Current Trends in Computer Sciences & Applications

DOI: 10.32474/CTCSA.2020.02.000126

ISSN: 2643-6744

Review Article

Vulnerability and Cost Analysis of Heterogeneous Smart Contract Programs in Blockchain Systems

Wan Yeon Lee* and Yun-Seok Choi

Department of Computer Science, Dongduk Women's University, South Korea

***Corresponding author:** Wan Yeon Lee, Dongduk Women's University, Seoul 02748, South Korea

Received: February 21, 2020 **Published:** March 06, 2020

Abstract

Blockchain and smart contract technologies were recently introduced. Due to their short histories, many issues have emerged but are not solved yet. In this article, we summarize the currently concentrated issues under developing with regards to smart contracts: vulnerability and cost optimization. We introduce other issues never considered but essential for successful commercialization of the smart contract technology. Also, we verify that the optimized code of smart contracts can significantly save transactions costs. Finally, we address that the cost-optimized design is affected by the platform type of blockchain systems. So, the vulnerability and cost analysis of smart contracts must consider the blockchain platform to be deployed.

Keywords: Blockchain; Smart Contract; Vulnerability; Cost Optimization; Heterogeneous Platform

Introduction

The first generation blockchain designed for Bitcoin cryptocurrency maintains a record of transactions across several computers linked in a peer-to-peer network [1]. The second generation blockchain maintains a record of digital contracts as well as cryptocurrency transactions in a peer-to-peer network [2]. These digital contracts are made by programming codes and referred to as *smart contracts*. Because of great commercial potentiality of smart contracts, many blockchain systems with their own designs for smart contract have been launched: Ethereum, EOS, Hyperledger Fabric, Tron, Qtum, Cosmos, Cardano, Klaytn, ICON, etc. However, based on the reentrancy vulnerability of smart contracts, the DAO attack steal about 3.6 million Ethers (valued about US$55,000,000 at that time) in June 2016 [3]. Also, similar attacks such as Parity MultiSig Wallet and SmartMesh occurred in 2017. Therefore, the vulnerability analysis study of smart contracts is beginning recently. Although the existing studies focused on the smart contracts of the Ethereum blockchain platform, we examine the vulnerability analysis for other blockchain platforms such as EOS, NEO, and ICON. Another main issue of smart contracts is the transaction costs of smart contracts. The blockchain system consumes computing resources for transactions of smart contracts. So, the blockchain system requires some costs of cryptocurrency for transactions of smart contracts. If a smart contract requests expensive costs for its transactions due to its bad design, its cumulative cost loss becomes more severe as the number of called transactions grows. Only a few recent study [4] dealt with the cost-optimized design of smart contracts. But it considered only the Ethereum blockchain platform. In this article, we examine the cost-optimized design of other blockchain platforms. The rest of this article is organized as follows; In Section 2, we introduce and discuss the vulnerability issues of smart contracts on various blockchain platforms. In Section 3, we introduce and discuss the cost optimization issues of smart contracts on various blockchain platform. In Section 4, we summarize the current issues and address other issues to be considered for smart contracts.

Vulnerability Analysis of Heterogeneous Smart Contracts

The vulnerability of smart contracts in Ethereum blockchain platform was recently issued due to critical money loss of attacks. In order to prevent the vulnerability of smart contracts, many developers try to find vulnerable coding patterns of smart contracts. But they focus on smart contract coding written with the solidity programming language [3, 5]. We introduce vulnerable coding patterns of smart contracts written with other programming languages such as python, C++, and C# [6-8]. The following are vulnerability issues of smart contract coding written with various programming languages.

Vulnerabilities of smart contracts written with solidity

Reentrancy, front-running, timestamp dependence, integer overflow and underflow, DoS with unexpected revert, DoS with block gas limit, Insufficient gas griefing, forcibly sending ether to a contact, deprecated/historical attacks, function default visibility, outdated compiler version, floating pragma, unchecked call return value, improper access control, uninitialized storage pointer, assert violation, use of deprecated solidity functions, delegated call to untrusted callee, authorization through tx.origin, signature malleability, incorrect constructor name, shadowing state variables, weak sources of randomness from chain attributes, and write to arbitrary storage location.

Vulnerabilities of smart contracts written with python

Timeout, unfinishing loop, package import, system call, outbound network call, internal API, randomness, fixed SCORE information, IRC2 token standard compliance, IRC2 token parameter name, event log on token transfer, event log without token transfer, ICX transfer event log, super class, keyword arguments, big number operation, instance variable, and state DB operation.

Vulnerabilities of smart contracts written with C++

Numerical overflow, authorization check, apply check, transfer error prompt, random number practice, and rollback attack.

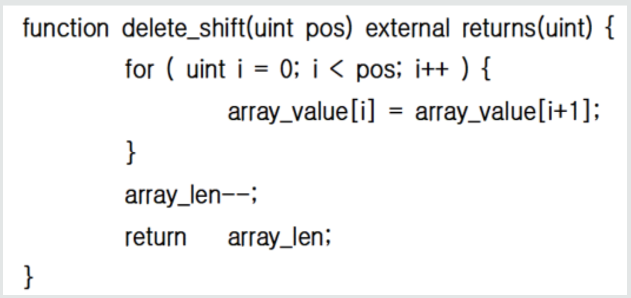
Vulnerabilities of smart contracts written with C#

NEP-5 tokens, DoS vulnerability, and storage injection.

Smart contracts written with solidity programming language are applied to the Ethereum, Klaytn, Tron and Qtum blockchain platforms. Smart contracts written with python language are applied to the ICON, EOS, and NEO blockchain platforms. Smart contracts written with C++ language are applied to the EOS blockchain platform, and those with C# language are applied to the NEO blockchain platform. Note that the above smart contract vulnerabilities are found until now. Other unknown vulnerabilities of smart contracts possibly exist, and many white hackers try to find other smart contract vulnerabilities. Also, there are other smart contracts written with different programming languages such as Ethermint and Plutus. To remove the vulnerabilities of smart contracts, commercial services, call *audit* service, have emerged. These services check whether a given smart contract includes some vulnerable code. So, the provider of smart contracts needs to go through the audit service before deploying the smart contracts to the commercial blockchain system.

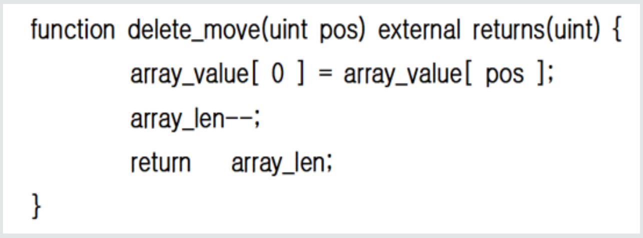
Cost Analysis of Heterogeneous Smart Contracts

Blockchain systems are managed in a distributed manner by multiple separate computers. The benefit of a computer to join in the blockchain system is to get some cryptocurrency. As the computing resources handling a transaction is larger, the charged cryptocurrency becomes severer. So, the minimization of computing resources leads to cost optimization of transactions in the blockchain system. But the computing resources depends on the platform of blockchain systems. For example, in the the Ethereum, Tron, Klaytn, Qtum, and NEO platforms, the number of storing variables dominates the costs of a transaction. In contrast, in the EOS platform, the foot print size and the execution speed dominate the costs of a transaction. Also, the cost of the same transaction may vary depending on the type of blockchain platform. Figure 1 shows a transaction code example in a smart contract written with the solidity language, where the first element in an array with 100 elements is deleted (the value of "pos" is set to 99). In this programming code, the rest ninety-nine elements are shifted in their front positions. On the other hand, Figure 2 shows its variant transaction code, where the first element is deleted by moving the 100-th element into the first element. While ninety-nine storing operations are performed in Figure 1, only one storing operation is performed in Figure 2.



```
function delete_shift(uint pos) external returns(uint) {
        for ( uint i = 0; i < pos; i++ ) {
                array_value[i] = array_value[i+1];
        }
        array_len--;
        return    array_len;
}
```

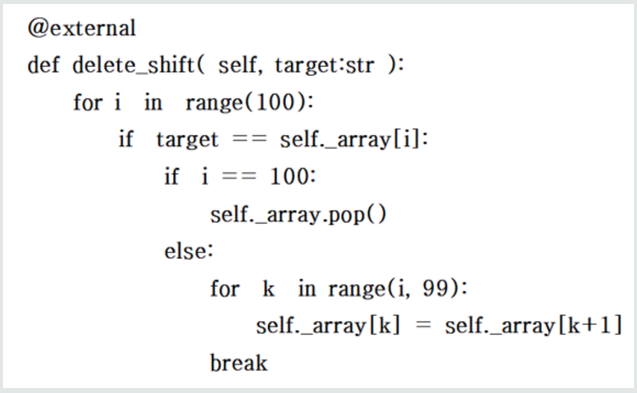
Figure 1: First element deletion with 99 shift operations in a smart contract written with solidity.



```
function delete_move(uint pos) external returns(uint) {
        array_value[ 0 ] = array_value[ pos ];
        array_len--;
        return    array_len;
}
```

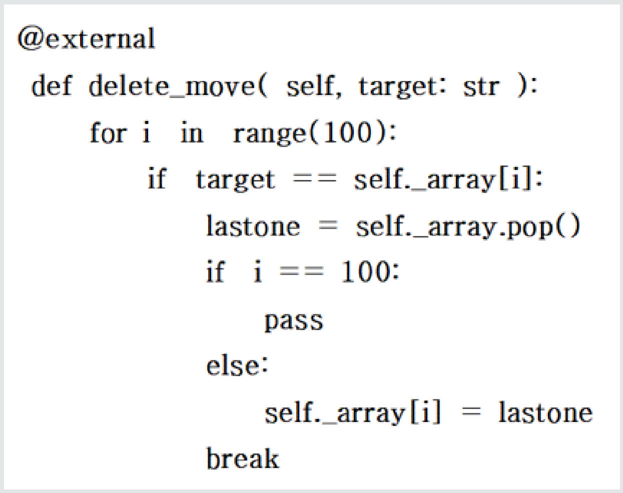
Figure 2: First element deletion with one move operations in a smart contract written with solidity.

In the Ethereum blockchain platform, the cost of "delete_shift (99)" function is 0.00113400 Ether and the cost of "delete_move (99)" functions are 0.00027552 Ether. In this comparison, we derive the fact that the optimized code for the first element deletion in Figure 2 saves about 76% costs of the non-optimized code in Figure 1. The costs of these transactions may vary when they are applied to other blockchain platforms. In the Klaytn blockchain platform, the cost of "delete_shift (99)" function is 0.00620111 Klay and the cost of "delete_move (99)" functions are 0.00033175 Klay. In this comparison, the optimized code in Figure 2 saves about 95% costs of the non-optimized code in Figure 1, 3 & 4 show similar transactions written with the python language, where the first element is deleted by ninety-nine shift operations or one move operation, respectively. In the ICON blockchain platform, the cost of "delete_shift" function in Figure 3 is 0.002751 ICX and the cost of "delete_move" functions in Figure 4 is 0.001355 ICX. In this comparison, the optimized code in Figure 4 saves about 50% costs of the non-optimized code in Figure 3.



```
@external
def delete_shift( self, target:str ):
    for i  in  range(100):
        if  target  == self._array[i]:
            if  i == 100:
                self._array.pop()
            else:
                for  k  in range(i, 99):
                    self._array[k]  =  self._array[k+1]
                break
```

Figure 3: First element deletion with 99 shift operations in a smart contract written with python.



```
@external
 def delete_move( self, target: str ):
     for i  in  range(100):
         if  target  == self._array[i]:
             lastone  =  self._array.pop()
             if  i == 100:
                 pass
             else:
                 self._array[i]  =  lastone
             break
```

Figure 4: First element deletion with one move operations in a smart contract written with python.

Through these experiments, we confirm that the cost of the same transaction varies depending on the blockchain platform. Also, the code design with the minimum number of storing operations is very important for the cost optimization of transactions in a smart contract. In contrast, in the EOS blockchain platform, the C++ code design with the fast execution and small foot print [9] will be efficient for the cost optimization of a smart contract.

Conclusions and Discussion

In this article, we summarize the current issues under studying with regards to smart contracts of blockchain systems. Also, we point out other issues never considered but essential for successful commercialization settlement of the smart contract technology. Due to short history of the smart contract technology, its software vulnerability issue is still not solved. Although many vulnerable coding patterns are verified, there are possibly more vulnerable coding patterns not verified yet. Also, the vulnerable coding patterns are affected by the programming language of smart contracts. Because the current study concentrates on the vulnerability of smart contracts. written with the solidity language, more vulnerability study is needed for other programming languages. To minimize the damage caused by attacks with unknown vulnerability, it needs to consider monitoring the real-time traffic of transactions and detecting the abnormal traffic pattern as soon as possible. Besides of vulnerability issue, another important issue is the optimization of transaction costs. We show that the efficient code design of smart contracts can significantly reduce the cost of transactions. The efficient code design of smart contracts depends on the blockchain platforms. While the minimization of storing operations is critical in most blockchain systems, the foot print size and the fast execution are critical in some blockchain system.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A1B07045806).

References

1. Satoshi Nakamoto (2009) Bitcoin: A Peer-to-Peer Electronic Cash System.
2. Gavin Wood (2014) Ethereum: A secured decentralized generalized transaction ledger. Ethereum project yellow paper 151: 1-32.
3. Puranthani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, Robin Doss (2019) Security analysis methods on Ethereum smart contract vulnerabilities: a survey. Cryptography and Security (cs.CR).
4. Ting Chen, Xiaoqi Li, Xiapu Luo, Xiaosong Zhang (2017) Under-optimized smart contracts devour your money. arXiv:1703.03994v1 [cs.SE].
5. Ethereum Smart Contract Security Best Practices.
6. Audit Checklist.
7. EOS Smart Contract Security Best Practices.
8. Discover-NEO Smart Economy.
9. Kurt Guntheroth (2016) Optimized C++: Proven techniques for heightened performance. O'Reilly Media.

DOI: 10.32474/CTCSA.2020.02.000126