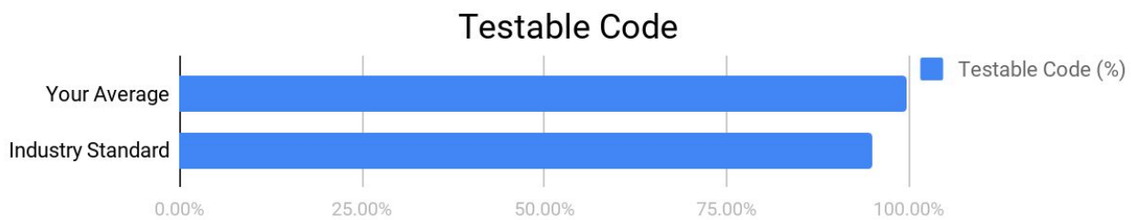
# LibraCredit Contract Audit

by Hosho, April 2018

## Executive Summary

This document outlines the overall security of LibraCredit's smart contract as evaluated by Hosho's Smart Contract auditing team. The scope of this audit was to analyze and document LibraCredit's token contract codebase for quality, security, and correctness.


All issues have been remediated. (See Complete Analysis)




Testable code is 99.72% which is higher than industry standard. (See Coverage Report)

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 Hosho recommend that the LibraCredit Team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table Of Contents

## 1. Auditing Strategy and Techniques Applied

The Hosho Team has performed a thorough review of the smart contract code, the latest version as written and updated on April 24, 2018. All main contract files were reviewed using the following tools and processes. (See All Files Covered)

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

- Implements and adheres to existing ERC-20 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; and
- Uses methods safe from reentrance attacks.
- Is not affected by the latest vulnerabilities

The Hosho Team has followed best practices and industry-standard techniques to verify the implementation of LibraCredit's token contract. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as they were discovered. Part of this work included 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.
2. Cross-comparison with other, similar smart contracts by industry leaders.
3. Testing contract logic against common and uncommon attack vectors.
4. Thorough, manual review of the codebase, line-by-line.
5. Deploying the smart contract to testnet and production networks using multiple client implementations to run live tests.

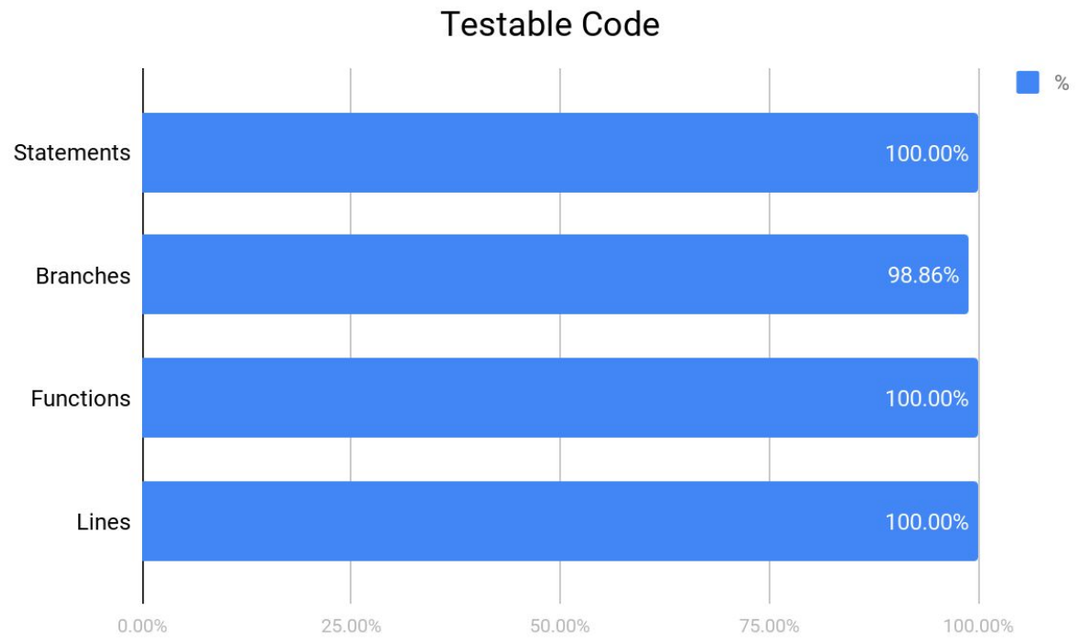
## 2. Structure Analysis and Test Results

### 2.1. Summary

The LibraCredit contracts consist of a token, token sale and whitelist. The token is a fully ERC-20 compliant token with a three phase token sale. The three phases are Deposit, Processing, and Excess. The whitelist determines who is able to purchase the tokens during the Deposit phase. Tokens are only able to be claimed during the processing stage where final WEI limits are put in place and claims cannot occur until the WEI cap is set. The LibraTokenVault contract controls the wallet addresses and time locks for the allocation and distribution of tokens for the team and reserves.

### 2.2 Coverage Report

As part of our work assisting LibraCredit in verifying the correctness of their contract code, our team was responsible for writing a unit test suite using the Truffle testing framework.




For individual files see Additional Coverage Report

### 2.3 Failing Tests

No failing tests.

See Test Suite Results for all tests.

## 3. Complete Analysis

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:

- **Informational** - The issue has no impact on the contract's ability to operate.
- **Low** - The issue has minimal impact on the contract's ability to operate.
- **Medium** - The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.
- **High** - The issue affects the ability of the contract to compile or operate in a significant way.
- **Critical** - The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

---

### 1.1. Resolved, Critical: Team Unable To Claim Tokens

LibraTokenVault

#### Explanation

The wallet vesting period is two years long and occurs in three month intervals in this contract. If more than three months after the vesting period has passed, the calculated vested amount will be greater than the allocated amount. This will prevent the team wallet from claiming any vested tokens remaining after two years and three months in the function `claimTeamReserve`.

#### Technical Example

Based on the calculations in the contract, after 1 year, 50% of the allocated tokens can be claimed as that is 12 of 24 months. However, after 2 years and 3 months, then the vesting calculation will try to claim 110%, based on 27 months out of 24 months, of the available tokens.

#### Resolution

The LibraCredit Team has placed a check that sets the maximum stage to the team vesting stage, allowing the team to collect their tokens past the wallet vesting period.

---

---

### 1.2. Resolved, High: Non-Implemented Phase Variable

LibraTokenSale

## Explanation

There is a global variable named `depositPhaseEndBlock` that is initialized and never updated, but is still utilized in various places. The most concerning is in the `onlyWhileProcessingPhaseOpen` modifier which is added to the `collectTokens` function. As the variable is initialized to zero, the check `block.number > depositPhaseEndBlock` will always pass, allowing the `collectTokens` function to be called during any of the phases of this contract, which is contrary to the system's design.

## Resolution

The LibraCredit Team has removed the global variable `depositPhaseEndBlock` and clarified the phrases to eliminate this issue.

---

### 1.3. Resolved, Medium: Opposite Function Operation

LibraTokenVault

## Explanation

The function `isStillLocked` returns the opposite of what the comment and function name seem to indicate. It reports "still has locked tokens" as true after the lock time has passed, and false when the lock time has yet to pass. This could break an external Dapp or other system that is interacting with this contract.

## Resolution

The LibraCredit Team has updated the name of the function to `canCollect` which accurately represents the functionality and resolves this issue.

---

### 1.4. Resolved, Low: Exact Check

LibraTokenVault

## Explanation

The `allocate` function reverts if contract balance is not exactly equal to a predefined amount. This allows anyone to deposit tokens to the contract and cause any subsequent `allocate` calls to fail. As an individual could easily prevent the contract from operating as intended. This would normally be a critical issue except that there is a `recoverFailedLock` function that allows for resetting the balance and starting the allocation process over again, preventing a permanently broken contract.

## Resolution

The LibraCredit Team has acknowledged the risks and given the ability to reset the locked contract as well as the limited scope of the issue, the Hosho Team accepts this issue as resolved.

---

## 4. Closing Statement

We are grateful to have been given the opportunity to work with the LibraCredit Team.

The team of experts at Hosho, having backgrounds in all aspects of blockchain, cryptography, and cybersecurity, can say with confidence that the LibraCredit contract is free of any critical issues.

**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.**

We at Hosho recommend that the LibraCredit Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

## 5. Test Suite Results

Contract: ERC-20 Tests for LibraToken

√ Should deploy a token with the proper configuration (99ms)

√ Should allocate tokens per the minting function, and validate balances (293ms)

√ Should transfer tokens from 0xd86543882b609b1791d39e77f0efc748dfff7dff to 0x42adbad92ed3e86db13e4f6380223f36df9980ef (162ms)

√ Should not transfer negative token amounts (61ms)

√ Should not transfer more tokens than you have (44ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to authorize 0x341106cb00828c87cd3ac0de55eda7255e04933f to transfer 1000 tokens (83ms)

√ Should allow 0xa3883a50d7d537cec8f9bad8e8404aa8ff3078f3 to zero out the 0x341106cb00828c87cd3ac0de55eda7255e04933f authorization (93ms)

√ Should allow 0x667632a620d245b062c0c83c9749c9bfadf84e3b to authorize 0x53353ef6da4bbb18d242b53a17f7a976265878d5 for 1000 token spend, and 0x53353ef6da4bbb18d242b53a17f7a976265878d5 should be able to send these tokens to 0x341106cb00828c87cd3ac0de55eda7255e04933f (292ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer negative tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (41ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer tokens from 0x667632a620d245b062c0c83c9749c9bfadf84e3b to 0x0 (39ms)

√ Should not transfer tokens to 0x0 (43ms)

√ Should not allow 0x53353ef6da4bbb18d242b53a17f7a976265878d5 to transfer more tokens than authorized from 0x667632a620d245b062c0c83c9749c9bfadf84e3b (42ms)

√ Should allow an approval to be set, then increased, and decreased (372ms)

Ensure 'LibraToken' defines the ERC20 Token Standard Interface

√ Should have the correct 'name' definition

√ Should have the correct 'approve' definition

√ Should have the correct 'totalSupply' definition

√ Should have the correct 'transferFrom' definition

√ Should have the correct 'decimals' definition

√ Should have the correct 'balanceOf' definition

√ Should have the correct 'symbol' definition

√ Should have the correct 'transfer' definition

√ Should have the correct 'allowance' definition

√ Should have the correct 'Transfer' definition

√ Should have the correct 'Approval' definition

Contract: LibraTokenVault

√ Should require that the deposit system has enough tokens in for the deposit phase (614ms)

√ Should require the excess phase to be open for returnExcess (488ms)

√ Should require return excess to revert if token transfer fails (622ms)

√ Should require a rate greater than zero in constructor (140ms)

√ Should require valid wallet address in constructor (225ms)

√ Should require valid token address in constructor (171ms)

Whitelist

√ Should allow owner to add an address to whitelist (115ms)

√ Should should return success if address is already whitelisted (158ms)

√ Should allow owner to add multiple addresses to whitelist (366ms)

√ Should allow owner to remove address from whitelist (327ms)

√ Should allow owner to remove multiple addresses from whitelist (974ms)

√ Should refund eth to depositor's account when removed from whitelist (485ms)

Deposits

√ Should allow the rate to be updated during deposit stage (427ms)

√ Should allow deposits for whitelisted address during deposit stage (243ms)

√ Should allow deposits via the fallback function (257ms)

√ Should require deposits to be made during the deposit phase time range (61ms)

√ Should require deposits to be from only for whitelisted addresses (154ms)

√ Should return the correct deposited amount (286ms)

√ Should report not closed before deposit phase has passed (62ms)

√ Should report as closed after the deposit phase has passed (63ms)

Withdraw

√ Should require withdraw amount ot be greater than zero (178ms)

√ Should allow withdraw of eth during the deposit phase time range (489ms)

Collect

√ Should allow allow collect during the processing phase time range (1430ms)

√ Should allow the owners to force distribute a token balance (2098ms)

√ Should have correct token balance from 1 eth deposit after collect (1001ms)

√ Should require collect to revert if called outside the processing phase time range (336ms)

√ Should require valid address to collect (84ms)

√ Should require more than zero wei to collect (321ms)

√ Should require more than zero deposit amount to collect (405ms)

√ Should refund excess eth when cap is passed (963ms)

√ Should require collect to revert if token transfer fails (909ms)

Contract: LibraTokenVault

√ Should have correct constants

√ Should report correct total balance (47ms)

Allocation

√ Should set correct allocation amounts for the reserve wallets (445ms)

√ Should revert if balance is greater than the predefined total allocation (262ms)

√ Should allow only wallets with allocated balance to get locked balance (537ms)

√ Should allow only reserve wallets to check lock (438ms)

√ Should report still locked if lock time has not passed (497ms)

√ Should report not still locked if lock time has passed (518ms)

Recover Failed Lock

√ Should require contract to not be locked to recoverFailedLock (503ms)

√ Should allow transferring tokens back to owner if locking failed (283ms)

√ Should require recover to revert if token transfer fails (384ms)

√ Should require tokens to not be already allocated (596ms)

Claim Token Reserve

√ Should require contract to be locked to claim token reserve (600ms)

√ Should require lock time to have passed for reserve wallets to claim tokens (563ms)

√ Should allow reserve wallets to claim tokens after lock time has passed (744ms)

√ Should allow only a reserve wallet to claim reserve tokens (506ms)

√ Should allow only wallets with allocation to claim reserve tokens (531ms)

√ Should allow only single reserve claim per wallet (858ms)

√ Should require claim to revert if token transfer fails (884ms)

3 Months Vesting Stages [91.25 days]

√ Should report correct vesting stage at 1 month (550ms)

√ Should report correct vesting stage at 3 months (528ms)

√ Should report correct vesting stage at 6 months (532ms)

√ Should report correct vesting stage at 1 year (540ms)

√ Should report correct vesting stage at 2 years (579ms)

Claim Team Reserve

√ Should allow only team reserve account to claim team reserve tokens (461ms)

√ Should allow claiming half of allocation after half of vesting time passed (853ms)

√ Should require waiting another vesting duration to claim more tokens (1142ms)

√ Should allow claiming full amount of tokens any time after the vesting duration has passed (900ms)

√ Should revert claim if token transfer fails (1063ms)

√ Should require greater than zero allocation to claim tokens (488ms)

## 6. All Contract Files Tested

Commit Hashes

Token: 0c5885c19cbfe02ee9d6a21298db6bfd119806d5

Vault: d4c0c0d6b7e05b5c3096aa4b354e6a81dd0ea0d9

| File | Fingerprint (SHA256) |
| --- | --- |
| contracts/LibraToken.sol | 67d4bbfe19a6b3979ff2372e8035dc390d9117aafb9a388ff67b0dc76a7c3682 |
| contracts/LibraTokenSale.sol | 28cee20f0f021d9e10aed6d312cd3c4027ad63a036fc3546a631c8b32386b1b7 |
| contracts/LibraTokenVault.sol | 13e154cd6887fd9f467dfaafa6a89073b4f1f92456df44e9fe82d81ac777df31 |
| contracts/Whitelist.sol | 907e4d8c447fc7b00c39a3435e46029ec61f1e372df2e8afdae75c57ada9f278 |

## 7. Individual File Coverage Report

| File | % Statements | % Branches | % Functions | % Lines |
|---|---|---|---|---|
| contracts/LibraToken.sol | 100.00% | 100.00% | 100.00% | 100.00% |
| contracts/LibraTokenSale.sol | 100.00% | 100.00% | 100.00% | 100.00% |
| contracts/LibraTokenVault.sol | 100.00% | 97.37% | 100.00% | 100.00% |
| contracts/Whitelist.sol | 100.00% | 100.00% | 100.00% | 100.00% |
| **All files** | **100.00%** | **98.86%** | **100.00%** | **100.00%** |