Cybersecurity and Ethereum Security Vulnerabilities Analysis

Tingyu Ma *

College of Science and Engineering, University of Glasgow, Glasgow G12 8QQ, United Kingdom

* Corresponding Author Email: 2572228M@student.gla.ac.uk

Abstract. As computer technology develops, the popularity of cryptocurrencies and their use will grow, and the newer people enter the industry. It changes the business model between organized businesses out of the need for another trusted party. Blockchain smart contracts can automatically enforce agreed contract between two unknowns. Briefly introduce Ethereum, a cryptocurrency, and focus on the security of its smart contracts in internet transactions. Ethereum was the first platform to support high-level programming languages to implement smart contracts, and the second largest blockchain platform, providing a runtime environment for essentially all Decentralized Finance applications. Bitcoin also supports the development and execution of smart contracts, but it is affected by the nature of the programming language used, and it hardly supports transactions except for verifying signatures. Because smart contracts can support a variety of large transactions, some security vulnerabilities can be extremely costly. In an extensive search and survey, the issue of smart contracts for the Ethereum blockchain was valued. The article will discuss some of the existing or former contract vulnerabilities and their solutions. It concludes with a discussion of the future direction of the smart contract space and provides some suggestions for those researching the field.

Keywords: Decentralization, finance, blockchain, smart contracts, security breaches, Ethereum trading, cybersecurity, development

1. Introduction

Ethereum is a technology used to build applications and organizations, hold assets, transactions, and communications, independent of the control of central authorities. Use Ethereum without handing over all your personal details – to maintain control over your own data and shared content. With the abundance of smart contract application scenarios in areas such as finance, insurance, gaming, and energy. A large number of financial assets have great attraction and inevitably become the target of attack. In addition, the security of the programming language that compiles Ethereum smart contracts and the diversity of smart contract software, contracts without third-party witnesses, these are the risks of smart contracts. Smart contracts were first introduced by Nick Szabo in 1994 to provide a secure approach over traditional contracts and to reduce the other transaction costs associated with contracts. Smart contracts allow trusted transactions to take place without a third party, and these transactions are traceable and irreversible.

One of the most important benefits of smart contracts compared to regular contracts is that when the contract conditions are fulfilled, the results are automatically executed. There is no need to wait for human execution results. In other words: smart contracts eliminate the need for trust. (eth.org)

Ether is an open source blockchain underpinning system, provides a very rich API and interface on which many people can quickly develop various blockchain applications.

Ethernet primarily uses Solidity to write smart contracts and provides a smart contract toolkit on Microsoft Cloud Services that runs on the Ethernet blockchain to ensure transactions are conducted fairly.

Here is a simple example: In the traditional contract, Bob and Amy bet $10, Bob thinks that tomorrow's game Team A will win, and team B wins, Bob thinks that because a player on Team A is injured and does not play without giving Amy $10, A has no way. Another $10 was bet before the second game, Amy thought team C would win and Bob thought team D would win, and the game was canceled.

Traditional contracts are affected by various dimensions, automation dimensions, subjective and objective dimensions, cost dimensions, execution time dimensions, default penalty dimensions, scope of application dimensions, etc.

The smart contract is to consider and stipulate all possible situations before betting, and the procedure will be automatically strictly implemented according to the pre-set contract content after the event occurs. Thus, the problems that arise with traditional contracts can be solved.

2. Method

Smart Contracts Special Mechanisms vs Traditional procedures

2.1. Gas mechanism

Gas refers to a certain fee required to perform operations on the Ethereum network and to trade. Contracts need to be enforced and proven by miners, so when packaging a transaction, i.e., adding it to a block, in order to avoid node resources being wasted due to the large number of loops and other operations included in the transaction, and the contract should pay a few gas to the miner first. The execution of a contract in Ethernet itself also requires a certain amount of gas to be consumed according to the rules set internally. The out-of-gas exception will be triggered and all execution status of the current contract will be rolled back.

2.2. Delegatecall mechanism

Contracts can invoke other contracts through the mechanism of message invocation, and delegatecall is a special type of message call. It differs from ordinary call instructions in that the behavior of ordinary call instructions is to jump to the called contract and execute the corresponding code in that contract before returning to execute the subsequent code, which has no effect on the context of the initiator of the call. The delegate call is equivalent to copying a piece of code in the contract, which is called, to the near of the call initiator contract, and modifying the information in the call initiator. This operator allows instructions to be sent to an external contract, executing a portion of the instructions in the context of the caller. This operator is convenient when using a shared code base.

2.3. Large-scale Ethereum trading incidents

The DAO Attack, according to David Siegel's investigation [2] and review of the entire incident published in Jun 19, 2016, DAO is a decentralized autonomous organization conceived and programmed using the team behind the German Ethereum startup Slock.it. It is designed to resemble a venture capital project fund, where people add cash to the DAO by buying tokens that represent ownership, providing it with the resources it needs. People can make spending proposals to the DAO, and members who have already purchased them can vote to approve those proposals, which is like a contribution that gives people the right to vote rather than ownership. As a result, it was attacked by hackers using the recursive call to Ethereum sending vulnerability.

When the smart contract is executed, call the fallback function if the specified function is not found, or if no function is specified at all. The function call.value() has vulnerable, a hacker can construct a loop when call.value() sends ether without specifying which function, the fallback function will be called and appended with all the currently remaining gas until the transfer has finished transferring all the coins.

In the research paper Finding the Greedy, Prodigal, and Suicidal Contractsat Scale by Ilya Sergei, an Associate Professor at UCL, they implemented Maian tool, a tool for specifying and reasoning about tracking properties, and an analysis of nearly a million smart contracts showed that more than 30,000 contracts per contract were vulnerable to attack in 10 seconds. They also sampled 3,759 smart contracts, 3,686 of which had a true positive rate of 89% of reproducing real exploits. Many

exchanges do not have filters in place and hackers can simply construct POCs to take all the tokens from the exchange's virtual accounts.

2.4. Common security vulnerabilities

Through certain research and consultation, the security vulnerabilities of smart contracts are roughly summarized into three categories, as shown in Table 1.

Table 1. Security Vulnerabilities

Solidity Programming Language	Integer type error Randomness Using 'Block Hash' Unchecked return value Permission control problem Denial of Service Call to the Unknown Gasless 'Send' Insufficient Gas Griefing Hash Collision with Arguments
Ethereum Virtual Machine Features	Re-Entrancy Short Address Attack Code injection Immutable Bugs or Mistakes
Design features of Ethereum Blockchain	Trading order dependency Timestamp dependency Lack of Transactional Privacy Predictable random processing

The following is the analysis of the security vulnerabilities

2.5. Solidity Programming Language (SPL)

1. Integer type error

Integer type errors consist mainly of truncation errors, sign errors and arithmetic errors. Arithmetic errors include integer overflow, divide by zero and modulo zero. EVM uses several fixed lengths to represent integers, which means that it can only represent numbers within a certain range, and once the result of the single integer operation is outside this range then the integer overflow can occur. An attacker can use an integer overflow vulnerability to skip certain conditional judgements or tamper with data, as was famously the case with the BEC vulnerability where an integer overflow occurred during a transfer resulting in the evaporation of a large number of tokens. Overflow vulnerabilities can be effectively avoided by using the SafeMath maths calculation library which provides security checks. In older versions of EVM and Solidity, dividing by zero and modulating zero only results in an operation with a zero result and no exception. A truncation error is a loss of precision caused by converting integer type data to integer type data with a shorter width. A symbolic error is a result in a negative number becoming a very large integer, and signed integer data of the same size is converted to unsigned integer data of the same size [8]. To tackle integer issues, it is recommended to add require() functions, assert() functions, or 'SafeMath.sol' [4] lib etc. databases to perform tests on arithmetic operations. It is also possible to spend time comparing symbols and arguments to implement arithmetic operations before the function is run.

2. Randomness Using 'Block Hash'

Sometimes randomness needs to in an operation, in this purpose, it is good to use the block hashing [13]. However, it can be manipulated like a timestamp, miners can predict as the same thing [7].

Other transactions within the same block can easily read the hash value of the block. If it is a miner who takes the attack, then it may cause a worse situation. For example, Oracle, RANDAO, RNG [5], and other random external resources can be used [7].

3. Unchecked return value

Function calls in the Solidity language of the Ethereum network also set the return value, but the failure of low-level function calls (such as calls, sends, delegatecall, etc.) simply shows in the return value whether an exception has occurred. An attacker could cause the program to execute differently than expected by deliberately sending a failed operation, thus causing confusion about the state of the smart contract. Developers can ensure that a contract executes with the predefined logic by checking the return value of low-level function calls to determine whether the call was successful when developing the contract [8].

4. Permission control problem

Functions and state variables can be made visible in Solidity using four descriptors: external, internal, public and private. undeclared visibility defaults to public, meaning that the function is not only allowed to be called internally but also as an external interface to the contract. If a function that involve sensitive operations such as transfers do not declare visibility, this could allow an attacker to take advantage of the situation [8].

5. Denial of Service

Denial-of-Service (DoS) means to an operation that cannot be recovered from or a controlled consumption of unlimited resources. A DoS attack against an Ether contract can lead to a massive drain on Ether and Gas, or in more severe cases, a permanent loss of use of the contract. major problems with the game's operations [8].

6. Call to the Unknown

When the fallback functions were executed under some certain conditions, and the signature in the function does not true. The function that led to the use of the Calle contract, through another smart contract, will be implemented in the transfer of Ethereum [7].

To improve the security of calling unknown functions and contracts, the last task of the local code is to make a call to the unknown function. The Effect Interaction Pattern should be used to prevent state changes after an unknown contract is called [7].

7. Gasless ‘Send’

Each Ether smart contract requires a gas fee to be paid when the Ether is transferred into the contract when it takes effect [15]. The out-of-gas exception will be thrown when the maximum gas for the fallback function in a Calle contract is 2300 [16]. A malicious user could use this vulnerability to profit by keeping the untransferred Ether due to the exception. the falllback function for exception handling due to a gas consumption fault should make execution less expensive, mitigating contract failures and costs [7].

8. Insufficient Gas Griefing

External data uses the same contract in subcall, vulnerable to gas griefing attacks when the subcall fails, the whole process can be fully recovered [13]. So, hackers can keep the subcall from failing to censor transactions [7, 14]. To prevent this, create a function should review whether it provide enough units of gas or not. Also, allow a trusted outsider to review and reward successful transactions [7] is helpful.

9. Hash Collision with Arguments

'abi.encodePacked()' is a non-standard packaging pattern that is mainly used to index packaging parameters. When there are many variable-size parameters, 'a bi.encodePacked()' at runtime in some cases can cause a hash conflict. This packaging pattern packs parameters sequentially, regardless of the order of the parameters, and hackers are able to change the position of the parameters to verify the signature [7]. It is recommended to use 'abi,encode()' to forward parameters, and when passing parameters, only fixed-length arrays or single parameters are allowed [7].

2.6. Ethereum Virtual Machine (EVM) Features

1. Re-Entrancy

When a user calls a smart contract, calling a fallback if the called contract does not find the function it should be called or if the contract only accepts Ethereum and no other information. An attacker can construct a special fallback function to attack a smart contract that is vulnerable to reentry, such as a fallback function that contains code that recalls a function in the attacked contract that previously transferred money to the attacker, thus enabling recursive calls and exhaust the assets of the user who hold the contract. The DAO event that famously led to the hard fork of Ether was linked to a re-entry vulnerability [8].

2. Short Address Attack

An unchecked user input is the Short Address Attack. An attacker uses the VM's auto-completion mechanism to construct an address with a zero at the end for a contract call, and intentionally omits the zero in the address, which is at the end of it, then passing in parameters. This means the amount left will be doubled, resulting in more spends being transferred than originally set. Detect user addresses and verify identities, reject incorrect format input, short address attacks can be avoided [8].

3. Code injection

Ether's delegate call mechanism, which uses a delegatecall directive that allows contracts to run some code from other contracts in their own context. An attacker could use this vulnerability to inject code into a contract that modifies malicious operations such as important state variables in the contract [8].

4. Immutable Bugs or Mistakes

This is the nature of the Ethereum blockchain [13], where once deploy a contract, it will not be changed anymore, and errors in the contract remain unchangeable and create some problems [7]. In order to make the contract changeable at any time, Marino et al. [6] suggested addressing this issue from the perspective of a legal contract by redefining the contract [7].

2.7. Design features of Ethereum Blockchain (EBD)

1. Trading order dependency

The order in which two transactions of the same smart contract are executed can lead to this vulnerability. Hackers can use this vulnerability to attack users who perform changes on transactions. The order of transactions is determined by the miners, and if the miners are hackers, there are serious consequences [7].

To prevent such an attack, Ethereum-based functions can be used to dictate the order in which transactions are executed [12] or the contract terms suggested by Luu [3] et al.: 'The contract code either returns the expected output or fails'.

2. Timestamp dependency

When the trigger for executing a transaction is a block timestamp [13], it can lead to a vulnerable situation as bad miners can exploit the block timestamp to treat others [7]. Some smart contracts can obtain the timestamp of the current block and use it as a basis for judgement via the parameter block.timestamp, while the timestamp can be determined by the miner within certain limits. Therefore, it is generally considered safe to use timestamps in a smart contract if the timestamp-dependent code allows for a 12-minute error [8].

Since block numbers cannot be changed, Luu et al. [9] believe that using block numbers would make it safer for hackers to make use of timestamps, and that variables in smart contracts are preferably not timestamps

3. Lack of Transactional Privacy

Due to the nature of DeFi, the user's transaction information is public [1]. Many people do not want to be monitored, which will restrict user transactions and threaten the privacy and security of users [7].

It is a good choice to encrypt the contract before it is deployed [10], or to use a tool [11] to encrypt the contract. All blockchains of the same type can use this privacy-preserving contract. The smart contract framework Hawk can store no financial transactions, which is very good protection for users.

4. Predictable random processing

Some smart contracts require external data sources [3], but if the external data is incorrect, the contract will fail and the correctness of the external data cannot be guaranteed [7]. An authenticator is required to review the external data and use encrypted data.

3. Discussion

Smart contracts need to execute as expected and errors can lead to significant losses. Security analysis tools are necessary and important to analyze and fix the security vulnerabilities of smart contract because people cannot change smart contracts after the blockchain has been deployed due to its immutable nature. There is a survey about the detection tools for the analysis of ETH smart contracts with different parameters is summarized in Satpal Singh Kushwaha [7] et al.

Most of the discussion relates to Ethereum smart contract vulnerabilities, but have not touch some security vulnerabilities in blockchain platforms except Ethereum. In future developments, more features will take place to Ethernet blockchain smart contracts, which will produce more vulnerabilities. Existing detection tools may not be able to prevent emerging vulnerabilities, and new technology product will be enhanced and developed to conduct further research to provide smart contracts with self-protection mechanisms to stop all actions in the event of an attack. As time goes on, Ether will get bigger and bigger, and it will bring a whole new set of security issues.

4. Conclusion

This paper has introduced the functional uses of smart contracts. Common and partial security vulnerabilities and solutions have been analyzed. Some suggestions for future research directions in smart contract cybersecurity are provided.

Each contract security vulnerability is defined differently in different research efforts, making it difficult to create a unified indicator to analyze and compare vulnerabilities, and more research is needed to strengthen this work. In order to further strengthen the security of smart contracts in the future, the new smart contract security analysis technology needs more research. The security of smart contracts on blockchain platforms outside of Ethereum is also a major direction for future research, such as EOS, NEO, Corda, Solana etc...

References

[1] M. Alharby and A. V. Moorsel "Blockchain based smart contracts: A systematic mapping study" Proc. Comput. Sci. Inf. Technol. pp. 125-140 Aug. 2017.

[2] David Siegel, Jun 19, 2016. Understanding The DAO Hack for Journalists, https://pullnews.medium.com/understanding-the-dao-hack-for-journalists-2312dd43e993#.kw0ufw25q

[3] L. Luu, D.-H. Chu, H. Olickel, P. Saxena and A. Hobor, "Making smart contracts smarter", Proc. ACM SIGSAC Conf. Comput. Commun. Secur., pp. 254-269, Oct. 2016.

[4] A. Singh, R. M. Parizi, Q. Zhang, K. K. R. Choo and A. Dehghantanha, "Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities", *Comput. Secur.*, vol. 88, Oct. 2020.

[5] C. F. Torres, J. Schátte and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts", *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, pp. 664-676, Dec. 2018.

[6] Y. Fu, M. Ren, et al., "EVMFuzzer: Detect EVM vulnerabilities via fuzz testing", *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 1110-1114, Aug. 2019.

[7] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur and H. -N. Lee, "Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract," in IEEE Access, vol. 10, pp. 6605-6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[8] A. Dika and M. Nowostawski, "Security Vulnerabilities in Ethereum Smart Contracts," Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 955-962, doi: 10.1109/Cybermatics_2018.2018.00182.

[9] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms applications and design patterns" in Financial Cryptography and Data Security, Cham, Switzerland:Springer, vol. 10323, pp. 494-509, 2017.

[10] N. Szabo, "Formalizing and securing relationships on public networks", 1st Monday, vol. 2, no. 9, pp. 1-15, Oct. 1997.

[11] A. Kosba, A. Miller, E. Shi, Z. Wen and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts", Proc. IEEE Symp. Secur. Privacy (SP), pp. 839-858, May 2016.

[12] C. Natoli and V. Gramoli, "The blockchain anomaly", Proc. IEEE 15th Int. Symp. Netw. Comput. Appl. (NCA), pp. 310-317, Oct. 2016.

[13] M. Wöhrer and U. Zdun "Design patterns for smart contracts in the ethereum ecosystem" Proc. IEEE Int. Conf. Internet Things pp. 1513-1520 Aug. 2018.

[14] S. Jumnongsaksub and K. Sripanidkulchai "Reducing smart contract runtime errors on ethereum" IEEE Softw. vol. 37 no. 5 pp. 55-59 Oct. 2020.

[15] J. Krupp and C. Rossow "TEETHER: Gnawing at ethereum to automatically exploit smart contracts" Proc. 27th USENIX Secur. Symp. pp. 1317-1333 2018.

[16] I. Ashraf X. Ma B. Jiang and W. K. Chan "GasFuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities" IEEE Access vol. 8 pp. 99552-99564 2020.