COALICHAIN SMART CONTRACT AUDIT REPORT

by Quill Audits, January 2019

Introduction

This Audit Report highlights the overall security of Coalichain's Smart Contract. With this report, we have tried to ensure the reliability of their smart contract by complete assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied -

Quillhash team has performed thorough testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract in order to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase we coded/conducted Custom unit tests written for each function in the contract to verify that each function works as expected. In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line
3. Deploying the code on testnet using multiple clients to run live tests

4. Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities
5. Checking whether all the libraries used in the code are on the latest version
6. Analyzing the security of the on-chain data

Summary of Coalichain Smart Contract -

The Coalichain contract is functionally an ERC-20 Smart Contract. The smart contract has all the functions of an ERC20 standard with the addition of a function to distribute tokens manually and have voting contract, anyone who have ZUZ token can participate in the voting.

Security Level references

Every issue in this report was assigned a severity level from the following:

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

High severity issues:-

1) Contract locking ether found in main contract using function deposit gas is a payable function, that will lock the ether and there is no function to withdraw ether from contract.

Status : Not yet Fixed.

Use withdraw function to transfer contract ether to any address if by mistake any ether is send to contract it will be locked forever.

2) isVoterInArray() function in Voting contract is a internal function and not called anywhere in contract, check if this function is important and need to use or implement.

Status : Not yet Fixed.

Also, you are not verifying voters before they participate in a voting, you may verify voters before they participate to ensure fair elections, else if you consider a case in which, any powerful candidate can send his ZUZ tokens to any address and will vote in favor of himself .

3) All the modifiers declared on voting contract did not used with any function if it's important to use modifier, kindly use it else you can remove them as they are just consuming storage cost.

onlyOwner(), onlyCoalichain(), onlyMainContract(), atStage(), transitionNext()

Status : Not yet Fixed.

4) updateNumTransactionToFund(), this function have no use in smart contract also .

Status : Not yet Fixed.

5) CreateBallot() function of Main contract at line 49, is calling external function, avoid state change after external calls, as external calls are very risky, you should finish internal work first before calling external functions.

Status : Not yet Fixed.

6) Function voteForCandidate() in voting contract is calling main contract function payForServices() avoid state change after external calls. Try to finish internal work first before calling external function, as external calls are risky.

Status : Not yet Fixed.

7)

Medium Severity Issues:-

1. Approval racing condition:- The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval, and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.
Line no 155:- **approve() has a race condition problem.**

approve() doesn't check if the value of allowance is equal to 0 before performing operation.
We recommend disabling users from calling this function if the value of allowance is not equal to 0.
We also recommend adding this code before performing operation:

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0));
```

2. Negative tokens approval :- Contract should not be able to approve negative value tokens but this test case is failing.
Add:

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0));
if(allowed[msg.sender][_spender] == 0){
        require(_value > 0);
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, spender, value);
        return true;
}
else {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, spender, value);
        return true;
}
```

3. VoteForCandidate() doesn't check if correlationID is correct or not, also voter is able to participate using voteForCandidate() by putting wrong id and tokens will be deducted from its account, you should check if voter using correct correlationId while participating in voting else its vote and tokens get wasted.

4.

Low Severity Issues:-

1.) Solidity version must be fixed (Always use latest Version).

It should not `pragma solidity ^0.4.21;`

Status : Not yet Fixed.

It should be `pragma solidity 0.4.21;`

2.) **Importing file in a wrong way.**

import "Types.sol"; is wrong,

Status : Not yet Fixed.

Use import `"./Types.sol";`

3.) Visibility of many functions is not declared and compiler will by default treat function as a `public`, try to declare visibility of each and every function.

4.) **SafeMath library Warnings** : functions can be restricted to pure.

5.) Use emit while

Unit Testing

Test Suite

Passed Test cases:

✓ Should correctly initialize constructor values of Coalichain Token Contract (365ms)

✓ Should correctly initialize constructor values of Main Contract (160ms)

✓ Should correctly Send ZUZ token from Contract Coalichain to Voters (393ms)

✓ Should Approve Coalichain Token Contract to spend token on the behalf of voters and ballot Creator (616ms)

✓ Should correctly deploy voting contract via Main contract by creating ballot (359ms)

✓ Should deploy Voting contract from voting ballot in main contract (84ms)

✓ Should vote for valid candidate (302ms)

❌ Should Not vote for Invalid correlationID (278ms)

✓ Should Not vote for Invalid Candidate (67ms)

❌ Should Unvote for valid candidate (54ms)

❌ Should not approve user to spend negative tokens (126ms)

✓ Should not increase approval user to spend negative tokens (60ms)

Final Result of Test:

✓ 9 passing (704ms)

❌ 3 failing

Implementation Recommendations

=> Please remove the unnecessary functions from your smart contract, smart contract contain many functions, variables and modifiers that are not used any where or have no significance in smart contract, they will only consuming gas.

=> Use function to withdraw ether as contract holds ether lock issue, also work on external calls, all the external calls are risky and try to finish internal work first before calling external contract.

=> Consider case of verify voter as that require high attention as voting will become biased if a candidate having zuz tokens can transfer to any one to influence election.

=> We recommend Coalichain team to redesign smart contract as it is not production ready and also check, all the possible cases while implementing logic for the functions, this smart contract hold multiple security issues as well as contract doesn't validate the use case of Coalichain.

=> **Warning :** Coalichain smart contract uses 6 decimal places, if you want to get register on an exchange with your token contract it will be difficult for you, as you are using 6 decimals and most exchanges will only allow 18 decimal tokens that's a standard.

Comments:

Overall, the code is clearly written, and demonstrates effective use of abstraction, separation of concerns, and modularity. Coalichain development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.

We found some critical issue and several additional issues that require the attention of the Coalichain team. Given the subjective nature of some assessments, it will be up to the Coalichain team to decide whether any changes should be made.