COALICHAIN SMART CONTRACT FINAL AUDIT REPORT

by Quill Audits, 05 January 2019

Git Commit used for Audit :

https://github.com/danielj86/CoaliSC/commit/3f0920d650467279f7ceee08776bd74ecadd3237

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, main contract hold functionality of creating vote ballot and new contract will be created for each and every voting ballot and have voting contract through which candidate is selected to vote and ZUZ tokens will be deducted from voters account.

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 depositGas() is a payable function, that will lock the ether and there is no function to withdraw ether from contract.

Status : Fixed by Developer

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 : Fixed by Developer

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 : Fixed by Developer

4) 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 : Fixed by Developer

5) 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 : Fixed by Developer

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));
```

Status : Not fixed yet

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

Use `require(_value > 0);`

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.

Status : Not fixed yet

4. Pausable Contract is not imported anywhere as well as not used, if you wish to pause, unpause voting of any other functionality you need to import it, or if it doesn't required to be use, you should remove pausable contract

Status : Not fixed yet

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.

Status : Not yet Fixed

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

Status : Not yet Fixed

5.) Use emit while event is fired.

Status : Not yet Fixed.

6.) updateNumTransactionToFund(), this function have no use in smart contract also there are many functions and variables have declared that are not used in contract.

Status : Fixed by developer.

Review Audit Contains Following Issues :

High severity issues:-

1) As you are now using 18 decimals and you want to deduct 5 ZUZ tokens who create ballot then you should use.

```
prices[uint(Types.Service.CREATE_BALLOT)] = 5 * 1 ether;
```

And Not

```
prices[uint(Types.Service.CREATE_BALLOT)] = 5000000;
```

If you use 5000000 it will only deduct 5000000/10**18 And not 5 ZUZ tokens.

Status : Not yet Fixed.

2) Similarly when you Vote for a candidate you want deduct 1 ZUZ to be deduct but from below statement, you will not be able to deduct 1 ZUZ token from voters account

`mainContract.payForService(msg.sender, 1000000);`

If you use `1000000` it will only deduct `1000000/10**18` And not 1 ZUZ tokens.

`mainContract.payForService(msg.sender, 1 ether);`

Status : Not yet Fixed.

Solidity version is not yet fixed

=> `balloutId` never used in main contract

Already mentioned in recommendations to remove unused variables, functions and modifier.

Status : Not yet Fixed.

=> `contract Pausable` never used

Already mentioned in medium severity issues to either remove or use pausable contract as pause function must be used in contract to stop functionalities of contract in case any malicious activities found.

Status : Not yet Fixed.

=> Emit is not used while emitting event in all the cases

Already mentioned in low severity issue

Example :
Line 168 coalichain contract,
line 262 coalichain contract,
line 268 coalichain contract.

Status : Not yet Fixed.

=> 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.

Already mentioned in low severity issue

Status : Not yet Fixed.

=> **SafeMath library Warnings** : functions can be restricted to pure.

Already mentioned in low severity issue

Status : Not yet Fixed.

=> 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.

Already mentioned in Medium severity issue

Status : Not yet Fixed.

Unit Testing

Test Suite

Test cases:

✓ Should correctly initialize constructor values of Coalichain Token Contract (310ms)

✓ Should correctly initialize constructor values of Main Contract (118ms)

✓ Should correctly Send ZUZ token from Contract Coalichain to Voters (450ms)

✓ Should Approve Coalichain Token Contract to spend token on the behalf of voters and ballot Creator (598ms)

✓ Should correctly deploy voting contract via Main contract by creating ballot (238ms)

✓ Should deploy Voting contract from voting ballot in main contract (121ms)

✓ Should vote for valid candidate (333ms)

╳ Should Not vote for Invalid correlationID (213ms)

✓ Should Not vote for Invalid Candidate

✓ Should set crowdsale address (90ms)

╳ Should Unvote for valid candidate (38ms)

╳ Should not approve user to spend negative tokens (103ms)

✓ Should not increase approval user to spend negative tokens (64ms)

✓ Should not decrease approval user to spend negative tokens (109ms)

✓ Should Enable transfer of tokens (106ms)

Final Result of Test:

✓ 12 passing

❌ 3 failing

Implementation Recommendations

=> Please **remove** the unnecessary functions from your smart contract, this smart contract contains many functions, variables and modifiers that are not used any where or have no significance in smart contract, they will only consume gas and add some getter functions to get info about voting status(start,end,paused) .

=> 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.

Comments:

Issues found in initial report is **not fixed yet**, all the issues are very serious issues that should not be ignored, all the recommendations and solutions are not implemented well.

QuillAudits consider each and every case during audit and try to convey developer team of coalichain to consider issues that raised during testing, all the issues are not fixed yet that are listed in initial report also new issues found in the latest commit on github , QuillAudits recommend coalichain to fix all issues that raised in initial report first and also consider issues in the final audit report.

Coalichain smart contract is not yet fully functional according to coalichain use case and also not production ready.

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.

Git Commit used for Initial Audit :

https://github.com/danielj86/CoaliSC
https://github.com/danielj86/CoaliSC/commit/e01aca3cf1520f5f38eee1958910338b6a5b5a33