Hindawi
Security and Communication Networks
Volume 2022, Article ID 5307697, 11 pages
https://doi.org/10.1155/2022/5307697

Research Article

Multiple-Layer Security Threats on the Ethereum Blockchain and Their Countermeasures

Li Duan,[1,2] **Yangyang Sun**,[1] **Kejia Zhang**,[3,4] **and Yong Ding**[2]

[1]*Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China*
[2]*Guangxi Key Laboratory of Cryptography and Information Security, Guilin, Guangxi, China*
[3]*School of Mathematical Science, Heilongjiang University, Harbin 150080, China*
[4]*Cryptology and Cyberspace Security Laboratory of Heilongjiang University, Harbin 150080, China*

Correspondence should be addressed to Li Duan; duanli@bjtu.edu.cn and Kejia Zhang; zhangkejia.bupt@gmail.com

Received 19 November 2021; Revised 10 January 2022; Accepted 20 January 2022; Published 14 February 2022

Academic Editor: Yuling Chen

Blockchain technology has been widely used in digital currency, Internet of Things, and other important fields because of its decentralization, nontampering, and anonymity. The vigorous development of blockchain cannot be separated from the security guarantee. However, there are various security threats within the blockchain that have shown in the past to cause huge financial losses. This paper aims at studying the multi-level security threats existing in the Ethereum blockchain, and exploring the security protection schemes under multiple attack scenarios. There are ten attack scenarios studied in this paper, which are replay attack, short url attack, false top-up attack, transaction order dependence attack, integer overflow attack, re-entrancy attack, honeypot attack, airdrop hunting attack, writing of arbitrary storage address attack, and gas exhaustion denial of service attack. This paper also proposes protection schemes. Finally, these schemes are evaluated by experiments. Experimental results show that our approach is efficient and does not bring too much extra cost and that the time cost has doubled at most.

1. Introduction

In recent years, with the rapid development of blockchain technology, the application scenarios of blockchain have not only been limited to digital currency and financial fields but have gradually been deeply integrated with all walks of life [1, 2], such as smart city and Internet of things (IoT). In 2008, Satoshi Nakamoto released his famous Bitcoin whitepaper [3], which first put forward the concept of "blockchain." Blockchain is a new distributed computing and storage paradigm which integrates many existing technologies. It uses cryptography principle and timestamp technology in data layer to ensure the immutability of data, uses peer-to-peer network to communicate data in network layer, uses distributed consensus algorithm to maintain the consistency of data in the consensus layer; uses scripts and algorithms to implement smart contracts in contract layer; and uses Turing complete virtual machine to realize various functions in the application layer. Compared with traditional databases, blockchain, as a distributed database, requires multiple nodes to maintain data together, which requires data consistency and business fairness.

At the end of 2013, Vitalik Buterin, founder of Ethereum, released the first edition of Ethereum White Paper [4], which realized the development of smart contracts with Turing's complete programming language. From then on, blockchain application was no longer limited to the currency field, and the blockchain 2.0 era started. As an open source public chain platform, Ethereum's function of supporting smart contracts will help its development. Smart contract is a representative technology in the blockchain 2.0 era, and its concept was put forward by cryptographer Szabo [5] as early as the end of the 20th century. He defined smart contract as a set of promises defined in digital form, and the participants of the contract can implement these promises on machines. It was limited to the science, technology, and environment at that time, and it was not until the birth of Ethereum that it gradually revived.

Blockchain technology, as a new technology, technically ensures transaction security through encryption algorithm and digital signature, and relies on consensus mechanism to generate blocks to form a chain structure to ensure that data cannot be tampered with. Nevertheless, blockchain is still facing great security threats [6], especially in smart contracts. Because of the differences in the programming ability of smart contract developers, security problems are inevitable. On June 18, 2016, hackers maliciously attacked The DAO project, resulting in the theft of 3.6 million Ether and the loss of nearly 100 million funds. On July 20, 2017, hackers exploited the contract loophole of Parity Multi-Signature Library, resulting in the freezing of over 500,000 Ether in 587 wallets and a loss of about RMB 220 million. In April 2018, nearly RMB 6 billion was stolen by hackers due to integer overflow loopholes in the contract code of American Chain BEC project, which reduced the market value of tokens to almost zero. In 2019, the global blockchain lost more than $6 billion due to security incidents. In 2020, the blockchain was hacked incurring a loss of nearly $3.8 billion.

Therefore, it is meaningful to study the security attacks of blockchain, especially to study the security threats of smart contracts that cause the greatest losses, which is helpful to improve the security level of the Ethereum blockchain. The contributions of this paper are summarized as follows:

(i) We introduce the background of various attacks, and analyze the principles and attack paths of ten kinds of security threats on Ethereum.

(ii) We construct several specific attack scenarios, and propose the protection schemes corresponding to Ethereum attacks.

(iii) We test and evaluate the proposed protection schemes. Finally, a demonstration system is built to demonstrate the multiple attack scenario.

The rest of this paper is organized as follows. Section 2 reviews the related work. The backgrounds and architecture of the Ethereum blockchain are introduced in Section 3. In Section 4, we study the principle of ten kinds of attacks on Ethereum. In Section 5, we explore the corresponding protection schemes. The protection schemes are evaluated in Section 6. In Section 7, this paper is concluded.

2. Related Work

Aiming at the security threats in blockchain scenarios, the existing research work mainly focuses on attack discovery and attack protection.

2.1. Attack Discovery. From the perspective of attack discovery, the detection methods based on symbol execution, fuzz testing, taint analysis, and formal verification are used to monitor the security threats of contract generation, release, and execution, and to detect the potential risks and vulnerable paths in the process of contract interaction. Luu et al. [7] proposed a detection method based on symbolic model to monitor the security threats in the whole process of contract generation-release-execution in real time, and to detect the potential risks in the process of contract interaction and the vulnerability of accurate location of vulnerable paths. Its design is fully modularized, allowing advanced level users to execute and plug in their own identification logic to check self-defined properties in their smart contracts. In addition, there are many automated detection tools, for example, teEther [8], Securify [9], ZEUS [10], EasyFlow [11], and SmarTest [12]. There are many detection tools at present, but they are difficult to be widely used. Tu et al. [13] proved that the detection efficiency is not high, and there are fewer vulnerabilities that can be detected. We can combine traditional detection methods with machine learning to improve the versatility and efficiency of detection tools to a certain extent.

In addition, Hou et al. [14] put forward the method of deep reinforcement learning by analyzing the behavior of associated users, and automatically discovering the attack of consensus strategy. Li et al. [15] proposed an improved selfish mining based on hidden Markov decision processes to maintain the benefit from selfish mining. Li et al. [16] focused on the validity of semi-selfish mining attacks considering the probability of being detected. Marcus et al. [17] put forward a method of solar eclipse attack on Ethernet network with very few resources.

2.2. Attack Protection. From the perspective of attack protection, based on multi-signature, Byzantine consensus virtual layer design, and safe miner selection, the problems of DoS attack and currency age attack in blockchain operation are solved. Li et al. [18] proposed a cross-chain system based on multiple signatures, which can ensure the credibility of trading groups by locking assets and resisting DoS attacks at the same time. Sonnino et al. [19] proposed a cross-ledger consensus protocol based on Byzantine consensus mechanism to resist cross-ledger replay attacks. Li et al. [20] used the amount of coins to select miners and limit the maximum value of currency age to fight against currency age attacks, thus improving the robustness of the system. Wang et al. [21] proposed a secure inter-chain transmission protocol, which can effectively resist the double-flower attack by recording the asset transmission process between multiple chains and ensuring its consistency. Luu et al. [22] put forward the verifier dilemma problem, that is, after the nodes participating in the consensus pay a lot of computing power to verify the transaction, if they do not get the bookkeeping right, the verifier will face the dilemma of paying more computing power to verify or accepting the wrong script, so as to solve the double-flower attack problem. Luu et al. [23] put forward a secure fragmentation protocol which can be used to build public chains, and the analysis proves that this scheme can effectively improve the system throughput. Nguyen et al. [24] proposed an approach and a tool, called SGUARD, which automatically patches vulnerable smart contracts.

As a complex system, blockchain faces security threats from the data layer to the application layer. At present, the related work of blockchain security attack and protection is mainly discussed from the attack as a whole, but not the

specific attack. However, this paper goes deep into the details, investigates the security problems faced by Ethereum at all levels, studies and tests several typical attacks existing in smart contracts, and proposes protection schemes.

3. Background

The related technologies and background knowledge involved in this section are introduced, including Ethereum architecture, memory layout, and transaction process.

3.1. Ethereum. Ethereum is an open-source public chain platform for executing intelligent contracts through Ethereum virtual machines. These machines execute intelligent contracts by consuming Ethereum coins. The concept of Ethereum first appeared at the end of 2013. Inspired by Bitcoin, Vitalik Buterin, founder of Ethereum, released the first edition of Ethereum white paper, which realized the development of intelligent contracts with Turing's complete programming language.

As of November 2021, the market value of Ethereum has exceeded $570 billion, which is the second highest cryptocurrency in market value after Bitcoin with $1.27 trillion. Ethereum is often described as "the computer of the world." From the point of view of computer science, Ethereum is a deterministic but unbounded state machine, which has two basic functions, the first is a globally accessible singleton state, and the second is a virtual machine that changes the state. It uses blockchain to synchronize and store the state of the system [25], and cryptocurrency called Ether is used to calculate and limit the execution resource cost. Ethereum developers can write intelligent contracts and build decentralized applications that can run on Ethereum virtual machines. While ensuring stable and normal operation, it can also reduce or eliminate examination procedures, and save resources and reduce risks by eliminating the participation of third parties.

3.2. Ethereum Architecture. As Figure 1 shows, Ethereum architecture is composed of five layers [26], namely, the data, network, consensus, contract, and application layers. Ethereum system runs on these five layers. The data layer includes technical elements such as data block, chain structure, hash function, asymmetric encryption, timestamp, Merkle tree, etc., which ensures the reliability and stability of Ethereum data [27]. The network layer specifies the peer-to-peer network, wherein each node can obtain the updated status of blockchain from some active nodes. There is no central server, and only the nodes exchange information fairly. The consensus layer ensures the consistent state of the blockchain. At present, Ethereum adopts the Proof of Work (PoW) consensus mechanism. But in the future version planning, Ethereum consensus mechanism will gradually transition to Proof of Stake (PoS) mechanism [28]. This design can speed up the transaction and save the resource consumption [29]. It is also effective to avoid the disadvantage of unfair initial equity distribution existing in the simple equity proof mechanism. The contract layer

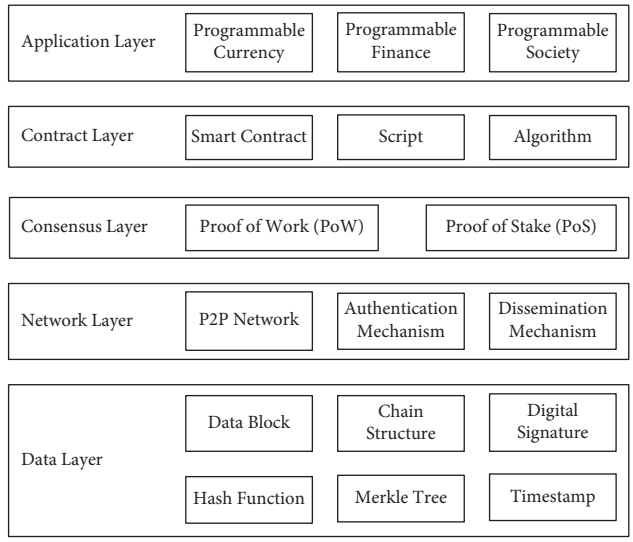

Figure 1: Ethereum's hierarchical structure.

encapsulates various scripts, algorithms, and smart contracts, so that various instructions can be executed automatically and determinately. Smart contract is executed in Ethereum virtual machine at a certain cost of gas according to different instructions. A smart contract is also an Ethereum account, which we call a contract account (CA). This means that they have a balance and they can trade through the network. But they cannot be manipulated by humans. They are deployed on decentralized network nodes and run as programs. Individual users can interact with the smart contract by submitting a transaction to execute a certain function. Smart contracts can define rules like regular contracts and automatically enforce them through code. The application layer encapsulates various application scenarios and cases. For example, various blockchain applications built on Ethereum are deployed in the application layer. And, it is the basis for the realization of a programmable society in the future. Finally, some corresponding components are needed to serve these five layers, which is called external necessary environment, such as web user interface interacting with applications, database for storing blockchain data, cryptographic mechanism supporting consensus protocol, etc. [30].

3.3. Memory Layout of Ethereum. Ethereum virtual machine (EVM) is Turing-complete, and its operations are limited by the number of gas provided by users for each transaction. The implementation of the Ethereum virtual machine is based on the stack. Unlike traditional computers, all instructions of Ethereum virtual machine are executed on the stack, and the parameters or operation results required by the instructions can be obtained through the stack operation. The maximum depth of the Ethereum virtual machine stack is 1024, and the size of each data unit in the stack is 256 bits, which is convenient for executing Keccak-256 elliptic curve hashing algorithm. There are two main storage models in the Ethereum virtual machine, namely, temporary memory

model and permanent storage model. Temporary memory of virtual machine is a simple byte array based on word addressing, which is similar to the concept of traditional computer memory, and its storage is unstable. Unlike temporary memory, permanent storage is a word array based on word addressing. As a part of the system state, permanent storage will be maintained in real time, and it is more stable than its temporary memory. In the initial state, both the data in the temporary memory area and the data in the permanent storage area of Ethereum virtual machine are initialized to 0. Storage stores data through key-value pairs, which maps 32-byte keys to 32-byte data. Global variables in smart contracts are stored in the storage area, and their storage location is determined by the type of the variable and its position in the code. If a variable store is less than 256 bits, Ethereum virtual machine may store multiple variables in one slot. Or the mapping type occupies a slot in which the mapping or array length is stored. The specific locations of arrays and mapping elements are stored in slots according to a set of special hash rules.

3.4. Transaction Process of Ethereum. The trading process of Ethereum is as follows. (1) The sender constructs a transaction and digitally signs it. (2) The sender calls api through JSON-RPC to submit the signed transaction to the Ethereum client. (3) After verifying the received transaction, the Ethereum client broadcasts it to Ethereum point-to-point network. (4) Any client that receives the transaction information will add the transaction to its transaction pool if the client is also a miner. (5) The miner executes a series of transactions selected from its trading pool, creates a new block, and updates the status of the block chain. There are three types of transactions. For transfer transactions, the specified amount needs to be updated and transferred from the sender's account to the receiver's account or contract account. For contract deployment, enter a bytecode to create a new contract account and associate it with the entered bytecode. For contract call, where the recipient is the called smart contract, the input uniquely identifies the callee function through the hash digest algorithm, and the bytecode associated with the called smart contract account is loaded into the Ethereum virtual machine for execution. (6) Miners solve the problem of workload proof by looking for a random nonce value. The hash value of metadata of this new block needs to be smaller than a certain value, which reflects the difficulty of creating the block. Unlike Bitcoin's computationally intensive workload, Ethereum uses a memory-intensive problem called "Ethash." (7) When creating the block, the miners broadcast it to the point-to-point network of Ethereum, so that other clients can verify the block. (8) When other Ethereum clients verify a new block, the client will add the block to the blockchain.

4. Security Attacks on Ethereum

Attacks on Ethereum can be divided into five layers: application layer, contract layer, consensus layer, network layer, and data layer attacks. In this paper, we focus on the attacks in the application layer, the contract layer, and the network layer. The other two layers of attacks are our future research directions.

4.1. Attacks of Application Layer. The application layer is the carrier of blockchain technology and provides solutions for various business scenarios. Security vulnerabilities in various trading platforms and user accounts seriously threaten the asset security of blockchain wallet users. Therefore, we analyze three common types of attacks.

4.2. Replay Attack. The replay attack is to replay transaction information. The user signs a message, uploads it to the contract, and then verifies the signature inside the contract. But since the user's signature information is online, everyone can get it. When verifying the user's signature in the contract, if the signed message does not include variables that change randomly with the number of transactions, such as timestamp, nonce, etc., the attacker will hold the user's signature and forge transactions, thereby obtaining a profit. It can be widely understood as the process of using the same payment information to purchase goods multiple times. When the Ethereum and Ethereum Classic chains emerged after the hard fork, it was found that transactions on the Ethereum chain were still valid when they were replayed on the Ethereum Classic chain. As Figure 2 shows, while parameters remain unchanged, multiple transfers can be made through the replay attack.

4.3. False Top-Up Attack. The status field in the Ethereum token transaction receipt is true or false depending on whether an exception is thrown during the execution of the transaction. When the user calls the transfer function of the token contract to transfer, if the transfer function runs normally and no exception is thrown, the status of the transaction is true. If digital currency exchanges, wallets, and other platforms have flaws in determining whether tokens' recharge transactions are successful, it will lead to serious false top-up attack. As Figure 3 shows, when balances[msg.sender] < _value, it enters the else logical section and returns false, and finally no exception is thrown. In this attack, although the exchange did not receive the real tokens, the transaction execution did not throw an exception, and the user did get the real recharge record. In this case, users can steal real assets. The false top-up attack has become a type of attack that cannot be ignored in blockchain system.

4.4. Transaction Order Dependence Attack. Transaction order dependence attack is a kind of attack that widely exists in the blockchain system; an example of transaction order dependence attack is shown in Figure 4. In blockchain, transactions initiated by nodes need to be packaged by miners before they can be finally recorded on the blockchain. Miners select a series of transactions from the trading pool and then package them into a new block. According to

```
function transferProxy(address _from, address _to, uint256 _value, uint256 _fee,
    uint8 _v, bytes32 _r, bytes32 _s) public returns (bool){
    bytes32 h = keccak256(_from,_to,_value,_fee);
    if(_from != ecrecover(h,_v,_r,_s)) revert();
}
```

Figure 2: Replay attack.

```
function transfer1(address _to, uint256 _value) returns (bool) {
    if(_value <= balance[msg.sender] && _value > 0)
    {
      balance[msg.sender] -= _value;
      balance[_to] += _value;
      return true;
    }
    else
        return false;
}
```

Figure 3: False top-up attack.

```
event Purchase(address _buyer, uint256 _price);
event PriceChange(address _owner, uint256 _price);
modifier ownerOnly() {
        require(msg.sender == owner);
        _;
}
function TransactionOrdering() {
        owner = msg.sender;
        price = 100;
}
function buy() returns (uint256) {
        Purchase(msg.sender, price);
        return price;
}
function setPrice(uint256 _price) ownerOnly() {
        price = _price;
        PriceChange(owner, price);
}
```

Figure 4: Transaction order dependence attack.

miners' criteria for transaction selection, miners will generally choose the transaction fees to sort and package in order to get the maximum benefits. Therefore, the sequence of a series of transactions packaged in the block is not the same as the sequence of transaction generation but is also related to the gas cost consumed by the transaction. Therefore, the contract code cannot know the order of transactions. And, the transaction is visible to each node in the transaction pool, so its execution order can be observed.

The attacker observes the transactions that may contain the target contract in the pool. If they exist, the status of the contract that is not conducive to the attacker or the authority of the contract will be modified by the attacker. Attackers can also steal transaction data, create their own transactions at a higher gas price, and then package their own transactions in the block before the original transaction, thus obtaining transaction processing priority. In Ethereum geth client, txpool consists of two parts, namely, pending queue and queued queue. When the sending transaction Nonce is greater than the completion transaction nonce+1, the transaction will be queued, and if the current sending transaction nonce is equal to the completion transaction nonce + 1, the transaction will be placed in pending waiting for packaging.

4.5. Attacks of Contract Layer. As an indispensable part of blockchain technology, smart contract not only expands the application of blockchain technology but also increases the attack surface faced by the blockchain system. The smart contract is written in a high-level language like solidity, and then the contract will be compiled into bytecode, which will be deployed to the blockchain by the contract owner and run on various virtual machines similar to Ethereum virtual machines. In the process, the smart contract will face various security threats [31].

4.6. Integer Overflow Attack. Integer overflow is a typical loophole in the blockchain system, which once caused serious economic losses in the development of blockchain. In the Ethereum platform, Solidity language is the most mainstream language for writing intelligent contracts. Because of the insecurity of its design, integer overflow is a serious problem. Generally speaking, integer overflow can be divided into integer overflow and integer underflow. According to arithmetic classification, there are three overflow problems: multiplication overflow, addition overflow, and subtraction overflow. In April 2018, nearly RMB 6 billion was stolen by hackers due to integer overflow loopholes in the contract code of the American Chain BEC project, which reduced the market value of tokens to almost zero. In the same month, hackers used the integer overflow vulnerability of SMT project side to create a huge amount of SMT currency for selling, and the Firecoin Exchange suspended the recharge and withdrawal of all other currencies for this purpose.

In Solidity, the variable supports unsigned integers, and the value after uint represents the number of bits occupied by its unsigned integers in storage, and supports 8-bit unsigned integers to 256-bit unsigned integers. An unsigned integer of type uint8 stored in the range of 0 to $2^8 - 1$, that is, [0, 255], and an unsigned integer of type uint256 stored in the range of 0 to $2^{256} - 1$. Because the range of stored integers from uint8 to uint256 is limited, and the range of represented integers is also limited, there is an overflow problem. The integer overflow attack is shown in Figure 5. When balances[msg.sender] < _amount, it results in an underflow.

4.7. Re-Entrancy Attack. Re-entrancy attack is a typical attack in Ethereum, which directly led to the hard bifurcation of Ethereum. The main reason for the attack is the sequencing and atomicity of updating smart contract

```
function withdraw(uint _amount) {
        balancesAndAmount(balances[msg.sender], _amount);
        require(balances[msg.sender] -_amount > 0);
        msg.sender.transfer(_amount);
        balances[msg.sender] -= _amount;
}
```

Figure 5: Integer Overflow attack.

variables and transferring operations, the re-entrancy attack is shown in Figure 6. When the logic in the smart contract code adopts the sequence of transferring operation first and then modifying the variable value, the attacker can construct a smart contract with the malicious callback function. If the object of the transfer operation is a malicious contract, it can lead to recursively calling the contract, destroying the original business logic of the contract, and bypassing its inspection to obtain additional transfer income.

By default, the Ethereum smart contract has an unnamed callback function, which has no parameters or return values. If no function can be found in the calling contract to match the hash of the provided function, the callback function will be called. When the contract receives a transfer without data, it will also call the callback function. In addition, in order to receive Ether, the callback function must be marked as payable. If it is not marked as payable, the contract can only receive Ether by calling other functions marked with payable. Imagine such a scenario, if a special callback function is constructed, in which the transfer function of the other party is called, then a recursive transfer will be generated, and the contract with loopholes will continuously transfer money to the special contract until the gas is exhausted. It should be noted that this attack is only aimed at the transfer method of address.call.value () in Ethereum solidity.

4.8. Honeypot Attack. Honeypot contracts are the most interesting findings. These contracts hold ether, and pretend to do so insecurely. In short, they are scam contracts that try to fool us into thinking we can steal the ether they hold, while in fact all we can do is lose ether. As Figure 7 shows, CryptoRoulette is a type of honeypot attack. The variable of game is not initialized, so it by default points to the first location of the contract storage space, and then stores the caller's address here. The submitted number is stored in the second location. In fact, the variable of secretNumber eventually is overwritten by the address of the caller's. A common pattern they follow is, in order to win the ether they hold, we must send them some ether of our own first. However, if we try that, we are in for a nasty surprise: the smart contract eats up our ether, and we find out that the smart contract does not do what we thought it would.

4.9. Short Url Attack. Short url attack is a typical attack in Ethereum, which usually occurs in exchanges. In Ethereum virtual machine, the data end of the input will be automatically filled with 0. Malicious attackers can use an address account with the end of 0, and the exchange fails to verify the address length

```
contract Victim{
    function withDraw(){
        uint amount = userBalannce[msg.sender];
        if (amount > 0) {
            msg.sender.call.value(amount)();
            userBalannce[msg.sender] =0;
        }}
}
contract Attacker{
    function() payable{
        test++;
        Victim(msg.sender).call(bytes4(keccak256("withDraw()")));
    }
}
```

Figure 6: Re-entrancy attack.

```
struct Game {
    address player;
    uint256 number;
}
Game[] public gamesPlayed;
function shuffle() {
    secretNumber = uint8(sha3(now, block.blockhash(block.number-1)) )% 10 ;
}
function play(uint256 number) payable public {
    require(msg.value >= betPrice && number <= 10);
    Game game;
    game.player = msg.sender;
    game.number = number;
    gamesPlayed.push(game);
    if (number == secretNumber) {
        msg.sender.transfer(this.balance);
    }
    shuffle();
    lastPlayed = now;
}
```

Figure 7: Honeypot attack.

input by the user, which causes the transferred related variables to shift and enlarge, thus expanding the actual transfer amount by several times, and malicious attackers can obtain a large amount of benefits. There are two main reasons for this vulnerability; one is that the exchange has not verified the incoming address length of the user, and the other is that the Ethereum virtual machine has an automatic completion mechanism for the data whose length does not conform to the specification when calling the smart contract, resulting in the shift amplification of parameters. We can use sendRawTransaction() to achieve this attack and the code is shown in Figure 8.

4.10. Airdrop Hunting Attack. The airdrop hunting attack uses multiple new accounts to call the airdrop function in order to obtain airdrop coins, and attackers transfer them to

```
mapping (address => uint) balances;
event Transfer(address indexed _from, address indexed _to, uint256 _value);
function transfer(address to, uint amount) public returns(bool success) {
    if (balances[msg.sender] < amount) return false;
    balances[msg.sender] -= amount;
    balances[to] += amount;
    emit Transfer(msg.sender, to, amount);
    return true;
}
```

Figure 8: Short URL attack.

their account to achieve wealth accumulation. This attack is relatively common that as long as it is a contract with an airdrop function, it can make multiple profits. The first automated attack was the Simoleon contract. As Figure 9 shows, the contract was designed to give some amount of ether to initialized an account, so the attacker thinks that we can create a few more accounts to get rewards, then transfer all the money to one account. The attacker write attack the contract and create many temporary contracts, and call this function in these contracts.

4.11. Writing of Arbitrary Storage Address Attack. The attack of arbitrary memory address writing is a common and harmful attack in the blockchain system. The attack can cause malicious users to write and overwrite any storage variable in the smart contract. In Ethereum, the state variables of intelligent contracts will be stored in the storage area, which is an important and open contract storage space. Generally speaking, contract developers will set strict access control to the global variables stored in the storage area to ensure the security of contracts. Storage key-value pair mapping is used to store data. If the user can arbitrarily control the key value of storage when writing, he or she can modify any storage variable value, so as to avoid all the related detection operations in the contract that uses the state variable value to check the authority, and thus achieve the purpose of improving the authority. In addition, because the attacker can use this vulnerability to destroy the contract storage structure, and perform any variable overwriting operation, such as overwriting the value of the state variable storing the address of the contract owner, this may cause abnormal execution of contract functions, freezing of funds, and other hazards. Since the required guard is invalid, the contract owner can try to underflow the array size by executing the code of Figure 10 when the array length bonusCodes is 0. Therefore, we can write to any location in the storage arbitrarily.

4.12. Attacks of Network Layer. The network layer is the most basic technical architecture in the blockchain system. It encapsulates the blockchain system's networking methods, message dissemination mechanisms, authentication mechanisms, etc., so that the blockchain has decentralized and nontamperable characteristics. But these features also

```
function transfer(address _to, uint256 _amount) returns (bool success) {
    initialize(msg.sender);
    if (balances[msg.sender] >= _amount
        && _amount > 0) {
        initialize(_to);
    } else {
        return false;
    }
}
function initialize(address _address) internal returns (bool success) {
    if (_totalSupply < _cutoff && !initialized[_address]) {
        initialized[_address] = true;
        balances[_address] = _airdropAmount;
        _totalSupply += _airdropAmount;
    }
    return true;
}
```

Figure 9: Airdrop hunting attack.

```
function PopBonusCode() public {
    require(0 <= bonusCodes.length);
    bonusCodes.length--;
}
function UpdateBonusCodeAt(uint idx, uint c) public {
    require(idx < bonusCodes.length);
    bonusCodes[idx] = c;
}
```

Figure 10: Writing of arbitrary storage address attack.

provide convenience for attackers who can easily launch a DoS attack. The purpose of the attack is to make users temporarily or permanently unable to use these services provided by the smart contract.

4.13. Gas Exhaustion Denial of Service Attack. According to the design of Ethereum, when the smart contract is deployed or the function in the smart contract is called, the execution of the contract code needs a certain amount of gas to ensure that the calculation is completed completely. At the same time, the Ethereum system limits the maximum total amount of gas consumed by each block, and the total amount of gas of all transactions in the block cannot exceed the maximum total amount of gas in this block. Once an operation in an intelligent contract consumes a lot of gas, resulting in the consumed gas value reaching the maximum total amount of gas in the block, the operation will not be successfully executed, and all processes depending on the operation will fail, so the contract cannot normally complete other functions, resulting in a denial of service state. As Figure 11 shows, transferring money to everyone at once is likely to result in reaching the gas limit of ethereum blocks. Usually, this denial of service attack occurs when a contract

```
address public owner;
address[] investors;
uint[] investorTokens;
function invest() public payable {
        investors.push(msg.sender);
        investorTokens.push.(msg.value * 5);
}
fuction distribute() public {
        require(msg.sender == owner);
        for(uint i = 0, i < investors.length; i++) {
                transferToken(investors[i],investorTokens[i]);
        }
}
```

Figure 11: Gas exhaustion denial of service attack.

developer does not consider the block gasLimit and introduces the operation of modifying dynamic data structure variables such as arrays whose size will change with time. After a block is mined, an attacker can issue multiple transactions at a higher gas price immediately, and then use the above operations of the contract to consume the gas limit of the whole block, so that the block does not contain any other transaction before a certain time, thus preventing other users from using the functions of the contract normally.

5. Security Protection Schemes

In this section, we propose the protection schemes against the ten attacks mentioned in the previous section. The details follow.

5.1. Protection Schemes of the Application Layer. We can prevent the replay attack in the following ways: (1) Avoiding using the transferProxy function and using a more secure signature method. (2) Adding variables such as nonce, timestamp, etc. The nonce generation algorithm does not adopt the design of self-increment from 0 to avoid the same value as other scenarios. (3) Adding address (this) in keccak256(). (4) Adding chainID, which is the blockchain's name.

To prevent the false top-up attack, we judge not only transaction success but also whether the balance of the top-up wallet address increases accurately. This judgment can be made through the Event log. Many centralized exchanges, wallets, and other service platforms obtain the transfer amount and judge the accuracy of the transfer through Event logs. However, we need to pay special attention to the evil situation of the smart contract, because the Event can be written arbitrarily, and it is not a mandatory default option that cannot be tampered with. The required and asserted methods can also be used that an exception will be thrown directly to interrupt the execution of the subsequent instructions of the contract when the conditions are not met.

The protection of transaction order dependence attack is a very complicated process. For the ERC20 transaction order dependence attack that happened once, it only needs the contract developers to pay attention to this problem and follow the best programming practices. For the attack scenario constructed in this example, this problem is not the problem of the contract developer, but the problem of the Ethereum system itself. At present, the better solution is to confuse transactions, such as hiding transactions as internal transactions, and so on.

5.2. Protection Schemes of Contract Layer. For the problem of integer overflow, we can consider the results of each step by setting up a complete inspection mechanism, but this method is difficult and cumbersome, and it is not universal. Therefore, OpenZeppelin provides SafeMath [32] in an intelligent contract function library, which can effectively prevent integer overflow. There are two ways to use the SafeMath library. The first one is to use the library functions directly, such as SafeMath.add(a,b). The other is that library functions can be called after using SafeMath for unit. For example, a.add(b) means that add(a,b) in safemath library has been executed.

For the protection of re-entrancy attack, the most fundamental solution is to update all the states that should be changed in advance before the transfer, instead of updating them after the transfer, which depends on the smart contract developers to follow the best practices. In addition, it is also an idea to use other transfer methods instead of the msg.sender.call.value() function. For the designed attack scenarios, we use these two methods to test them, respectively. For the first method, we put the change in account balance before the transfer, and then judge whether the transfer is successful or not, and if the transfer is not successful, restore the balance of the user's account. In this way, the code re-entrancy attack is successfully prevented, and the protection scheme is effective. For the second scheme, we use the transfer() function to replace the msg.sender.call.value() function, which can also prevent the re-entrancy attack. The above two schemes can well prevent a re-entrancy attack, but the best scheme is the first one, which updates the status first and then transfers money.

Honeypot contracts are diverse and unpredictable. For the CryptoRoulette attack, we can clearly use memory or storage for variables. We can also use the new version of the compiler with version 0.5.0 and later where this problem has been solved by the system because smart contracts with uninitialized storage variables cannot be successfully compiled. Finally, we remind everyone that some people use Ethereum smart contracts to cheat. They fully figure out the psychology of some people's greed for small profits, and throw out some seemingly handy bait, then run away after having enough users. Because these creators spend for fees to create these contracts, they have a purpose that putting a certain amount of ether can get all the balance of the account, so it is definitely arbitrage. Publishing the source code on Github also uses various tricks to make people not find loopholes in a short time, thus encouraging users to enter the trap.

Short url attack protection only needs the exchange to increase the address length check at the client. In addition, for contract developers, the web3 interface used has already fixed the vulnerability. When users call the contract with web3, if they find that the data length is insufficient, they will not add 0 at the end, but add 0 at the beginning of the field, which effectively prevents the short url attack. In a word, the protection of this vulnerability mainly depends on two parts, one is that the client actively checks the address length, the other is that the parameter format check is added at the web3 level. Although this vulnerability can be reproduced at the virtual machine level of Ethereum, there will be no problem in the actual application scenario of the blockchain.

To prevent an airdrop hunting attack, we can set permission control for the airdrop function. For example, only the contract creator can distribute tokens to target addresses. Or only externally owned accounts can receive airdrop rewards, and contract accounts cannot participate.

For any memory address write attack, this attack is rare, and it is often the result of many factors. Therefore, the protection of this attack can be achieved by the contract developers following the best practices. In the development of contracts, developers need to pay attention to dynamic arrays. Errors in the processing of dynamic arrays may lead to contract loop-holes in an unobvious way. Therefore, in unnecessary cases, dynamic array is not used, which can effectively avoid this attack.

5.3. Protection Schemes of the Network Layer. Gas exhaustion denial of service attack protection also depends on the best practices of contract developers. The size of the gas consumed by different instructions is not certain. By debugging the attack scenario, it is found that the sload instruction was executed in the loop, consuming 800 gas. However, the operation with high gas consumption is usually to operate the data in the storage area, so the contract developer should try not to operate the data in the storage area in the loop. Besides, we can also add an end mark of the loop in the execution.

6. Program Evaluation

6.1. Experiment Setup. The private Ethereum blockchain is deployed on Alibaba cloud server, which has 4 processors with 8 GB RAM and 200 GB hard Disk, and each processor has 2 cores. The server is running with Ubuntu 18.04. Smart contracts are written by Solidity programming language.

6.2. Experiment Processes. Based on the above configuration, we implemented ten defense methods as mentioned in the previous section. To analyze their efficiency, we tested time cost of 50, 100, 150, 200, 250, and 300 transactions. The experimental results are shown in Figure 12.

6.3. Result Analysis. In Figure 12, we find the most time-consuming is the replay attack's protection scheme, followed by the short url attack's protection scheme. The replay attack's protection scheme adds signing and signature verification, which need to perform complex calculations, so it is the most time-consuming. For the short url attack's protection scheme in web3, if the data length is insufficient, it will add 0 at the beginning of the field. First judge the address length, if the length is less than 40 bit, then web3 calls a function to automatically complement the address which is time-consuming. Even with these two schemes, the time cost has doubled at most. The time cost of the other eight protection schemes is roughly the same as transactions without them, and can be ignored. Because they either change the execution order of the code or add a judgment, they do not bring too much extra cost. All in all, these protection schemes do not bring much time cost and they are efficient.

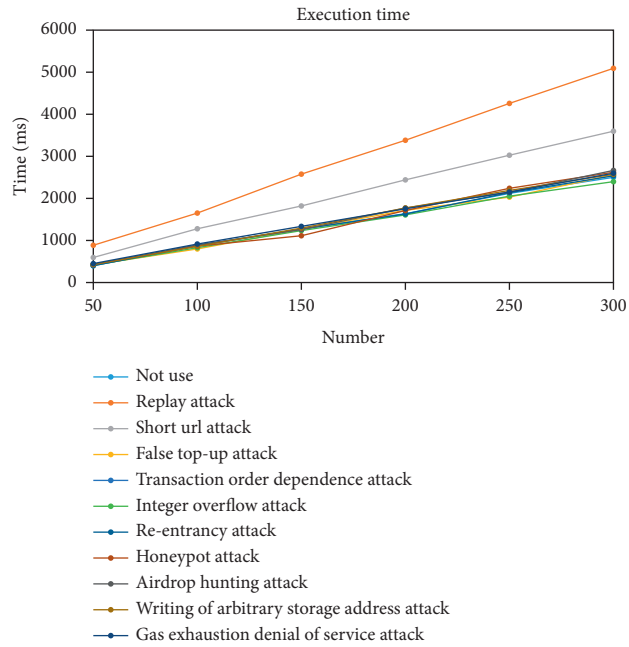

Figure 12: Time cost with different protection schemes.

7. Conclusions

This paper discussed the security threats of the Ethereum blockchain, the attack scenes of these threats, and their protection schemes. Ten security attacks were studied at different levels of Ethereum, which mainly included the application layer, the smart contract layer, and the network layer. The paper presented the corresponding preserved methods in detail according to their attack principles. In general, improving the quality of Ethereum smart contracts can fundamentally prevent attacks. Finally, we evaluate these protection schemes by experiments.

In the future, on the basis of studying public chain security, alliance chain security and cross-chain security will be studied, and security protection schemes for multi-attack scenarios between cross-chains will be realized. The automatic attack detection of cross-chain system is also our important research direction.

Data Availability

No Data Support.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant 2020YFB1005604, Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201915), the National Natural Science Foundation of China under Grant No. 61902021, Beijing Natural Science Foundation under Grant No. 4212008, supported in part by the National Key R&D Program of China under projects 2020YFB1006003, the Guangdong Key R&D Program under project 2020B0101090002, and the Guangxi Natural Science Foundation under grants 2018GXNSFDA281054.

References

[1] Y. Chen, J. Sun, Y. Yang, T. Li, X. Niu, and H. Zhou, "PSSPR: a source location privacy protection scheme based on sector phantom routing in WSNs," *International Journal of Intelligent Systems*, vol. 37, 2021.

[2] B. C. Ghosh, T. Bhartia, S. K. Addya, and S. Chakraborty, "Leveraging public-private blockchain interoperability for closed consortium interfacing," 2021, https://arxiv.org/abs/2104.09801.

[3] S. Nakamoto, *Bitcoin: A Peer-To-Peer Electronic Cash System*, Decentralized Business Review, 2008, https://bitcoin.org/bitcoin.pdf.

[4] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.

[5] N. Szabo, *Formalizing and Securing Relationships on Public Networks*, First monday, 1997.

[6] Y. Li, H. Liu, Z. Yang et al., "Protect your smart contract against unfair payment," in *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*, pp. 61–70, IEEE, Shanghai, China, September 2020.

[7] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 254–269, Vienna, Austria, October 2016.

[8] J. Krupp and C. Rossow, "teether: gnawing at ethereum to automatically exploit smart contracts," *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1317–1333, Baltimore, MD, USA, August 2018.

[9] P. Tsankov, A. Dan, D. Drachsler-Cohen, and A. Gervais, "Securify: practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 67–82, Toronto, Canada, October 2018.

[10] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: analyzing safety of smart contracts," *Ndss*, pp. 1–12, San Diego, CA, USA, February 2018.

[11] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "Easyflow: keep ethereum away from overflow," in *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 23–26, IEEE, Montreal, QC, Canada, May 2019.

[12] S. So, S. Hong, and H. Oh, "SMARTEST: effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution," *30th USENIX Security Symposium (USENIX Security 21)*, August 2021.

[13] L.-Q. Tu, X.-B. Sun, J.-L. Zhang, J. Cai, B. Li, and L.-L. Bo, "Survey of vulnerability detection tools for smart contracts," *Computer Science*, vol. 48, no. 11, pp. 79–88, 2021.

[14] C. Hou, M. Zhou, Y. Ji, P. Daian, G. Fanti, and A. Juels, "SquirRL: automating attack discovery on blockchain incentive mechanisms with deep reinforcement learning," 2019, https://www.arxiv-vanity.com/papers/1912.01798/.

[15] T. Li, Z. Wang, G. Yang, Y. Cui, Y. Chen, and X. Yu, "Semi-selfish mining based on hidden Markov decision process," *International Journal of Intelligent Systems*, vol. 36, 2021.

[16] T. Li, Z. Wang, Y. Chen, C. Li, Y. Jia, and Y. Yang, "Is semi-selfish mining available without being detected?" *International Journal of Intelligent Systems*, vol. 36, no. 10, 2021.

[17] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," *IACR Cryptology ePrint Archive*, vol. 1, no. 236, pp. 1–26, 2018.

[18] D. Li, J. Liu, Z. Tang, Q. Wu, and Z. Guan, "AgentChain: a decentralized cross-chain exchange system," in *Proceedings of the trust security and privacy in computing and communications*, pp. 491–498, Rotorua, New Zealand, August 2019.

[19] A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis, "Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers," *IEEE*, in *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 294–308, Genoa, Italy, September 2020.

[20] A. Li, X. Wei, and Z. He, "Robust proof of Stake: a new consensus protocol for sustainable blockchain systems," *Sustainability*, vol. 12, 2020.

[21] H. Wang, Y. Cen, and X. Li, "Blockchain router: a cross-chain communication protocol," in *Proceedings of the 6th international conference on informatics, environment, energy and applications*, pp. 94–97, Jeju Republic of Korea, March 2017.

[22] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 706–719, Denver, CO, USA, October 2015.

[23] L. Luu, V. Narayanan, C. Zhena, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 17–30, Vienna, Austria, October 2016.

[24] T. D. Nguyen, L. H. Pham, and J. Sun, "SGUARD: towards fixing vulnerable smart contracts automatically," 2021, https://arxiv.org/abs/2101.01917.

[25] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Proceedings of the International conference on principles of security and trust*, pp. 164–186, Springer, 2017.

[26] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: vulnerabilities, attacks, and defenses," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–43, 2020.

[27] Y. Chen, S. Dong, T. Li, and Y. Wang, H. Zhou, Dynamic multi-key FHE in asymmetric key setting from LWE," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 5239–5249, 2021.

[28] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, "A survey on long-range attacks for proof of stake protocols," *IEEE Access*, vol. 7, pp. 28712–28725, 2019.

[29] Y. Huang, J. Tang, Q. Cong, and A. Lim, "Do the rich get richer? Fairness analysis for blockchain incentives," in *Proceedings of the 2021 International Conference on Management of Data*, pp. 790–803, 2021.

[30] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, vol. 107, pp. 841–853, 2020.

[31] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts," *IACR Cryptol.ePrint Arch.*vol. 2016, 2016.

[32] Github, "Safemath," 2021, https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/9a42784ccc26980ca48d79191edc91e8af2185ed/contracts/utils/math/SafeMathUpgradeable.sol.