QuillAudits

Audit Report

September, 2020

CXN

Contents

Introduction

This Audit Report highlights the overall security of CXN Network 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

The 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 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 -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the process.
- Analysing the complexity of the code by thorough, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests
- Analysing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: CXN Network
Website/Etherscan Code: cxn.network
Languages: Solidity (Smart contract), Javascript (Unit Testing)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Slither, Surya

Summary of Golden Goose Smart Contract

QuillAudits conducted a security audit of a smart contract of Golden Goose. Golden Goose contract is used to create the ERC20 token, which is a GOLD TOKEN, Smart contract contains basic functionalities of an ERC20 token.

Name: CXN Network
Symbol: CXN
Total supply : 300,000,000 CXN

And some advanced features other than essential functions.

- Stake
- Unstake
- Transfer fees with token distribution and burn

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient and working according to its specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contracts.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Correctness
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Security

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

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

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

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

Number of issues per severity

	Low	Medium	High
Open	0	0	0
Closed	4	1	0

High severity issues

No High severity issues

Medium Severity Issues

1. Solidity integer division might truncate. As a result, performing multiplication before division might reduce precision.

CXN._transfer(address,address,uint256) performs a multiplication on the result of a division:
-toDistribute = amount * _forStakers / 100
-_board.retPerToken = _board.retPerToken.add((toDistribute * _Scale) / _board.totalStaked)

CXN.unStake(uint256) performs a multiplication on the result of a division:
-toStakers = amount * _unstakeForStaker / 100 (CXN.sol#629)
-_board.retPerToken = _board.retPerToken.add((toStakers * _Scale) / _board.totalStaked)

Exploit Scenario

```
contract A {
   function f(uint n) public {
      coins = (oldSupply / n) * interest;
   }
}
```

If n is greater than oldSupply, coins will be zero. For example, with oldSupply = 5; n = 10, interest = 2, coins will be zero.
If (oldSupply * interest / n) was used, coins would have been 1.

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Recommedation

Consider ordering multiplication before division.

Low Severity Issues

1. Recommended to use safemath in all the operations.

```
uint256 toBurn = amount * _burnRate / 100;
uint256 _transferred = amount - toBurn;
uint256 toDistribute = amount * _forStakers / 100;
_board.retPerToken = _board.retPerToken.add((toDistribute * _Scale
/_board.totalStaked);
uint256 toBurn = amount *_burnRateStaker/100;
uint256 toStakers = amount * _unstakeForStaker/100;
_board.retPerToken = _board.retPerToken.add((toStakers*_Scale
/_board.totalStaked);
```

All the operations are vulnerable to overflow or underflow we recommend to use safemath and check value updates using assert.

Status: Fixed

2. Remove commented code from the Smart contract code.

Status: Fixed

3. _setupDecimals() is an internal function used to set up decimals for smart contract and can be called by constructor only, this function has net been used inside a constructor and have used different versions to initialize decimals of smart contract.

Status: Fixed

4. _burn function is never used in smart contract, also a smart contract contain burn function with internal access, please either make it accessible through public domain or remove it from smart contract

Status: Fixed

Unit Testing

Test Suite

Contract: CXN Token Contracts

- Should correctly initialize constructor of CXN token Contract (249ms)
- Should check the name of a token (48ms)
- Should check a symbol of a token
- Should check a decimal of a token
- Should check balance of a token
- Should check a balance of a owner contract (50ms)
- Should check a total supply of a contract (49ms)
- Should check a Min stake contract (42ms)
- Should check a total stake contract
- Should check a party stake of deployer of contract (56ms)
- Should check a party stake of deployer of contract 1st (43ms)
- Should check a party stake of deployer of contract 2nd
- Should check a party stake of deployer of contract 3rd (52ms)
- Should check a party stake of deployer of contract 4th
- Should check a party stake of deployer of contract
- Should check a party stake return of deployer of contract
- Should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4

- Should Approve accounts[1] to spend specific tokens of accounts[4]
- Should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (96ms)
- Should increase Approve accounts[4] to spend specific tokens of accounts[1]
- Should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (87ms)
- Should decrease Approve accounts[4] to spend specific tokens of accounts[1]
- Should check approval by accounts 4 to accounts 1 to spend tokens on the behalf of accounts 4 (73ms)
- Should check a balance of a beneficiary accounts[1] before sending tokens
- Should check a balance of a owner contract
- Should be able to transfer token when locking period (73ms)
- Should check a balance of a beneficiary accounts[1] before sending tokens
- Should check a balance of a owner contract
- Should be able to transfer token (71ms)
- Should check a balance of a beneficiary accounts[1] before sending tokens
- Should check a balance of a owner contract
- Should be able to transfer token (67ms)
- Should check a balance of a beneficiary accounts[1] before sending tokens
- Should check a balance of a owner contract
- Should be able to transfer token (61ms)
- Should check a balance of a beneficiary accounts[1] before sending tokens

- Should check a balance of a owner contract
- Should be able to stake token by accounts[4] (81ms)
- Should be able to stake token by accounts[3] (160ms)
- Should be able to stake token by accounts[2] (150ms)
- Should be able to transfer token by accounts 1 to accounts 5 (125ms)
- Should check a balance of a beneficiary accounts[5] after sending tokens
- Should check a balance of a beneficiary accounts[5] after sending tokens
- Should check a balance of a beneficiary accounts[4] before redeem gain tokens
- Should be able to redeemGain token by accounts[4] (64ms)
- Should check a balance of a beneficiary accounts[4] after redeem gain tokens
- Should check a balance of a beneficiary accounts[3] before redeem tokens
- Should be able to redeemGain token by accounts[4] (58ms)
- Should check a balance of a beneficiary accounts[3] after redeem gain tokens
- Should check a balance of a beneficiary accounts[2] before redeem tokens
- Should be able to redeemGain token by accounts[2] (69ms)
- Should check a balance of a beneficiary accounts[3] after redeem gain tokens
- Should be able to transferFrom accounts4 by account 1 (123ms)
- Should be able to change admin (74ms)
- Should be able to unstake token by accounts[4] (103ms)
- Should be able to set min stake token by accounts[9] (53ms)

Final Result of Test

56 Passings (3s) Passed

0 Failed

Coverage Report

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	59.09	100	100	
CXN.sol	100	59.09	100	100	
All files	100	59.09	100	100	

Slither Tool Result

```
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.5.16', '>=0.4.22<0.8.0']
        - >=0.4.22<0.8.0 (Migrations.sol#2)
        - 0.5.16 (CXN.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Pragma version>=0.4.22<0.8.0 (Migrations.sol#2) is too complex
Pragma version0.5.16 (CXN.sol#3) necessitates versions too recent to be trusted. Consider deploying with 0.5.11
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Parameter CXN.changeAdmin(address)._to (CXN.sol#286) is not in mixedCase
Parameter CXN.eliters(address,bool)._status (CXN.sol#456) is not in mixedCase
Variable CXN._Burnt_Limit (CXN.sol#162) is not in mixedCase
Variable CXN._Min_Stake (CXN.sol#163) is not in mixedCase
Variable CXN._Scale (CXN.sol#165) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
minStake() should be declared external:
        - CXN.minStake() (CXN.sol#537-539)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external
```

Implementation Recommendations

- setCompleted(uint256) should be declared external:
  - Migrations.setCompleted(uint256) (Migrations.sol#16-18)

- name() should be declared external:
  - CXN.name()

- symbol() should be declared external:
  - CXN.symbol()

- decimals() should be declared external:
  - CXN.decimals()

- changeAdmin(address) should be declared external:
  - CXN.changeAdmin(address)

- allowance(address,address) should be declared external:
  - CXN.allowance(address,address)

- approve(address,uint256) should be declared external:
  - CXN.approve(address,uint256)

- transferFrom(address,address,uint256) should be declared external:
  - CXN.transferFrom(address,address,uint256)

- increaseAllowance(address,uint256) should be declared external:
  - CXN.increaseAllowance(address,uint256)

- decreaseAllowance(address,uint256) should be declared external:
  - CXN.decreaseAllowance(address,uint256)

- stake(uint256) should be declared external:
  - CXN.stake(uint256)

- unStake(uint256) should be declared external:
  - CXN.unStake(uint256)

Implementation Recommendations

- partyDetails(address) should be declared external:
  - CXN.partyDetails(address)

- setMinStake(uint256) should be declared external:
  - CXN.setMinStake(uint256)

- minStake() should be declared external:
  - CXN.minStake()

Recommended
The use of SafeMath is recommended.
require(!(_balances[_address].sub(_lockedAmount[_address])
< requestedAmount), "Insufficient unlocked balance");

Comments

Use case of the smart contract is very well designed and Implemented. Overall, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. The CXN Network development team demonstrated high technical capabilities, both in the design of the architecture and in the implementation.
All the bugs, suggestions and recommends has been considered by CXN Network team and some of the issues they will handle to their own as those issues or calls have been handle by the only owner.

CXN

QuillAudits

448-A EnKay Square, Opposite Cyber Hub,
Gurugram, Harayana, India - 122016

audits.quillhash.com

hello@quillhash.com