Making Smart Contracts Smarter

Deniz Aytemiz

April 17th 2022

Since the Bitcoin white paper was introduced, distributed database systems called blockchain and cryptocurrencies leveraging this technology are getting more mainstream, especially in recent years. As the use of cryptocurrencies as financial assets increases, the security vulnerabilities associated with each cryptocurrency is crucial for people who are part of that cryptocurrency's network. In this paper, multiple exploitable vulnerabilities in smart contracts in Ethereum network are introduced and examined. The authors of the paper exemplify how these vulnerabilities can be exploited in addition to introducing ideas on how to refine Ethereum's protocol to prevent these attacks.

One of the critical points on smart contracts is that they are immutable and irreversible. Hence unlike many distributed applications it cannot be patched and one must ensure its correctness and safety before its deployment. Authors introduce a symbolic execution tool 'OYENTE' for users to detect bugs in pre-deployment. The quantitative data in the paper is given by analyzing real-world Ethereum smart contracts with this tool which underlines the severity and frequency of the certain vulnerabilities.

First security bug introduced in the paper is is due to Transaction-Ordering Dependence(TOD). In each epoch, blockchain state is updated several times depending on the new blocks contents. There is a possibility that a single block would contain two transcations (almost occuring around same times) invoking the same contract. In that case, the order of which transaction would be executed is crucial since depending on the order two different blockchain states can be achieved. The order which they will be executed will be up to the miner. A malicious party can utilize this fact by listening transaction broadcasts, observing a transaction happening(Tu) and initiating a new transaction(To) in the time period of 12 seconds between broadcast of transaction and creation of the block for Tu hoping they would end up in the same block. The attacker can deploy transaction 'To' such that it contradicts or alters the functionality of 'Tu'. Hence the in the case of 'To' executed before 'Tu', attacker can successfully exploit the contract.

Second vulnerability introduced is due to Timestamp Dependence. Time dependence contracts uses block timestamp to invoke a critical operation.The miners can manipulate block timestamp by approximately 900 seconds. This is risky because a malicious miner can set the block timestamp in a way such that it would invoke certain operations that would serve the miner. For example, in a smart contract block time stamp is utilized as random seed, a malicious miner can alter the timestamp(hence random seed) for increasing the probability of getting more desired outcomes for his benefit.

Third vulnerability introduced is mishandled exceptions. It occurs due to inconsistency in exception policy. In Ethereum there are multiple ways one contract calls another, however depending on how the call is made, if there occurs an exception the callee might or might not get informed. Not in all calls the execution is checked whether it was proper or not. As this may create errors in natural flow of things, it also provides a basis for 'exceed call-stack depth limit' attack in which a malicious party allocates all space in call-stack by invoking calls resulting in failure of the execution of the rest of the contract.

The authors suggest that, in order to handle TOD related vulnerabilities, a guard condition should be added such that if current state of blockchain does not satisfy the condition the upcoming transactions will not be executed. In the case of timestamp related vulnerabilities they suggest avoiding usage of timestamp as random seed and use block index as a time measure instead. For exception handling, authors propose either automatic exception propagation at every level or including throw/catch mechanisms for reversing callers state and process termination.

This paper has many crucial arguments, ideas, methods and quantitative/qualitative analysis. Its step by step explanation of vulnerabilities and countermeasures also by exemplifying with possible attack or accidental scenarios makes it much easier for reader to comprehend the main idea. It also presents the inner structure of Ethereum network and 'OYENTE' tool in a very structured and comprehensive way. Apart from 'Validator' part of 'OYENTE' not being complete and hence no fully execution of the platform being realized yet, I cannot see any room to improve.

I have learned how smart contracts reside and function in Ethereum network and how gas fee system works. In addition to gaining an understanding on what can create a vulnerability in coding a smart contract. I also gained an intuitive sense on the severity and frequency of the possible security violations smart contracts introduce and the necessity of improvement before it can become a trusted technology for financial asset trading.