COLLATERAL

# SMART CONTRACT AUDIT

ZOKYO.

April 6th, 2021 | v. 1.0

## PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

# TECHNICAL SUMMARY

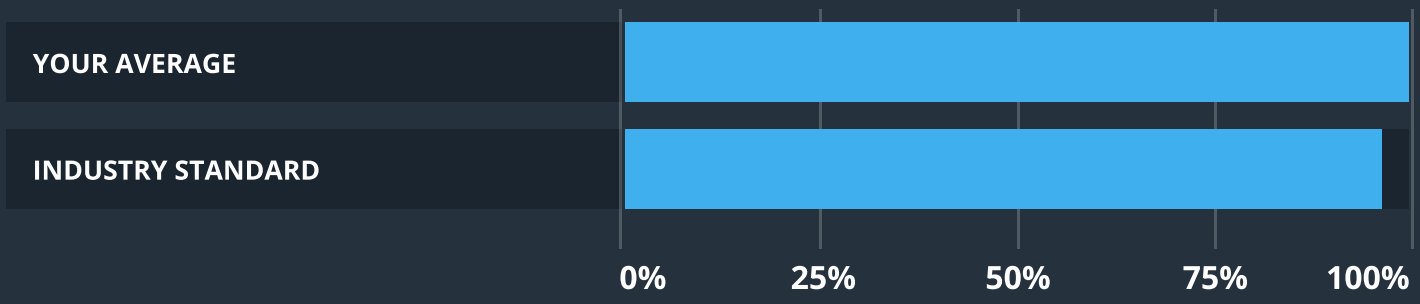
This document outlines the overall security of the COLLATERAL DEFI smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the COLLATERAL DEFI smart contract codebase for quality, security, and correctness.

## Contract Status


There were no critical issues found during the audit.

## Testable Code




The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the COLLATERAL DEFI team put in place a bug bounty program to encourage further and active analysis of the smart contract.

# TABLE OF CONTENTS

Auditing Strategy and Techniques Applied . . . . . 3
Executive Summary . . . . . 4
Structure and Organization of Document . . . . . 5
Complete Analysis . . . . . 6
Code Coverage and Test Results for all files . . . . . 13

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the COLLATERAL DEFI repository.

Repository - https://github.com/CollateralDefi/Collateral

Commit id - 5e882b4eaeca8c402bec1b6a6bc06d8c70102072

Last commit reviewed - f5e21e9c989a638542eec419c461ba8d72c062c5

**Throughout the review process, care was taken to ensure that the token contract:**

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of CollateralDefi smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

| | | | |
|---|---|---|---|
| 1 | Due diligence in assessing the overall code quality of the codebase. | 3 | Testing contract logic against common and uncommon attack vectors. |
| 2 | Cross-comparison with other, similar smart contracts by industry leaders. | 4 | Thorough, manual review of the codebase, line-by-line. |

# EXECUTIVE SUMMARY

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Nevertheless, contracts have several misleading namings, several structural inconveniences and missing blocks of documentation. The findings during the audit have a slight impact on contract performance or security, and can affect the further development and can influence some.

Also, missing minimal framework and unit testing reduce the overall security of the contract set and can not guarantee correct core logic implementation.

Despite the fact, the expected logic has several non-standard approaches and should be rewritten to get better security level and performance.

Nevertheless, all issues were successfully fixed.

## STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

### Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

### High

The issue affects the ability of the contract to compile or operate in a significant way.

### Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

### Low

The issue has minimal impact on the contract’s ability to operate.

### Informational

The issue has no impact on the contract’s ability to operate.

## COMPLETE ANALYSIS

HIGH | RESOLVED

**Missing infrastructure and test coverage.**
Project doesn't have contracts' infrastructure (no framework like Truffle/Brownie) and no unit tests or functional testing at all. Such an approach can not guarantee the security and correct logic implementation.

**Recommendation:**
Add infrastructure for the contracts and the set of unit tests to cover the core logic.

HIGH | RESOLVED

**Outdated and the lowest Solidity version.**
All contracts are written with the outdated solidity version 0.5.0. Also, the general recommendation is to use at least the highest version in the release to avoid all the bugs and vulnerabilities fixed in later releases.

**Recommendation:**
Change Solidity version to the highest in the stable release - 0.7.6.

MEDIUM | RESOLVED

**Commented code in production.**

Distribution.sol, 384
Commented and undeleted code in most cases means incomplete or obsolete or forgotten functionality which may contain a piece of missing logic and vice versa - keep obsolete logic. Such code is not acceptable for the production and can affect further development.

**Recommendation:**
Delete or review commented pieces of code.

LOW | RESOLVED

**Return value is not handled.**

Coll.sol, line 344.
Return value from removeAddress is not handled, so the transaction becomes successful instead of being reverted in case of deleting the un-listed address. This is misleading behavior.

**Recommendation:**
Handle return value and revert in case of missing address.

LOW | RESOLVED

**Unhandled situations with missing value for the address.**

Coll.sol, line 479.
_feeAccount is used in several places in the custom ERC20 contract, but in general there is no check if this account is set. This may lead to the unhandled situations of changing the balance for the zero address instead of target one. Since _feeAccount is set in the constructor of the successor, there is no harm for the system. Though inheritance from this custom ERC20 contract can not be ensured as safe, and this behavior can affect future development.

**Recommendation:**
Provide a constructor with mandatory _feeAccount parameter.

LOW | RESOLVED

**Public functions should be declared as external.**

Coll.sol
updateTaxInfo(address,uint256), addNoFeeAddress(address), removeNoFeeAddress(address) are declared as public, though they are never called in the code.

Distribution.sol
deposit(address,uint256,uint256), distribute(uint256) are declared as public, though they are never called in the code.

**Recommendation:**
Functions should be declared as external.

LOW | RESOLVED

**Optimization for better performance.**

Coll.sol, 478
Statement “amount.mul(_feePercent).div(100)” can be calculated just once instead of 4.

**Recommendation:**
Optimize the code for the calculation to be performed just once.

LOW | RESOLVED

**Missing security checks.**

Distribution.sol, 373
Method deposit does not contain any checks regarding zero address or zero amount or correct timestamp for its parameters. It can lead to incorrect settings in incorrect work of the contract.

**Recommendation:**
Add checks for incorrect values.

LOW | RESOLVED

**Unsafe token operations.**

Distribution.sol, 374, 401
Contract performs unsafe token operations without return value handling.

**Recommendation:**
Use standard SafeERC20 library for IERC20 interfaces.

LOW | RESOLVED

**Missing documentation.**

Distribution.sol - Distribution contract,
No method has documentation or doc string. Missing documentation may lead to misreading of the logic and can affect the future development.

**Recommendation:**
Add proper documentation.

LOW | RESOLVED

**Unbounded percent.**

Coll.sol
Method updateTaxInfo() does not contain any restrictions regarding the value of fee percentage. Thus, incorrect value leads to instant revert of transfer transaction.

**Recommendation:**
Add necessary checks.

INFORMATIONAL | RESOLVED

**Public values should be declared as constant.**

Coll.sol, line 591.
Collg, line 461
INITIAL_SUPPLY should be declared as constant for the storage and gas savings.

**Recommendation:**
Turn public variables to the constants.

INFORMATIONAL | RESOLVED

**Custom event in the standard interface.**

Coll.sol, line 86.
The IERC20 interface declared as the one from the standard OpenZeppelin library is extended with custom event UpdateTaxInfo(). It leads to misreading and may affect further development. The best way is to use standard OpenZeppelin interfaces without changes and move custom logic to the appropriate contract.

**Recommendation:**
Move UpdateTaxInfo() event to the COLL token contract.

INFORMATIONAL | RESOLVED

**Misleading naming.**

Coll.sol, line 307.
The system has the contract ERC20 which inherits the IERC20 interface but contains custom and extended logic. While there is no harm for the system, it provides misleading effects and can affect future development.

**Recommendation:**
Rename the custom ERC20 contract.

INFORMATIONAL | RESOLVED

**Too many digits.**

Collg.sol, line 461.
Too many digits in the literal INITIAL_SUPPLY. Such a form can contain a hidden mistake such as incorrect number of decimals in the number. It is especially dangerous in the purely tested contracts system.

**Recommendation:**
Use Ether suffix or exponential form for the constant.

INFORMATIONAL | RESOLVED

**Unreasonable return statement.**

Distribution.sol
Methods deposit() and distribute() contain unreasonable and unused return statements which always return true. Such behavior can be optimized.

**Recommendation:**
Remove unnecessary return statements.

| | CollToken | CollgToken | Distribution |
|---|---|---|---|
| Re-entrancy | Pass | Pass | Pass |
| Access Management Hierarchy | Pass | Pass | Pass |
| Arithmetic Over/Under Flows | Pass | Pass | Pass |
| Unexpected Ether | Pass | Pass | Pass |
| Delegatecall | Pass | Pass | Pass |
| Default Public Visibility | Pass | Pass | Pass |
| Hidden Malicious Code | Pass | Pass | Pass |
| Entropy Illusion (Lack of Randomness) | Pass | Pass | Pass |
| External Contract Referencing | Pass | Pass | Pass |
| Short Address/ Parameter Attack | Pass | Pass | Pass |
| Unchecked CALL Return Values | Pass | Pass | Pass |
| Race Conditions / Front Running | Pass | Pass | Pass |
| General Denial Of Service (DOS) | Pass | Pass | Pass |
| Uninitialized Storage Pointers | Pass | Pass | Pass |
| Floating Points and Precision | Pass | Pass | Pass |
| Tx.Origin Authentication | Pass | Pass | Pass |
| Signatures Replay | Pass | Pass | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass | Pass | Pass |

# CODE COVERAGE AND TEST RESULTS FOR ALL FILES

## Tests written by Zokyo Secured team

As part of our work assisting COLLATERAL DEFI in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the COLLATERAL DEFI contract requirements for details about issuance amounts and how the system handles these.

```
Tokens test coverage
  Collg token
    √ should have correct initial supply (112ms)
    √ should mint initial supply to the deployer (130ms)
    √ should have correct name (66ms)
    √ should have correct symbol (61ms)
    √ should have correct decimals (60ms)
  Coll token
    General checks
      √ should have correct initial supply (107ms)
      √ should mint initial supply to the deployer (136ms)
      √ should have correct name (65ms)
      √ should have correct symbol (64ms)
      √ should have correct decimals (55ms)
    Fees functionality
      √ should have correct owner (70ms)
      √ should have correct fees account (50ms)
      √ should revert cretion with incorrect fees account (1158ms)
      √ should have correct fee percent
      √ should update taxes info (171ms)
      √ should not update with incorrect fee account (73ms)
      √ should not update with incorrect fee (81ms)
      √ should not update tax info from not owner (75ms)
      √ should not get fee if fee is not set (204ms)
      √ should get correct fee if fee is set (294ms)
      √ should correctly add user to the whitelist (124ms)
      √ should correctly delete user from the whitelist (190ms)
      √ should revert while deleting unexistent user (73ms)
      √ should revert while deleting by not owner (137ms)
      √ should revert while adding by not owner (79ms)
      √ should not get fees from whitelisted user (343ms)
```

```
Distribution contract
  √ should have correct token address
  √ should not accept zero token address (81ms)
  √ should set distributor (57ms)
  √ should accept only owner to set distributor (61ms)
  √ should not set zero address as depositor (58ms)
  √ should decline non-distributor for deposit (68ms)
  √ should get correct arguments for the deposit (177ms)
  √ should decline non-distributor for distribution (56ms)
  √ should deposit (376ms)
  √ should decline incorrect unlock time (115ms)
  √ advance time
  √ should decline incorrect lock number (117ms)
  √ should decline incorrect claimer (58ms)
  √ should distribute (178ms)
  √ should allow claiming (161ms)

Test addresses array library
  √ should push the addr (158ms)
  √ should not push if addr exists (104ms)
  √ should delete the addr (379ms)
  √ should not delete if addr does not exist (326ms)
  √ should revert with incorrect index (86ms)
  √ should return all addresses


47 passing (9s)
```

| File | % Stmts | % Branch | % Funcs | % Lines | Uncovered Lines |
|---|---|---|---|---|---|
| contracts\ | 100 | 96 | 100 | 100 | |
| AddrArrayLib.sol | 100 | 100 | 100 | 100 | |
| COLL.sol | 100 | 90 | 100 | 100 | |
| COLLG.sol | 100 | 100 | 100 | 100 | |
| Distribution.sol | 100 | 100 | 100 | 100 | |
| contracts\test\ | 100 | 100 | 100 | 100 | |
| TestAddLib.sol | 100 | 100 | 100 | 100 | |
| All files | 100 | 96 | 100 | 100 | |

We are grateful to have been given the opportunity to work with the COLLATERAL DEFI  team.

**The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.**

Zokyo's Security Team recommends that the COLLATERAL DEFI team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.