Jiuzhou: a classification framework for Ethereum smart contract bugs

Feng Xiao
harleyxiao@foxmail.com
College of Computer and Information, Hohai University
Nanjing, P.R.China

Pengcheng Zhang
pczhang@hhu.edu.cn
College of Computer and Information, Hohai University
Nanjing, P.R.China

Xiapu Luo
csxluo@comp.polyu.edu.hk
Department of Computing, the Hong Kong Polytechnic University
HongKong, P.R.China

ABSTRACT

Ethereum is an open-source blockchain platform with smart contracts. Users deploy smart contracts by publishing the bytecode of smart contracts to the blockchain. Because the data in the blockchain cannot be modified, the deployed smart contract cannot be patched through code updates. However, both blockchain and smart contract are only in the initial stage, so there are inevitable bugs in the smart contracts. In recent years, researchers have tried to develop smart contract analysis tools to help detect smart contract bugs. However, there is still no formal and standardized method to classify all smart contract bugs, nor is there a set of problem smart contracts that can fully reflect the capabilities of smart contract analysis tools. In this demo paper, we detail the collection of existing smart contract bugs, categorize them according to the causes of these bugs, and present a set of problem smart contracts to demonstrate these bugs and to provide researchers with the ability to evaluate smart contract analysis tools.

CCS CONCEPTS

• Security and privacy; **• Software and its engineering → Software testing and debugging**;

KEYWORDS

Blockchain, Ethereum, Solidity, Smart contract bug

ACM Reference Format:
Feng Xiao, Pengcheng Zhang, and Xiapu Luo. 2018. Jiuzhou: a classification framework for Ethereum smart contract bugs. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Blockchain is a decentralized distributed ledger that cannot be modified after data is written. It was first used as the underlying storage technology to support Bitcoin. In recent years, unlike Bitcoin, which only supports peer-to-peer payments, Ethereum has enabled the blockchain to perform more complex operations by using smart contracts. Smart contracts are autonomous programs running on the blockchain that are typically developed in several high-level languages and then compiled into bytecode. Deploy a contract by packaging the bytecode into the blockchain in the form of a transaction. Since both blockchain and smart contracts are in their infancy, and smart contracts often handle cryptocurrency-related transactions, attacking smart contracts is often feasible and profitable.

Over the past few years, there have been several serious accidents caused by smart contract bugs, resulting in huge losses. The most notorious of which is *The DAO*, which lost $60 million due to a *re-entrancy vulnerability* [?] in its code.

The bugs in smart contracts are not just common to blockchain applications, they come from multiple levels. First, Ethereum smart contracts run on the blockchain, which are affected by the blockchain protocol. Second, the Ethereum protocol also affects smart contracts; Finally, because smart contracts are made up of code statements, common coding bugs also affect the quality of smart contracts.

Some studies have proved that the lack of a comprehensive and standardized classification framework for smart contract bugs is the reason for the proliferation of smart contract bugs [?]. Although the existing work has preliminarily summarized and classified the existing smart contract bugs [? ?], these existing works still have the following limitations:

- Statistical bugs are not comprehensive enough. Atezi et al. summarized 11 kinds of bugs, while Dingman et al. counted 49 kinds of bugs but only 24 of them classified accurately. Their work does not cover all known bugs. For example, the *Transaction order dependence* bug is missing in Atezi et al.'s statistics [?], and the *suicide contracts* bug is not included in Dingman et al.'s statistics [?].
- Some of the statistical bugs have been fixed. Ethereum has been working on fixing known bugs, which has allowed some old bugs to be fixed. These outdated bug classification can cause developers to focus unnecessarily on bugs that have been fixed. For example, in Dingman et al.'s statistical bugs [?], 13 have been fixed; in Atezi et al.'s statistical bugs [?], *stack size limit* has also been fixed.
- No matching problem smart contract set is provided. As a smart contract bugs classification framework, it is necessary to describe the causes and consequences of each bug in detail. Therefore, providing a set of problem smart contracts based on the bugs being counted can help smart contract developers and researchers better understand each bug.

To address these limitations, we ran a detailed inventory of existing smart contract bugs, assessed the need for each bug, removed bugs that had been fixed by Ethereum, and categorized existing bugs based on the causes of the bugs. Besides, we provide examples for each bug. According to our statistics, there are 49 kinds of smart contract bugs, which are divided into 19 categories. We call this classification framework and data set *Jiuzhou*, the corresponding set of smart contract bugs can be obtained at the following website: https://github.com/xf97/JiuZhou.

2 *JIUZHOU*: PROBLEM STATISTICS AND CLASSIFICATION FRAMEWORK

2.1 An overview of the classification framework of *Jihzhou*

To achieve our goals, we need to fully collect existing Ethereum smart contract bugs and understand the causes and consequences of the bugs so that we can accurately categorize the bugs. First, we get this information from relevant academic literature, the Internet and blogs. Secondly, related open-source projects are also our focus, because the open-source community plays an important role in the field of software security [?]. Specifically, we used the *smart contract analysis tool* and *smart contract security* keywords to retrieve related open-source projects on *GitHub* [?] and to focus on the smart contract bugs described in the open-source projects.

From the above resources, we collected 117 smart contract bugs with different names. We will merge the repeated bugs. The standard of merging is that the causes of the two bugs are the same and the consequences are the same too. After merging the duplicate bugs, we verify the validity of each bug (that is, each bug has not been fixed), and delete the fixed bugs (some of the bugs we counted were fixed after *Solidity* 0.5.0 version, but *Solidity* 0.4.x versions are still in use, so we still counted these bugs). Finally, we divide the remaining bugs into 19 categories according to the causes of the bugs.

In this demo paper, we only list all kinds of bugs and their classifications. Please refer to our *GitHub* page[1] for details.

2.2 Statistical and classification framework of smart contract bugs

A. Affected by blockchain protocol

Blockchain protocol will lead to several types of smart contract bugs. Specifically, there are four kinds of bugs:

(1) *Results of contract execution affected by miners.*
(2) *Transaction order depends.*
(3) *Lack of privacy on the blockchain.*
(4) *Replay attack.*

B. No checked return value

In Ethereum, if an exception occurs when a contract interacts with another address using *call*, *delegatecall* or *send* statement, the transaction will not terminate but instead notifies the result with a return value. The return value for each call should be checked and handle exceptions correctly. Specifically, there is one kind of bug:

(1) *Unhandled exception.*

C. Dynamic array operation

In *Solidity*, dynamic arrays do not automatically shorten the length and move subsequent elements after deleting elements, but only set the deleted elements as the default value of the element type. Therefore, removing dynamic array elements can be risky. Specifically, there is one kind of bug:

(1) *Delete dynamic array elements.*

D. One way flow of ether

Developers can specify whether the smart contracts will receive and send ethers. However, due to the Ethereum protocol and developers’ mistakes, smart contracts that do not want to receive ethers could be forced to receive ethers, while smart contracts that want to receive ethers couldnot be transferred out of ethers. Specifically, there are two kinds of bugs:

(1) *Force to receive ethers.*
(2) *Locked money.*

E. CEI mode is not followed when sending ethers with the call-statements

Using the *call*-statements to send ethers is flexible but dangerous because it carries all the remaining *gas*. The attacker can re-enter into the attacked contract, thus withdrawing ethers several times. The way to avoid this attack is to follow the *CEI model* (check-effect-interaction). Specifically, there is one kind of bug:

(1) *Re-entrancy vulnerability.*

F. Accuracy loss

Currently, the fixed-point number type and the decimal number type are not supported in Ethereum, so all integer division results are rounded down, which will result in loss of accuracy. At the same time, changing the long integer to the short integer will also bring the loss of accuracy. Specifically, there are two kinds of bugs:

(1) *Integer division.*
(2) *Integer truncation error.*

G. Missing or incorrect access control

Access control for contracts is required, otherwise, an attacker can easily modify the contract state and control the values of key variables. Lack of or incorrect access control can cause bugs. Specifically, there are four kinds of bugs:

(1) *Arbitrary write storage.*
(2) *Suicide contracts.*
(3) *Wasteful contracts.*
(4) *Use tx.origin for authentication.*

H. Call information controlled by external

In Ethereum, it is allowed to use the *call*-statements to call functions of other contracts or use the *delegatecall*-statements to load functions of other contracts into this contract for execution. If the call data is controlled externally, the attacker can arbitrarily specify the called function and the called function’s parameters; if the call address is controlled externally, the attacker can arbitrarily specify the called contract address. Specifically, there are three kinds of bugs:

(1) *Call data is controlled externally.*
(2) *Call address is controlled externally.*
(3) *Both the call data and the address are controlled externally.*

I. DOS by external address

[1]https://github.com/xf97/JiuZhou

In Ethereum, smart contracts are allowed to call other contracts or interact with external accounts, returning a Boolean value or terminal execution based on the call result. If the features of the contract depend on the result of the interaction with an address, the features of the contract are affected when an exception occurs at that address. Specifically, there are two kinds of bugs:

(1) *DOS by non-existent address or dead contract.*
(2) *DOS by complex fallback function.*

J. Incorrect integer operation result

Integer overflow occurs when the result of an integer operation exceeds the boundary value. Besides, the result of converting a negative integer to an unsigned integer is not correct too. These actions result in incorrect integer calculations. Specifically, there are two kinds of bugs:

(1) *Integer overflow or underflow.*
(2) *Integer sign error.*

K. DOS by gaslimit

In Ethereum, any block has the *gaslimit* attribute, which specifies the upper limit of the sum of *gas* consumed by the packaged transactions in the block. This means that if a transaction consumes too much *gas*, it is likely that the transaction will not be packaged into any block. Therefore, operations that consume too much *gas* should be avoided. Specifically, there are three kinds of bugs:

(1) *Costly loop.*
(2) *Delete dynamic array.*
(3) *Unsafe variable type inference.*

L. Uninitialized bugs

Initialization variables are very important in smart contracts. Basically, it will reduce your chances of making mistakes. Otherwise, the contract may be at risk. For example, not initializing storage variables can cause key variables of smart contracts to be unexpectedly modified. Specifically, there are three kinds of bugs:

(1) *Uninitialized state variables.*
(2) *Uninitialized local variables.*
(3) *Uninitialized storage variables.*

M. Redundant statements

To reduce the waste of network computing power, Ethereum charges a handling fee (the handling fee and its unit are also called *gas*) for each executed statement, which is converted from ethers. Some statements have no effect or redundancy in the contract, which will consume more gas. Specifically, there are three kinds of bugs:

(1) *Unused return results.*
(2) *Use uint type to compare with 0.*
(3) *Unused variables.*

N. Use worse statements

In Ethereum, you can achieve the same goal through different statements, but the use worse statements will reduce the quality of the code or increase the consumption of *gas* when achieving the same goal. Specifically, there are six kinds of bugs:

(1) *Invariants in loop.*
(2) *byte[].*
(3) *Invariant state variables are not declared constant.*
(4) *Too many digits.*
(5) *Unused public functions within a contracts should be declared external*
(6) *Unlimited compiler versions.*

O. Bad inheritance

Inheritance of contracts is allowed in Ethereum, but inheritance can cause some bugs. First, the state variable in the subclass will hide the state variable with the same name of the base class, which causes bugs in some cases. Secondly, inheritance order is very important in multi inheritance. It determines which function or variable with the same name will be used when there are functions or variables with the same name in the base classes. Wrong inheritance order will cause the features of the contract to be inconsistent with the expectation. Specifically, there are two kinds of bugs:

(1) *Hide state variables.*
(2) *Incorrect inheritance order.*

P. Failure to comply with specified standards

In Ethereum, some standards are specified to facilitate contract interaction. For example, *ERC20*, *ERC721* and other token standard interfaces specify the functions, events and other information that should be included in the token contracts. Compliance with these standard development contracts makes the code easier to understand and facilitates interaction with other contracts. Specifically, there are three kinds of bugs:

(1) *view/constant function changes contract status.* This bug was fixed after *Solidity* 0.5.0 version.
(2) *Nonstandard token interface.*
(3) *Nonstandard naming.*

Q. Write error

Write errors are situations where you write the wrong code but the contract still compiles. For example, prior to *Solidity* 0.5.0 version, it was possible to use the function which has the same name with contract's as the constructor, but miswriting the name of the constructor makes the constructor a normal function. Moreover, if +=, -= operators are incorrectly written as =+, =-, they will not cause compilation errors. Specifically, there are two kinds of bugs:

(1) *Misspelled constructor name.*
(2) *Write the wrong operators.* In our practice, the bug mentioned in the example was fixed after *Solidity* 0.5.0 version.

R. Using assembly code

Using assembly code allows developers to operate the Ethereum virtual machine at a finer granularity, but this requires a deep understanding of the smart contract. At the same time, using assembly code brings more security risks. For example, there is a risk that assembly code can be used to specify function variables of any type. Specifically, there are two kinds of bugs:

(1) *Any specified function variable type.*
(2) *Use assembly code return in the constructor.* This bug was fixed after *Solidity* 0.5.0 version.

S. Bugs with compilers prior to 0.5.0

Some of the bugs mentioned above also only existed before *Solidity* 0.5.0 version, but those bugs have clearer bug characteristics, so they are classified into the classes mentioned before. The remaining bugs that only existed before *Solidity* 0.5.0 version are in this category. Specifically, there are three kinds of bugs:

(1) *Using continue-statement in do-while-statements.*

(2) *Use deprecated structures or keywords.*
(3) *Implicit visibility level.*

3 *JIUZHOU*: PROBLEM SMART CONTRACT SET

Jiuzhou provides examples (problem smart contracts) of each bug to help smart contract researchers and developers better understand the bugs and use these contracts as test cases to evaluate the capabilities of smart contract analysis tools.

When providing the problem smart contracts, *Jiuzhou* divides the bugs into two categories:

(1) *Statement bugs.* This kind of bugs is characterized by the use of some statements, so if the statement exists, the bug exists. A typical example of statement bugs is *byte[]*, which results in additional *gas* consumption.
(2) *Context-related bugs.* This kind of bugs characteristic is that it can't be considered as a bug through the existence of some statements, so it needs to be analyzed with context. A typical example of context-related bugs is *re-entrancy vulnerability*. Sending ethers with a *call*-statement does not necessarily bring *re-entrancy vulnerability*, which needs to be analyzed in combination with context.

For the statement bugs, *Jiuzhou* will provide the smart contract containing the bug and the smart contract after fixing the bug by replacing statements. For context-related bugs, unlike the existing problem smart contract sets, which provide smart contract containing the bug and the smart contract that fixing the bug by replacing statements. *Jiuzhou* specializes in providing misleading contracts, misleading contract means:

- Smart contracts that keep statements that might introduce the bug but fix the bug by changing the context.

The purpose of providing misleading contracts is to lure smart contract analysis tools to make mistakes, which helps to distinguish the capability differences of smart contract analysis tools. For example, the code in Listing 1 still uses a *call*-statement to send ethers but fixes the *re-entrancy vulnerability* by adding the mutex variable.

```
contract EtherStore{
  bool reEntrancyMutex = false;
  uint256 public withdrawLimit = 1 ether;
  mapping( address => uint256) public lastWithdrawTime;
  mapping( address => uint256) public balances;
  bool public flag;      bytes public data;
  constructor() public{}
  function depositFunds() public payable{
  balances[msg.sender] += msg.value;    }
  function withdrawFunds(uint256 _weiToWithdraw) public{
        require(!reEntrancyMutex);
        require(balances[msg.sender] >= _weiToWithdraw &&
              _weiToWithdraw <= withdrawLimit);
 require(now>=lastWithdrawTime[msg.sender]+1 weeks);
        balances[msg.sender] -= _weiToWithdraw;
        lastWithdrawTime[msg.sender] = now;
        reEntrancyMutex = true;
        (flag, data) = msg.sender.call.value(
             _weiToWithdraw)("");
        require(flag);
        reEntrancyMutex =false; } }
```

Listing 1: a contract without *re-entrancy vulnerability* by using a mutex variable

4 RELATED WORK

A spate of Ethereum smart contract accidents has caught researchers' attention. Atezi et al. [?] identified 11 Ethereum smart contract bugs through their investigation, and their research also shows that the lack of documentation summarizing these bugs is one of the reasons for the proliferation of smart contract accidents. Dingman et al. [?] used the NIST framework to classify existing smart contract bugs, but some of the bugs with their statistics have been fixed. Tikhomirov et al. [?] divided 21 smart contract bugs into four categories: security, functional, performance and development, and developed *SmartCheck* to detect these bugs, but their statistics were not comprehensive enough to cover all smart contract bugs. Feist et al. [?] developed *Slither*, an smart contract analysis tool, to detect the 39 bugs they counted, but they failed to categorize the bugs and some of the bugs were repetitive.

5 CONCLUSION

In this demo paper, we first make a statistical analysis of the existing Ethereum smart contract bugs. Then, the bugs are classified according to the causes of these bugs. Finally, we provide a set of problem smart contracts. We carefully check every bug that we have counted, and they have not been fixed yet. As far as we know, this is the most comprehensive set of smart contract bug statistics available today.

REFERENCES

[] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust*. Springer, 164–186.
[] W. Dingman, A. Cohen, N. Ferrara, A. Lynch, P. Jasinski, P. E. Black, and L. Deng. 2019. Classification of Smart Contract Bugs Using the NIST Bugs Framework. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*. 116–123. https://doi.org/10.1109/SERA.2019.8886793
[] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 8–15.
[] GitHub Inc. [n.d.]. Open-source project repository. https://github.com/. Accessed Dec 19,2019.
[] Reza M Parizi, Ali Dehghantanha, Kim-Kwang Raymond Choo, and Amritraj Singh. 2018. Empirical vulnerability analysis of automated smart contracts security testing on blockchains. In *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 103–113.
[] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of ethereum smart contracts. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 9–16.