Flora Loans Audit Report

Prepared by bytes032

Contents

1 About bytes032

bytes032 is an independent smart contract security researcher.

His knack for identifying smart contract security vulnerabilities in a range of protocols is more than a skill; it's a passion. Committed to enhancing the blockchain ecosystem, he invests his time and energy into thorough security research and reviews. Want to know more or collaborate? bytes's always up for a chat about all things security.

Feel free to reach out on X.

2 Protocol Summary

Permissionless Money MarketFeaturing isolated risk pools and capital-efficient vaults. With Flora Loans, you can isolate your assets from the risk of any other lending pair on the platform. Users can lend compatible assets to the protocol to use as collateral for loans and earn interest.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of {X} according to the specific commit. Any modifications to the code will require a new security review.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Overview

Project	Flora Loans
Repository	flora
Commit	03a29c339bc2...
Date	July 2023

Issues Found

Severity	Count
Critical Risk	1
High Risk	2
Medium Risk	3
Low Risk	2
Informational	17
Gas Optimizations	4
Total Issues	**29**

Summary of Findings

Title	Status
[C-1] Users can receive less collateral than expected from liquidations	Resolved
[H-1] Users can use the protocol without accruing interest	Acknowledged
[H-2] Lack of Access Control in Setting Base Assets	Resolved
[M-1] Incorrect Collateral Factor Initialization in LendingPair.sol	Resolved
[M-2] Lack of Freshness Check on Chainlink Oracle Price	Resolved
[M-3] Some common non-standard ERC20 tokens are incompatible with the protocol.	Resolved
[L-1] FeeConverter wont work with tokens with a fee-on-transfer or a rebasing mechanism	Acknowledged
[L-2] Unbounded `callIncentive` can cause DoS	Resolved
[I-01] Unsafe Assumptions About Average Time Between Blocks	Acknowledged
[I-02] For readability, add commented parameter names (`Type Location /* name */`)	Resolved
[I-03] Non-standard documentation	Acknowledged

Summary of Findings

Title	Status
[I-04] Duplicated `require()`/`revert()` Checks should be refactored to a modifier or function	Acknowledged
[I-05] Event is never emitted	Resolved
[I-06] Events that mark critical parameter changes should contain both the old and the new value	Resolved
[I-07] Function ordering does not follow the Solidity style guide	Resolved
[I-08] Lack of checks in setters	Resolved
[I-09] Missing Event for critical parameters change	Resolved
[I-10] NatSpec is completely non-existent on functions that should have them	Resolved
[I-11] Incomplete NatSpec: `@param` is missing on actually documented functions	Resolved
[I-12] Consider using named mappings	Resolved
[I-13] `require()` / `revert()` statements should have descriptive reason strings	Resolved
[I-14] TODO Left in the code	Resolved
[I-15] Use Underscores for Number Literals (add an underscore every 3 digits)	Resolved
[I-16] Internal and private variables and functions names should begin with an underscore	Resolved
[I-17] Usage of floating `pragma` is not recommended	Resolved
[G-1] `a = a + b` is more gas effective than `a += b` for state variables (excluding arrays and mappings)	Acknowledged
[G-2] Use assembly to check for ‘address(0)	Acknowledged
[G-3] Use Custom Errors instead of Revert Strings to save Gas	Acknowledged
[G-4] State variables only set in the constructor should be declared `immutable`	Resolved

5 Findings

5.1 Critical Risk

[C-1] Users can receive less collateral than expected from liquidations

Context: LendingPair.sol

Impact

The impact of this vulnerability could be significant for liquidators. Liquidators might end up receiving less collateral than expected, especially when dealing with positions that are substantially underpriced or lack enough collateral to cover the liquidation.

Description

The described issue is within the `_liquidateAccount()` function in the `LendingPair` contract, where the collateral amount to be liquidated is calculated based on the repayment amount and various other parameters.

The code snippet:

Where:

- `_repayAmount` - The amount that the liquidator is repaying
- `supplyOutput` - Collateral amount returned, proportional to how much debt is being liquidated.

As seen above, if the position does not have sufficient collateral to repay the amount being liquidated or is very underpriced, it simply repays the liquidator with the remaining collateral amount.

This could cause liquidators to receive less collateral than expected, especially if they fully liquidate positions with bad debt.

Recommendation: It is recommended to add a `minCollateralAmount` parameter in the `_liquidateAccount()` function within the `LendingPair` contract.

This parameter should represent the minimum amount of collateral a liquidator is willing to receive.

If the returned collateral amount (`supplyOutput`) is less than the specified `minCollateralAmount`, the transaction should be designed to revert.

5.2 High Risk

[H-1] Users can use the protocol without accruing interest

Context: LendingPair.sol

Impact: The discovered vulnerability could allow a malicious user to perform specific actions such as depositing, withdrawing, repaying, or borrowing without accruing any interest. This can lead to financial inconsistencies within the contract and potentially unfair advantages to certain users. It directly affects the integrity of financial transactions within the system and can erode trust in the contract's fairness.

Description: In the `LendingPair` contract, the function `accrue` is called to calculate the interest for a specific token being used in various operations. The code snippet below represents the logic:

The issue lies in the reliance on `block.number` to determine the timing of interest accrual.

According to Arbitrum's documentation, block numbers and timestamps should be considered reliable only in the longer term, and they can be unreliable in the shorter term (minutes).

Multiple Arbitrum transactions (up to 50) in a single L1 block may have the same `block.number`. This behavior creates a vulnerability where a user might perform two consecutive actions within the same block without the expected interest being accrued, leading to inconsistencies in the contract's financial calculations.

Recommendation: It is advisable to avoid relying solely on `block.number` for timing assumptions in the `LendingPair` contract. Consider implementing an additional mechanism to track time such as ERC6372

[H-2] Lack of Access Control in Setting Base Assets

Context: LendingController.sol

Impact: The vulnerability allows a malicious actor to Deny Service (DoS) for the creation of permissionless pairs. By continuously setting the base asset to false, they can prevent the creation of new pairs, leading to potential disruption of the platform's functioning.

Description: When creating a pair through the PairFactory.sol contract, it makes a call to the lending controller to check if it's a base asset:

The issue here is that the `setBaseAsset` function lacks access control and can be called by anyone:

This design flaw means that a malicious actor can repeatedly call `setBaseAsset` for a specific token, setting it to false, and consequently deny the creation of permissionless pairs.

Recommendation: The recommendation is to add proper access control to the `setBaseAsset` function to ensure that only authorized users or contracts can modify the base asset status. This could be achieved by implementing a role-based access control mechanism or by requiring specific permissions to call the function. An example modification might look like:

5.3 Medium Risk

[M-1] Incorrect Collateral Factor Initialization in LendingPair.sol

Context: LendingPair.sol

Impact: The incorrect retrieval of the collateral factor may lead to unexpected behavior in the lending protocol, potentially resulting in financial inaccuracies or imbalances. If different tokens are supposed to have different collateral factors, but the system always retrieves the default, it could lead to either over-collateralization or under-collateralization, depending on the specific use case.

Description: When initializing a lending pair, the current implementation always queries the default collateral factor

ignoring any specific collateral factors that have been set for the individual tokens, such as:

This could lead to the scenario where even though the lending controller has collateral factor values for that token, they won't be considered, because it's always looking at the default collateral factor.

Recommendation: To resolve this vulnerability, the collateral factor should be retrieved based on the specific token. If a collateral factor has been set for a particular token, it should be used; otherwise, the default collateral factor should be used. The recommended code change is:

This ensures that token-specific collateral factors are considered, falling back to the default collateral factor only if no specific value has been set.

[M-2] Lack of Freshness Check on Chainlink Oracle Price

Context: UnifiedOracleAggregator.sol

Impact: The absence of a freshness check on the price timestamp from the Chainlink oracle can lead to the utilization of stale or outdated prices. This can cause incorrect or unfair asset valuations within the system, potentially leading to financial loss for users or an imbalance in the ecosystem.

Description: The code snippet provided does not include any measures to verify the freshness of the prices fetched from the Chainlink oracle:

Oracle price feeds can become stale due to a variety of reasons. Without a staleness check, there is no guarantee that the price used is recent, and the system could use an outdated price if the OCR was unable to push an update in time.

Recommendation: Apply the checks that you have implemented in `setOracle`:

[M-3] Some common non-standard ERC20 tokens are incompatible with the protocol.

Context: FeeConverter.sol

Impact: The direct usage of the ERC20's `transfer` and `transferFrom` methods can lead to unexpected behaviors with certain tokens, causing the application to be incompatible with tokens like `ZRX`, `USDT`, `BNB`, `OMG`. This poses a significant risk as it could lead to failed transactions or incorrect processing of certain tokens, especially considering that popular tokens like `USDT` exhibit these issues.

Description: The code in `FeeConverter` utilizes the standard ERC20 `transfer` and `transferFrom` methods. Two major issues arise from this:

1. **Returning False on Failure**: Some tokens (e.g., `ZRX`) do not revert on failure when calling `transfer` or `transferFrom`. Instead, they return false, which could lead to unexpected behaviors if not handled appropriately within the code.
2. **No Boolean Return**: Certain tokens like `USDT`, `BNB`, `OMG` do not return a boolean value on a `transfer` or `transferFrom` call. This discrepancy can lead to incompatibility and can cause errors or misinterpretation of the transaction status.

The application's incompatibility with these behaviors, especially with widely used tokens like `USDT`, can lead to an unpredictable and unreliable system, creating vulnerabilities in transaction handling.

Recommendation

Leveraging a well-established library such as OpenZeppelin's `SafeERC20`, and its `safeTransfer`/`safeTransferFrom` methods, would ensure consistent behavior.

5.4 Low Risk

[L-1] FeeConverter wont work with tokens with a fee-on-transfer or a rebasing mechanism

Context: FeeConverter.sol

Impact The absence of specific handling for tokens with arbitrary changes to account balances (e.g., fee-on-transfer tokens or tokens with rebasing mechanisms) can cause inconsistencies between the recorded and actual balances within the contract.

Description

Certain tokens on the blockchain have mechanisms that allow for arbitrary changes to account balances, such as fee-on-transfer tokens and tokens with rebasing mechanisms. The `FeeConverter` contract, however, does not account for these special behaviors:

1. **Fee-on-transfer Tokens**: These tokens automatically deduct a fee from the transferred amount. If not handled, the `FeeConverter` contract might record more tokens than what is actually transferred.
2. **Tokens with Rebasing Mechanisms**: These tokens can change balances periodically based on specific criteria, potentially making the actual balance in the `FeeConverter` contract different from what is accounted for.

The lack of specific handling or documentation regarding the support for these special behaviors leads to potential vulnerabilities where the `FeeConverter` contract's records might not reflect the actual token balances, causing inconsistencies and potential exploitation avenues.

Recommendation: Check the balance before and after the transfer and use the difference between the two as the actual transferred value.

[L-2] Unbounded `callIncentive` can cause DoS

Context: FeeConverter.sol

Impact: The absence of an upper bound on the `callIncentive` value allows it to be set to more than 100e18, leading to a potential transaction revert in the `floraInput` function.

Description: In the given code, the `callIncentive` is used in the calculation of the `input` variable within the `floraInput` function. The `callIncentive` is subtracted from 100e18, and if it's set to a value greater than 100e18, the subtraction will result in a negative value. This negative value will then cause the calculation to revert.

The problematic code is shown below:

And the function to set the `callIncentive` value is as follows:

There is no constraint on the value of `_value`, allowing the owner to set `callIncentive` to any value, which can lead to the above-mentioned issue.

Recommendation: To mitigate this vulnerability, it is advised to add an upper bound check to ensure that the callIncentive cannot be set to a value greater than 100e18. This can be done by adding a requirement in the setCallIncentive function, as shown below:

5.5 Informational

[I-01] Unsafe Assumptions About Average Time Between Blocks

Context: LendingPair.sol

Description

Using blocks rather than seconds to measure time is highly sensitive to changes in the average time between Ethereum blocks.

Source: Compound's audit from OZ

[I-02] For readability, add commented parameter names (`Type Location /* name */`)

Context: CircuitBreaker.sol, FeeConverter.sol, LPTokenMaster.sol, LendingController.sol

Description

When the return statement is documented but unnamed, consider adding a little comment with the name as such: `Type Location /* name */`.

As an example:

[I-03] Non-standard documentation

Context: CircuitBreaker.sol, FeeConverter.sol, FeeRecipient.sol, LPTokenMaster.sol

Description

Documentation should be between `/** */`:

However, this is not the case in the following instances:

[I-04] Duplicated `require()/revert()` Checks should be refactored to a modifier or function

Context: LendingController.sol, TransferHelper.sol, UnifiedOracleAggregator.sol

Description

[I-05] Event is never emitted

Context: LendingController.sol

Description

The following are defined but never emitted. They can be removed to make the code cleaner.

[I-06] Events that mark critical parameter changes should contain both the old and the new value

Context: UnifiedOracleAggregator.sol, LendingPair.sol, LendingController.sol

Description

This should especially be done if the new value is not required to be different from the old value.

[I-07] Function ordering does not follow the Solidity style guide

Context: LPTokenMaster.sol, LendingController.sol, LendingPair.sol, UnifiedOracleAggregator.sol

Description

According to the Solidity style guide, functions should be laid out in the following order :`constructor()`, `receive()`, `fallback()`, `external`, `public`, `internal`, `private`, but the cases below do not follow this pattern

[I-08] Lack of checks in setters

Context: FeeRecipient.sol, LendingController.sol, FeeConverter.sol

Description

Be it sanity checks (like checks against `0`-values) or initial setting checks: it's best for Setter functions to have them

[I-09] Missing Event for critical parameters change

Context: FeeConverter.sol, FeeRecipient.sol, LendingController.sol

Description

Events help non-contract tools to track changes, and events prevent users from being surprised by changes.

[I-10] NatSpec is completely non-existent on functions that should have them

Context: CircuitBreaker.sol,FeeConverter.sol,LPTokenMaster.sol

Description

Public and external functions that aren't view or pure should have NatSpec comments.

[I-11] Incomplete NatSpec: `@param` is missing on actually documented functions

Context: LPTokenMaster.sol, LendingPair.sol

Description

The following functions are missing `@param` NatSpec comments.

[I-12] Consider using named mappings

Context: CircuitBreaker.sol, LPTokenMaster.sol, LendingController.sol, LendingPair.sol

Description

Using named mappings will make it easier to understand the purpose of each mapping

[I-13] `require()` / `revert()` statements should have descriptive reason strings

Context: LendingPair.sol

Description

[I-14] TODO Left in the code

Context: LendingPair.sol

Description

TODOs may signal that a feature is missing or not ready for audit, consider resolving the issue and removing the TODO comment.

[I-15] Use Underscores for Number Literals (add an underscore every 3 digits)

Context: LendingController.sol, LendingPair.sol, UnifiedOracleAggregator.sol

Description

[I-16] Internal and private variables and functions names should begin with an underscore

Context: CircuitBreaker.sol, LendingPair.sol, LPTokenMaster.sol, BytesLib.sol

Description

According to the Solidity Style Guide, Non-`external` variable and function names should begin with an underscore

[I-17] Usage of floating `pragma` is not recommended

Description

It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated or problematic compiler version which can cause bugs, putting your smart contract's security in jeopardy.

5.6 Gas Optimization

[G-1] `a = a + b` **is more gas effective than** `a += b` **for state variables (excluding arrays and mappings)**

Context: LendingPair.sol

Description

This saves **16 gas per instance.**

[G-2] Use assembly to check for ‘address(0)

Context: LPTokenMaster.sol

Description

[G-3] Use Custom Errors instead of Revert Strings to save Gas

Context: FeeConverter.sol,LPTokenMaster.sol,LendingPair.sol,PairFactory.sol,TransferHelper.sol,UnifiedOracleAggregator.sol

Description

Custom errors are available from solidity version 0.8.4. Custom errors save **~50 gas** each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

Additionally, custom errors can be used inside and outside of contracts (including interfaces and libraries).

Source: `https://blog.soliditylang.org/2021/04/21/custom-errors/`

Consider replacing **all revert strings** with custom errors in the solution, and particularly those that have multiple occurrences:

[G-4] State variables only set in the constructor should be declared `immutable`

Context: CircuitBreaker.sol

Description

Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around **20 000 gas** per variable) and replace the expensive storage-reading operations (around **2100 gas** per reading) to a less expensive value reading (**3 gas**)