the beginning, the smart contract code could be rigorously verified with a high level of assurance.

Confidentiality Preservation*:* The confidentiality of the data entered by the user into the smart contract and the smart contract workflow is not considered. Some studies have proposed an off-chain framework to execute smart contracts using a secure network. However, this kind of off-chain code execution does not fully guarantee the security of data and code. For future research, on-chain code execution with confidentiality preservation must be considered.

6 Conclusion

Blockchain has emerged as a perfect platform to smart contract deployment due to its decentralization, immunity, security, and trust. Currently, blockchain-based smart contracts have attracted considerable attention. However, they also confront severe security problems in terms of security because of its openness and decentralization. There are still many issues that have t yet been deeply investigated in academia and industry. In this paper, we performed a thorough survey on the security in blockchain-based smart contracts. We introduced the basic system

model of the blockchain-based smart contract platform. Based on the security model and threat analysis, we further proposed the requirements for a security countermeasure. Taking the requirements as essential criteria, we extensively reviewed the current literature and commented the pros and cons of existing work. Finally, we explored the open issues that have t yet been seriously investigated and proposed a number of research directions to stimulate future efforts.

References

1. N. Szabo. "*The Idea of Smart Contracts*". Accessed: Mar.20,2020. [Online]. Available: http://www.fon.hum.uva.nl/rob/ Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html
2. S. Nakamoto. Bitcoin: "*A Peer-to-Peer Electronic Cash System*". Accessed: Mar. 20, 2020 [Online]. Available: https://bitcoin.org/bitcoin.pdf
3. D. Siegel. "*Understanding the DAO hack for journalist*". Accessed: Mar.20,2020. [Online]. Available: https://medium.com/@pullnews/understanding-the-dao-hack-for-journalists2312dd43e993#. i6eydda9x
4. N. Atzei, M. Bartoletti, T. Cimoli (2017) "A Survey of Attacks on Ethereum Smart Contracts (SoK)". In: *Maffei M., Ryan M. (eds) Principles of Security and Trust. POST 2017. Lecture tes in Computer Science, vol 10204.* Springer, Berlin, Heidelberg
5. M. Alharby, A. Aldweesh and A. v. Moorsel, "blockchain-based Smart Contracts: A Systematic Mapping Study of Academic Research (2018)," *2018 International Conference on Cloud Computing, Big Data and blockchain (ICCBB)*, Fuzhou, China, 2018, pp. 1-6. Page
6. A. Mense and M. Flatscher. 2018. "Security Vulnerabilities in Ethereum Smart Contracts". In *Proc of the 20th International Conference on Information Integration and Web-based Applications & Services (iiWAS2018). Association for Computing Machinery*, New York, NY, USA, 375C380. DOI: https://doi.org/10.1145/3282373.3282419
7. S. Rouhani and R. Deters, "Security, Performance, and Applications of Smart Contracts: A Systematic Survey, " in *IEEE Access*, vol. 7, pp. 50759-50779, 2019.
8. J. Liu and Z. Liu, "A Survey on Security Verification of blockchain Smart Contracts, " in *IEEE Access*, vol. 7, pp. 77894-77904, 2019.
9. D. Vujičić, D. Jagodić and S. Ranđić, "Blockchain technology, bitcoin, and Ethereum: A brief overview," *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2018, pp. 1-6, doi: 10.1109/INFOTEH.2018.8345547.
10. D. Johnson, A. Menezes and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)", *International Journal of Information Security*, vol. 1, no. 1, pp. 36-63, 2001.
11. P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric" in Peer-to-Peer Systems, Springer, pp. 53-65, 2002.
12. "Ethereum Whitepaper | ethereum.org", *ethereum.org*, Accessed: 29-Jul-2020 [Online]. Available: https://ethereum.org/en/whitepaper/.
13. K. Sathish, "The Ultimate Guide to Consensus in Hyperledger Fabric | Skcript", *Skcript*, Accessed: 30- Jul- 2020 [Online]. Available: https://www.skcript.com/svr/consensus-hyperledger-fabric/.
14. Whitepaper.io. *Cardano whitepaper - whitepaper.io*. Accessed: 30 July 2020, [online] Available: https://whitepaper.io/document/581/cardano-whitepaper
15. "Neo Documentation", *Docs.neo.org*, Accessed: 30 July 2020 [Online]. Available: https://docs.neo.org/docs/en-us/index.html.
16. S. Chandel, W. Cao, Z. Sun, J. Yang, B. Zhang, TY. Ni, (2020) "A Multi-Dimensional Adversary Analysis of RSA and ECC in Blockchain Encryption", *Future of Information and Communication Conference*, 2019 https://doi.org/10.1007/978-3-030-12385-7_67
17. V. Buterin what proof of stake is and why it matters. Accessed: Mar. 24, 2020. [Online]. Available: https://bitcoinmagazine.com/articles/what-proof-of-stake-is-and-why-it-matters-1377531463
18. Hyperledger Proof of Elapsed Time (PoET). Accessed: Mar. 26, 2020. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/nightly/0-8/introduction.html#proof-of-elapsed-time-poet
19. D. Larimer. "DPOS Consensus Algorithm—The Missing Whitepaper. Steemit (2018)." (2018).
20. B. Iddo, L. Charles, M. Alex, and R. Meni. "Proof of Activity: Extending Bitcoins Proof of Work via Proof of Stake [Extended Abstract]" *SIGMETRICS Perform.* Eval. Rev. 42, 3 (December 2014), 34C37
21. NEM *Technical Reference*. Accessed: Jun. 2, 2020 [online]. Available: https://nemplatform.com/wpcontent/uploads/2020/05/NEM techRef.pdf
22. N. Kshetri and J. Voas, "Blockchain-Enabled E-Voting," in *IEEE Software*, vol. 35, no. 4, pp. 95-99, July/August 2018, doi: 10.1109/MS.2018.2801546.
23. S. Al-Megren et al., "blockchain Use Cases in Digital Sectors: A Review of the Literature," *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Halifax, NS, Canada, 2018, pp. 1417-1424.
24. E. Eze, S. Zhang and E. Liu, "Vehicular ad hoc networks (VANETs): Current state challenges potentials and way forward", 2014 20th International Conference on Automation and Computing, 2014.
25. "Blockchain Real Estate – How Will Blockchain Change Real Estate?", *Blockgeeks*, 2020. [Online]. Available: https://blockgeeks.com/guides/blockchain-real-estate/. [Accessed: 10- Aug- 2020]
26. K. Mohanta, S. S. Panda and D. Jena, "An Overview of Smart Contract and Use Cases in blockchain Technology," *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bangalore*, 2018, pp. 1-4
27. S. Tanwar, K. Parekh and R. Evans, "Blockchain-based electronic healthcare record system for healthcare 4.0 applications", *J. Inf. Secur. Appl.*, vol. 50, 2020.
28. E. Heilman, A. Kendler, A. Zohar, S. Goldberg "Eclipse Attacks on Bitcoins Peer-to-Peer Network," in *24th USENIX Security Symposium*, Washington, DC, August 2015.
29. S. Yu, "An Overview of DDoS Attacks". In: *Distributed Denial of Service Attack and Defense. SpringerBriefs in Computer Science.* Springer, New York, NY, 2014
30. J. R. Douceur, "The Sybil attack" in Peer-to-Peer Systems, Berlin, Germany: Springer, pp. 251-260, Mar. 2002, [online] Available: http://link.springer.com/chapter/10.1007/3-540-45748-8 24.
31. I. Eyal, E. Sirer, "Majority Is Not Enough: Bitcoin Mining Is Vulnerable". In: *Christin N., Safavi-Naini R. (eds) Financial Cryptography and Data Security*. FC 2014. Lecture Notes in Computer Science, vol 8437. Springer, Berlin, Heidelberg
32. N. Anita and M. Vijayalakshmi., "blockchain Security Attack: A Brief Survey", *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kanpur, India, 2019, pp. 1-6.
33. W. Usman. Chohan,"The Double Spending Problem and Cryptocurrencies", *Available at SSRN*, vol. 3090174, 2017.

34. J. Moubarak, E. Filiol and M. Chamoun, "On blockchain security and relevant attacks, " *2018 IEEE Middle East and rth Africa Communications Conference (MENACOMM)*, Jounieh, 2018, pp. 1-6.
35. A. Manning Solidity Security: Comprehensive list of known attack vectors and common anti-patterns, Accessed: May. 2020, [online] Available: https://blog.sigmaprime.io/solidity-security.html#entropy-vuln
36. A. Bryk, Blockchain Attack Vectors: Vulnerabilities Most Secure Technology, Accessed: may. 2020, [online] Available: https://www.apriorit.com/dev-blog/578-blockchain-attackvectors.
37. S. Pro, Smart Contract Security Issues: What are Smart Contract Vulnerabilities How to Protect, jun. 2020, [online] Available: https://smartym.pro/blog/smart-contract-security-issues-smartcontract-vulnerabilities-and-how-to-protect/.
38. M. Saad, L. Njilla, C. Kamhoua, J. Kim, D. Nyang and A. Mohaisen, "Mempool optimization for Defending Against DDoS Attacks in PoW-based blockchain Systems," *2019 IEEE International Conference on blockchain and Cryptocurrency (ICBC)*, Seoul, Korea (South), 2019, pp. 285-292, doi: 10.1109/BLOC.2019.8751476.
39. K. Wst and A. Gervais, Ethereum eclipse attacks, 2016.
40. M. Yuval, E. Heilman, and S. Goldberg, "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network." IACR Cryptology ePrint Archive 2018.236 (2018). *Proc. 24th USENIX Conf. Security Symp. (SEC)*, pp. 129-144, 2015.
41. P. Swathi, C. Modi and D. Patel, "Preventing Sybil Attack in blockchain using Distributed Behavior Monitoring of Miners," *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kanpur, India, 2019, pp. 1-6, doi: 10.1109/ICCCNT45670.2019.8944507.
42. E. Heilman, "One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner." *International Conference on Financial Cryptography and Data Security*. Springer, Berlin, Heidelberg, 2014.
43. L. Li, J. Yan, H. Peng, and Y. Yang. 2020. "An Improved Consensus Mechanism for the Blockchain Based on Credit Rewards and Punishments". *In Proceedings of the 2020 The 2nd International Conference on Blockchain Technology (ICBCT20)*. ACM, New York, NY, USA, 105C109.
44. S. Solat and M. Potop-Butucaru, "Zeroblock: Preventing selfish mining in bitcoin. CoRR, abs/1605.02435." (2016).
45. D. Kim, R. Ullah and B. Kim, "RSP Consensus Algorithm for Blockchain," *2019 20th Asia-Pacific Network Operations and Management Symposium (APMS)*, Matsue, Japan, 2019, pp. 1-4
46. J. Bae and H. Lim, "Random Mining Group Selection to Prevent 51% Attacks on Bitcoin," *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, Luxembourg City, 2018, pp. 81-82, doi: 10.1109/DSN-W.2018.00040.
47. X. Yang, Y. Chen and X. Chen, "Effective Scheme against 51% Attack on Proof-of-Work blockchain with History Weighted Information," *2019 IEEE International Conference on blockchain (blockchain)*, Atlanta, GA, USA, 2019, pp. 261-265, doi: 10.1109/blockchain.2019.00041.
48. C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," *in Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. Companion*, May 2018, pp. 65C68.
49. B. Jiang, Y. Liu, and W. Chan. "Contractfuzzer: Fuzzing smart contracts for vulnerability detection." *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018.
50. W. K. Chan and B. Jiang, "Fuse: An Architecture for Smart Contract Fuzz Testing Service," *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, Nara, Japan, 2018, pp. 707-708
51. P. Tsankov, A. Dan, D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*., Oct. 2018, pp. 67C82.
52. L. Luu, D.-H. Chu, H. Olickel, P. Saxena and A. Hobor, "Making smart contracts smarter," *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, pp. 254-269, Oct. 2016.
53. E. Zhou et al., "Security Assurance for Smart Contract", *2018 9th IFIP International Conference on New Techlogies, Mobility and Security (NTMS)*, Paris, 2018, pp. 1-5, doi: 10.1109/NTMS.2018.8328743.
54. S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko and Y. Alexandrov, "SmartCheck: Static Analysis of Ethereum Smart Contracts", *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Gothenburg, Sweden, 2018, pp. 9-16.
55. K. Bhargavan, A. Delignat-Lavaud, C. Fournet, et al. "Formal Verification of Smart Contracts: Short Paper". *In: PLAS 16. ACM. ACM; 2016;* NewYork, NY, USA: 91C9