Finding Vulnerabilities in Smart Contracts

Michele Massetti
big.michelemassetti@gmail.com
Karlsruhe Institute of Technologies
Karlsruhe, Baden-Württemberg, Germany

ABSTRACT

Blockchain is a revolutionary technology that enables users to communicate in a trust-less manner. The most prominent change brought by this technology is the mode of business between organizations: they do not need anymore a trusted third party. It is a distributed ledger technology based on a decentralized peer-to-peer (P2P) network. Since Bitcoin was deployed, many blockchain systems have been born with more capabilities, which have allowed them to fit many different use cases. Smart Contracts, which are programs running on blockchain systems, could extend the potentiality of blockchain from a platform for financial transactions to an all-purpose utility. The development of innovative and prominent applications is a consequence of them, such as NFT marketplaces, music royalty tracking, supply chain and logistics monitoring, voting mechanism, cross-border payments, and many others. Finding bugs and vulnerabilities in them is necessary for assuring their correct behaviour. This paper deals with the way for finding the vulnerabilities in Ethereum blockchain-based smart contracts. We review related works regarding the classification of the most common vulnerabilities and tools which support their detection.

KEYWORDS

Solidity, Software, Vulnerability, Blockchain

1 INTRODUCTION

Nowadays, the major platform for decentralized decentralized finance (DeFi) and applications (dApps) is Ethereum. It can be described as the "internet of Blockchain". Its ecosystem consists of the underlying blockchain, a large variety of smart contracts deployed on it, a wide range of valuable assets.

This growing technology has attaracted many investors, indeed, according CoinGeko, the crypto market's value is standing around $2 trillion. On the other hand, interest in such a market has grown even among malicious attackers. Attacks such as the “Parity Wallet Hack” and the “Decentralized Autonomous Organization Attack” cost millions of dollars simply because of naive bugs in the smart contract code. Blockchain and smart contract technologies have multiple aims, but unfortunately, new applications based on them still contain bugs and multiple vulnerabilities, which cause several issues for the end-users. Most of the use of this technology relates to finance or certifications, therefore integrity, authentication and authorisation in transactions are mandatory.

Colloquium on Vulnerability Discovery 2022, August 1st, 2022, KIT, Karlsruhe, Germany

The research field behind blockchain technology is growing, as well as the one concerning its security and accordingly, many analysis tools were developed. These incorporate various strategies for performing the analyses, concerning the technical aspects of smart contracts, so these would work differently according to the object of the analysis.

Among the many aspects of smart contract, our systematic literature review focuses on studies related to vulnerabilities and analysis tools for their detection. We will try to give an answer to the following research questions:

- Which are the main vulnerabilities in Smart Contracts?
- Which methodologies are implemented by analysis tools?
- How should we behave for the detection of Vulnerabilities?

In Section 2, we compare the actual papers and works regarding this topic. Section 3 explains the objective of our analysis: Smart Contracts Vulnerabilities. We give a taxonomy for the main vulnerabilities regarding Solidity. The classification of analysis tools is shown in Section 5. We discuss about the main strategies implemented by those. Options

- In the last Section 7 we try to define a guideline for detecting vulnerabilities.
- In the last Section 7 we give an overview, defining the suitable cases which the tools work better.
- In the last section 7 we propose a real case.

2 RELETED WORKS

- Papers about vulenrabilities detection for defing a taxonomy
- papers regarding comperison between tools
- papers of the tools that we want to have a Look

Kushwaha et al. [2]

3 VULNERABILITIES IN SOLIDITY

An overview of classification of Vulnerabilities in Solidity is given in this section. Classifications by scholars and community taxonomies are the strating point for drowing our own classification. These knoledges are used for having our own classification; we depicted which ones are the most relevant vulnerabilities for our work. The term vulnerability is used in a similar sense than is common in computer security. It refers to a weakness or limitation of a smart contract that may result in security problems. A vulnerability represents even the way the attacker can exploit the contract. This includes locked or stolen resources, breaches of confidentiality or data integrity, and state changes in the environment of smart contracts that were not detected by developers or users, therefor they possibly be exploited by a malicious part.

3.1 Solidity Vulnerabilities and real-world examples

3.2 Vulnerabilities Classification and examples

We summeries 10 different Vulnerabilities, providing a definition and a real-world example.

Reentrancy. It is one of the vulnerabilities much more exploited during the recent year. Many attacks occured and could steal great amount of many to vulnerable protocol. A reentrancy attack can occur when a function makes an external call to another untrusted contract before it resolves any effects. A recursive call back to the original function is made by the attacker, before the function could updated the state of the state of the variables. The goal is repeating interactions that would have otherwise not run after the effects were resolved. The function withdaw, line 8 to 16 Listing 1, implements this bug, because the balance of the user is updated after the call. The attacker can run multiple the function withdaw without updating the balance, withrawing all the funds.

```
1  contract EtherStore {
2      mapping(address => uint) public balances;
3      ...
4      //vulnerable function exploited by the attacker
5      function withdraw() public {
6          uint bal = balances[msg.sender];
7          require(bal > 0);
8
9          (bool sent, ) = msg.sender.call{value: bal}("");
10         require(sent, "Failed to send Ether");
11
12         balances[msg.sender] = 0;
13     }
14     ...
15 }
```

Listing 1: Reentrancy

Unexpected Ether balance. An exploites can generated a Dos attack when the smart contract strictly assume a specific token balance. It is always possible to forcibly send ether to a contract (without triggering its fallback function), using selfdestruct, or by mining to the account. A common practice for detecting this bug is checking the invariant. Smart contracts which manage ERC20token, currenciens built on the blockchain, keep track of the total supplay. This is an example of invariant.

Delegatecall to Untrusted Callee. The definition of delegatecall is given by [1]. Delegatecall is a special variant of a message call. It is identical to a message call apart from the fact that the code at the target address is executed in the context of the calling contract and msg.sender and msg.value do not change their values. As a result of the context-preserving nature of DELEGATECALL, building vulnerability-free custom libraries isn't as easy as one might think. The code in libraries themselves can be secure and vulnerability-free; however, when run within the context of another application new vulnerabilities can arise. An example is the use of a function call to send Ether, which could cause the execution of the fallback function of a malicious attacker. The consequence can be a reentrancy attack.

Table 1: Our Classification of Vulnerabilities

Vulnerability	DASP	SWC
Reentrancy	1	107

Overflows & Underflows. The Ethereum Virtual Machine (EVM) defineds fixed-size data types for integers. An example is the identifier uint8, unsign integer over 8 bit, for variables means the variable can only store numbers in the range [0,255]. The attacker can craft the input value of a vulnerable contract's function in order to force a specific operation. In the contract TimeLock in Listing 2, an attacker could forge a parameter of the function function increaseLockTime, in oder to set lockTime to 0 because of Overflows. So it can call withraw and passing the require at line 17.

```
1  contract TimeLock {
2
3      mapping(address => uint) public balances;
4      mapping(address => uint) public lockTime;
5
6      function deposit() public payable {
7          balances[msg.sender] += msg.value;
8          lockTime[msg.sender] = now + 1 weeks;
9      }
10
11     function increaseLockTime(uint _secondsToIncrease)
            public {
12         lockTime[msg.sender] += _secondsToIncrease;
13     }
14
15     function withdraw() public {
16         require(balances[msg.sender] > 0);
17         require(now > lockTime[msg.sender]);
18         msg.sender.transfer(balances[msg.sender]);
19         balances[msg.sender] = 0;
20     }
21 }
```

Listing 2: Overflows & Underflows

Authentication & Access Control Vulnerabilities.

Transaction Order Dependence.

Block Timestamp Manipulation.

Unchecked CALL Return Values.

Race Conditions / Front Running.

Denial Of Service (DOS). rthyj
The Table 1 gives a graphical overview of our classification, referring with the SWC Registry and DASP.

4 METHODOLOGIES IMPLEMENTED BY SECURITY ANALYSIS TOOLS

Static analysis. refers to a class of methods that examine the source code or bytecode of a contract without execut- ing it. Most methods listed below are static.

Dynamic analysis. means to observe a contract while executing (parts of) it in the original context.

Disassembling. means to translate EVM bytecode into bet- ter readable assembly language, where machine operations and storage addresses are represented symbolically.

Decompilation. is the process of transforming EVM byte- code to a more compact representation on a higher abstrac- tion level (like intermediate or Solidity code) to enhance the readability of the code or to ease data flow analysis.

Control flow graph (CFG). is a directed graph, where the basic blocks of a program serve as the nodes. An arc connects node A with node B if it is possible that block B gets executed immediately after block A. The arc may be labeled by the condition under which this path is chosen.

Formal verification. means verification by formal methods with the aim of proving or disproving system properties rigorously. As a prerequisite, all components referenced by such a property as well as their behavior must have been specified formally. E.g., to verify properties of smart contracts on bytecode level formally, we need a formal specification of the EVM and of the properties.

Dynamic CFG. is similar to a CFG with the difference that arcs indicate the actual control flow encountered during a particular execution of the code.

Model checking. is a technique for automatically verifying cor- rectness properties of finite-state systems. It requires a model of the system which is then checked against a given specification.

Call graph. is a directed graph, where the nodes are functions. There is an arc from node A to node B if function A calls function B.

Abstract Syntax Tree (AST). represents the syntactic struc- ture of Solidity code as a tree. It occurs as an intermediate product when compiling Solidity to bytecode. Often, it is better suited for analyzing Solidity code.

Symbolic execution. means to execute code using symbols instead of concrete values for the variables. Operations on these symbols lead to algebraic terms, and conditional statements give rise to propositional formulas that char- acterize the branches. A particular part of the code is reachable if the conjunction of formulas on the path to this part is satisfiable, which can be checked by SMT-solvers.

5 SECURITY ANALYSIS TOOLS

6 REAL-WORLD CASE

7 CONCLUSION

ACKNOWLEDGEMENTS

REFERENCES

[1] 2017. Solidity Doc. https://docs.soliditylang.org. https://docs.soliditylang.org/en/v0.4.21/introduction-to-smart-contracts.html

[2] Satpal Singh Kushwaha, Sandeep Joshi, Dilbag Singh, Manjit Kaur, and Heung-No Lee. 2022. Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. *IEEE Access* 10 (2022), 6605–6621. https://doi.org/10.1109/ACCESS.2021.3140091

A RESEARCH METHODS

A.1 Part One