Liberty Finance

SMART CONTRACTS REVIEW

zokyo

October 29th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING LIBERTY FINANCE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issues: 0 points deducted
- 2 Low issues: 2 resolved = 0 points deducted
- 2 Informational issues: 1 resolved and 1 acknowledged = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

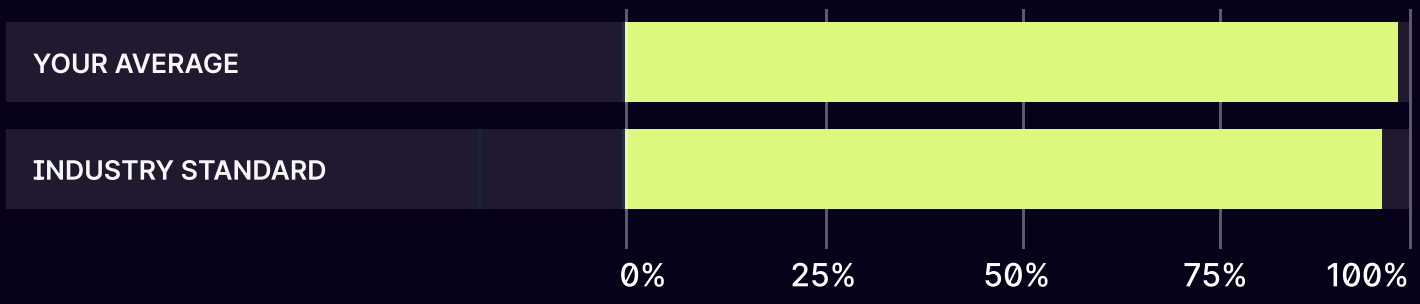
This document outlines the overall security of the Liberty Finance smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Liberty Finance smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



100% of the code is testable, 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/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Liberty Finance team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Liberty Finance repositories:

Repo #1: https://github.com/libfi-io/amana-vaults
Last commit: 95e850a4bd2c27ca9ffaab1b28038b20097ebfdc

Contracts under the scope:

- /Rewards.sol
- ./LIBXToken.sol
- ./AmanaVaultV1.sol
- ./AmanaVaultV2.sol2nd

Repository #2: https://github.com/libfi-io/libfi-token-contracts
Last commit - db93e8cc021bdfe9d697b3ea33217a999ecc4467

Contracts under the scope:

- /./LibertyFinanceToken.sol
- ./LibertyFinanceTokenV2.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Liberty Finance smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Foundry testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The Liberty Finance codebase for the audit consists of the Vault contract, Reward contract, and Token contract. The LibFi AmanaVaults protocol implements a yield-generating vault and a reward distribution system. The Vault contract implements ERC4626 and is upgradeable. It allows users to deposit LibFi tokens and receive shares, which can later be redeemed for LibFi tokens. The Reward contract manages and distributes rewards and is also upgradeable. Finally, the LibertyFinance token is an ERC20 token that is burnable, pausable, and uses the permit method.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Liberty Finance team and the Liberty Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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

FINDINGS SUMMARY

#	Title	Risk	Status
1	_decimalOffset can be set to 6	Low	Resolved
2	Method ownerDepositForDistribution should check if enough tokens present in the rewards contract	Low	Resolved
3	Upgradeable ERC20 token	Informational	Acknowledged
4	Disable initializer	Informational	Resolved

_decimalOffset can be set to 6

Contract AmanaVaultV1 inherits ERC4626 from OpenZeppelin library which uses virtual shares to mitigate inflation attack by setting decimalOffset as `0` but mentions the following as well:

While not fully preventing the attack, analysis shows that the default offset
(`0`) makes it non-profitable, as a result of the value being captured by the virtual shares (out of the attacker's
donation) matching the attacker's expected gains.

This means although the attacker will not make a profit, it still cannot be in the magnitude to prevent the attack completely.

Openzeppelin doc further mentions:
With a larger offset, the attack becomes orders of magnitude more
expensive than it is profitable. More details about the underlying math can be found
xref:erc4626.adoc#inflation-attack[here]

Hence it is advised to set decimalOffset as 6 by overriding the method in the AmanaVaultV1 contract.

Recommendation:
Set _decimalOffset as 6.

LOW-2 | RESOLVED

Method ownerDepositForDistribution should check if enough tokens present in the rewards contract

In Contract Rewards.sol, the method ownerDepsoitForDistribution(..) sets a distribution amount but does not check if enough tokens are present in the contract to distribute such an amount. In the case there are not enough tokens, distributeRewards() will fail unless tokens are transferred.

Also, this method does not need nonReentrant modifier as it can be called only by owner and there are no transfer of tokens here.

Recommendation:
Add a check to ensure enough tokens are present in the contract when the distribution amount is being set.

INFORMATIONAL-1 | ACKNOWLEDGED

Upgradeable ERC20 token

Contract LibertyFinanceToken is an ERC20 token that is upgradeable as well. It is generally not advised to make an ERC20 token upgradeable as it becomes unpredictable for token holders regarding their token shares and poses a centralization risk.

Recommendation:
Consider not upgrading the ERC20 contract unless required by DAO/Governance and use multi-sig wallet for upgrades.

Comment: The client decided to acknowledge this finding.

Disable initializer

Contract AmanaVaultV1 and Contract Rewards do not disable initializer as recommended by OpenZeppelin by adding the code as follows:

```
constructor() {
      _disableInitializers();
  }
```

Recommendation:

Disable intializers by adding the above code.

	/Rewards.sol ./LIBXToken.sol ./AmanaVaultV1.sol ./AmanaVaultV2.sol2nd
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	/./LibertyFinanceToken.sol ./LibertyFinanceTokenV2.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Nerd Labs in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Foundry testing framework.

The tests were based on the functionality of the code, as well as a review of the Nerd Labscontract requirements for details about issuance amounts and how the system handles these.

AmanaVault and Rewards Contract Tests
✓ Should not initialize Vault proxy again (57ms)
✓ Should allow multiple users to deposit LIBX tokens and receive shares (99ms)
✓ Should upgrade the contract and retain state (139ms)
✓ Should distribute rewards evenly over time (65ms)
✓ Should handle rewards distribution after the period has ended with a new deposit (294ms)
✓ Should not initialize Rewards proxy again
✓ Should set distribution end time correctly
✓ Should not set distribution end time if distribution end time is less than current timestamp
✓ Should not set distribution end time if not owner
✓ Should not set distribution amount if not owner
✓ Should not set distribution amount if distribution time is not set properly
✓ Should not set distribution amount if distribution amount is 0 (101ms)
✓ Should handle rewards distribution after the period has ended with a new deposit (115ms)

13 passing (4s)

[PASS] testCLOCKMODE() (gas: 14848)
[PASS] testClock() (gas: 13493)
[PASS] testFuzzMint(address,address,uint256) (runs: 10010, μ: 41759, ~: 32407)
[PASS] testFuzzPause(address) (runs: 10010, μ: 30610, ~: 24922)
[PASS] testFuzzRescueTokens(address,uint256,uint256,uint256) (runs: 10010, μ: 202143, ~: 200057)
[PASS] testFuzzSetMaxSupply(address,uint256,uint256) (runs: 10010, μ: 137980, ~: 138100)
[PASS] testFuzzTransferOnBlackList(address,address,uint256,uint256,uint256) (runs: 10009, μ: 221920, ~: 219905)
[PASS] testFuzzUnpause(address) (runs: 10010, μ: 45459, ~: 50380)
[PASS] testFuzzUpdateBlacklist(address,address,bool) (runs: 10010, μ: 28373, ~: 25477)
[PASS] testNonces(address) (runs: 10010, μ: 16014, ~: 16014)
[PASS] testVersion() (gas: 14870)

Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 6.42s (18.84s CPU time)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% Uncovered Lines
AmanaVaultV1.sol	100 (4/4)	100 (2/2)	100 (1/1)	100 (4/4)	
AmanaVaultV2.sol	100 (1/1)	100 (0/0)	100 (1/1)	100 (1/1)	
LIBXToken.sol	100 (1/1)	100 (0/0)	100 (1/1)	100 (1/1)	
Rewards.sol	100 (14/14)	84.62 (22/26)	100 (4/4)	92.31 (24/26)	60.61
LibertyFinanceToken.sol	100 (32/32)	100 (12/12)	100 (12/12)	100 (30/30)	
LibertyFinanceTokenV2.sol	100 (1/1)	100 (0/0)	100 (1/1)	100 (1/1)	
All files	**100**	**90**	**100**	**96.82**	

We are grateful for the opportunity to work with the Liberty Finance team.

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

Zokyo Security recommends the Liberty Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo