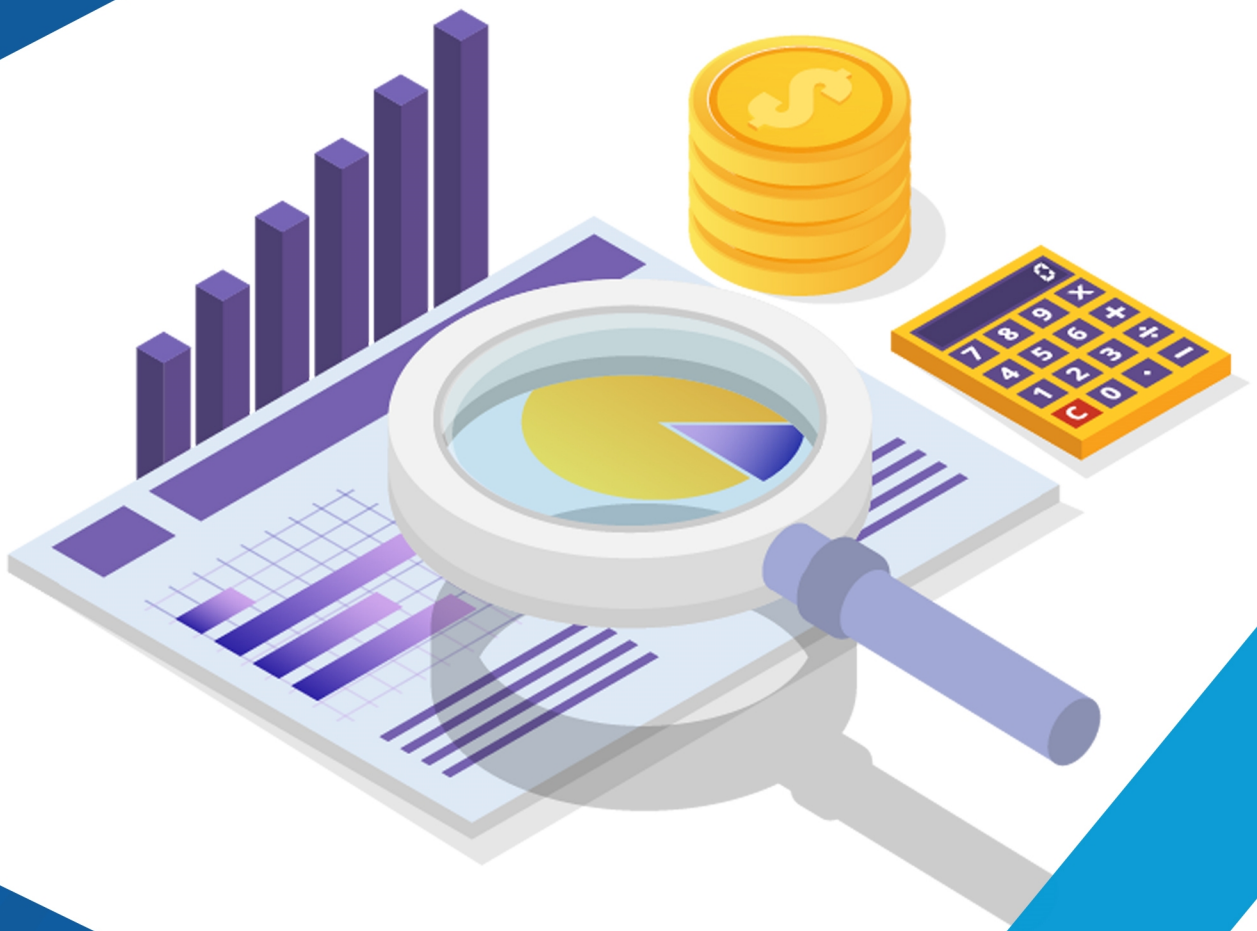
QuillAudits



Audit Report
November, 2020

CENTRALEX

Contents

Introduction

This Audit Report mainly focuses on the overall security of Centralex Smart Contract. With this report, we have tried to ensure the reliability and correctness of their Smart contract by complete and rigorous assessment of their system's architecture and the Smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed rigorous 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 whole process.
- Analysing the complexity of the code in depth and detailed, 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 any 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: Centralex - CenX
Languages: Solidity (Smart contract)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Open Zeppelin CLI

The contract code is inspired from standard OpenZeppelin contracts with slight modifications to improve gas utilization specific to the token requirement and therefore any issue reported within these contracts is of low severity.

Smart contracts are deployed using OpenZeppelin Upgrades Plugins (which uses EIP EIP1967) that can be used to upgrade their code, while preserving their address, state, and balance. This allows you to iteratively add new features to your project, or fix any bugs you may find in production.

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

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

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:

- Accuracy
- 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 level severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level 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	Recommendations
Open	1	0	0	2
Closed	3	0	0	0

Manual Audit

For this section the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality. We also tested code upgradability using Open Zeppelin CLI for following scenarios:

- State immutability
- Function signature updates
- Function logic updates

Low level severity issues

1. The pragama versions used within these contracts are not locked. Consider using version 0.5.16 for deploying the contracts. Solidity source files indicate the versions of the compiler they can be compiled with.

   ```
   pragma solidity ^0.5.0; // bad: compiles with 0.5.0 and above
   pragma solidity 0.5.0; // good : compiles w 0.5.0 only
   ```

   It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. It is also a good way to ensure contract readers about the exact version used in deploying the contract.

   Status: Fixed

2. It is advised to remove commented code before deploying the contracts:

   ```
   function __Ownable_init_unchained(address owner_) internal initializer {
       //address msgSender = _msgSender();
       _owner = owner_;
       emit OwnershipTransferred(address(0), _owner);
   }
   ```

   Status: Fixed

3. _msgData function has not been used within the contracts and therefore can be removed.

```
function _msgData() internal view  returns (bytes memory) {
    this; // silence state mutability warning without generati
    return msg.data;
}
```

4. The following list of functions should be made external for saving gas during function calls:
   renounceOwnership
   transferOwnership
   pause
   unpause

   Note: *There are four types of visibilities for functions and state variables. Functions can be specified as being external, public, internal or private, where the default is public. For state variables, external is not possible and the default is internal.*

   Status: Fixed

Medium level severity issues

No medium severity issues

High level severity issues

No high severity issues

Recommendations

1. Inside each contract, library or interface, the following order should be used as per Solidity style guide:
   - Type declarations
   - State variables
   - Events
   - Functions

```
/**
 * @dev Returns true if the contract is paused, and false otherwise.
 */
function paused() public view returns (bool) {
    return _paused;
}

/**
 * @dev Modifier to make a function callable only when the contract is not paused.
 */
modifier whenNotPaused() {
    require(!_paused, "Pausable: paused");
    _;
}
```

2. The modifier order for a function should be as follows as per Solidity style guide:
   - Visibility
   - Mutability
   - Virtual
   - Override
   - Custom modifiers

   Multiple breaches to this specification were found which should be corrected.

```
function approve(address spender, uint256 amount) whenNotPaused external  returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Centralex contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. The contract has no high severity issue but there are a number of vulnerabilities / recommendations which could be tackled in various other severity levels, and it is recommended to fix the medium severity issues before

CENTRALEX

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com