Audit Report

Bifrost Finance Leveraged Staking

v1.0

March 11, 2024

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **3**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 7
Functionality Overview 7
How to Read This Report **8**
Code Quality Criteria 9
Summary of Findings **10**
Detailed Findings **12**
1. Static calculation of weights for the claim_reward extrinsic enables DoS attack vector 12
2. Price feeder centralization risks 12
3. Multiple foreign_asset_id can be mapped to the same asset_id 13
4. Missing check that foreign_asset_id is specifically of the ForeignAsset type 14
5. Default mantissa precision can lead to an incorrect price 14
6. Missing validation for the lend_token_id 15
7. The force_update_market extrinsic could break existing markets 15
8. Static calculation of weights for the update_liquidation_fee_collateral extrinsic 16
9. Missing validation of liquidate_incentive_reserved_factor during market update 17
10. Incorrect interest is calculated if requested before market initialization 17
11. Inefficient active markets search 18
12. Inefficient market data aggregation 18
13. Redundant storage queries 19
14. Code duplication 19
15. The get_special_asset_price function consistently returns None 20
16. Inefficient reduce_reserves extrinsic execution in case of reduce_reserves parameter equal to zero 20
17. Use of magic numbers decreases maintainability 21
18. State change events are emitted even if no change has occurred 21
19. Miscellaneous comments 22

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by BIFROST FOUNDATION (BIFROST GLOBAL LTD.) to perform a security audit of Bifrost Finance Leveraged Staking.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/bifrost-finance/bifrost
Commit	`fcf0acbaa93b631a3618af07e986c5c9c1c4ed38`

Scope	The following pallets and all the inherent imports from same the repository i.e. are in scope:
	
	`.`
	`└── pallets`
	`    ├── lend-market`
	`    ├── leverage-staking`
	`    └── prices`

Fixes verified at commit	`a0aef4bb83bab000cb99b8fb96e38b58afce432d` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation if applicable.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Bifrost is a multi-chain liquidity derivatives platform in the Polkadot ecosystem. It offers standardized interest-bearing assets across the Web3 space through the use of Cross-Consensus Messaging (XCM), with the vision of seamless liquidity of staking derivatives (LSD) across any blockchain network.

As a DeFi protocol, Bifrost facilitates connections between Polkadot and various Proof of Stake (PoS) blockchains, enabling the creation, exchange, lending, and borrowing of staking derivatives.

The audit scope is limited to the `lend-market`, `leverage-staking` and `prices` pallets.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Low-Medium	The comments in the codebase were not sufficient to describe functionalities and the transaction flow.
Level of documentation	Low	The client did not provide any documentation.
Test coverage	Medium	The test coverage for the audited pallets is: • `leverage-staking`: 35.76% • `lend-market`: 67.44% • `prices`: 51.25%

Summary of Findings

No	Description	Severity	Status
1	Static calculation of weights for the `claim_reward` extrinsic enables DoS attack vector	Major	Resolved
2	Price feeder centralization risks	Minor	Acknowledged
3	Multiple `foreign_asset_id` can be mapped to the `same asset_id`	Minor	Resolved
4	Missing check that `foreign_asset_id` is specifically of the `ForeignAsset` type	Minor	Resolved
5	Default mantissa precision can lead to an incorrect price	Minor	Acknowledged
6	Missing validation for the `lend_token_id`	Minor	Resolved
7	The `force_update_market` extrinsic could break existing markets	Minor	Resolved
8	Static calculation of weights for the `update_liquidation_fee_collateral` extrinsic	Minor	Resolved
9	Missing validation of `liquidate_incentive_reserved_factor` during market update	Minor	Resolved
10	Incorrect interest is calculated if requested before market initialization	Minor	Resolved
11	Inefficient active markets search	Informational	Resolved
12	Inefficient market data aggregation	Informational	Acknowledged
13	Redundant storage queries	Informational	Acknowledged
14	Code duplication	Informational	Acknowledged
15	The `get_special_asset_price` function consistently returns `None`	Informational	Resolved
16	Inefficient `reduce_reserves` extrinsic execution in case of `reduce_reserves` parameter equal to zero	Informational	Resolved

17	Use of magic numbers decreases maintainability	Informational	Acknowledged
18	State change events are emitted even if no change has occurred	Informational	Acknowledged
19	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Static calculation of weights for the `claim_reward` extrinsic enables DoS attack vector

Severity: Major

The `claim_reward` function, defined in `pallets/lend-market/src/lib.rs:782-800`, iterates over all the existing markets to collect rewards from each of them.

However, since the number of registered markets is not bounded, the execution could exceed the computation limit if too many markets are registered.

Since the calculation of the extrinsic weights does not take into account the cardinality of the markets, the fee charged to the origin will be uncorrelated with the actual computational resource usage. This leads to a situation where the execution of multiple claims within the `claim_reward` extrinsic will cost the same as a single claim done with the `claim_reward_for_market` extrinsic.

Consequently, attackers could leverage this behavior to overload chain nodes by executing `claim_reward` on multiple markets paying a smaller fee, potentially allowing a denial-of-service attack.

Recommendation

We recommend implementing a pagination pattern, or storing an index of pending rewards as `StorageDoubleMap<AccountId, AssetId>`, to improve the performance of iterating through all markets' rewards.

Status: Resolved

2. Price feeder centralization risks

Severity: Minor

In the `prices` pallet, the `FeederOrigin` can set arbitrary emergency prices by executing the `set_price` and `reset_price` extrinsics defined in `pallets/prices/src/lib.rs:120-143`.

However, prices are not validated before being stored and every value is accepted.

As a consequence, this could lead to the following scenarios:

1. Privilege abuse: An attacker may get access to the private key of the `FeederOrigin`. He can then use the account to manipulate prices, for example by setting the price for

all assets to a small value, which would allow the attacker to liquidate all users at their loss.

2. Input error: Since the provided `Price` is not validated, incorrect values could be provided as input and stored in the contract. Even a `Price` equal to zero is allowed.

We classify this issue as minor since only a privileged account can perform the aforementioned operations.

Recommendation

We recommend performing validation on the provided prices. For instance, there could be a maximum allowed delta per time unit, such that a price of zero would not be accepted.

While this does not fully resolve the centralization issue, privilege abuse would be more involved and require multiple transactions over a longer time span. This would allow operators and users to react.

Status: Acknowledged

The client states that they use multi-signature accounts for privileged roles to reduce centralization risks.

3. Multiple `foreign_asset_id` can be mapped to the same `asset_id`

Severity: Minor

In `pallets/prices/src/lib.rs:149-157`, the `set_foreign_asset` extrinsic permits the `UpdateOrigin` origin to establish a mapping within the pallet, associating a `foreign_asset_id` with an `asset_id`.

However, it lacks a verification mechanism to ensure that the given `asset_id` is not already linked to a different `foreign_asset_id`.

As a result, an `asset_id` can be associated with multiple foreign assets which would lead to market misconfigurations.

We classify this issue with minor severity since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend introducing a verification mechanism checking that the `asset_id` is not already linked to a different `foreign_asset_id`.

Status: Resolved

The client removed `set_foreign_asset` and `get_special_asset_price` from the codebase.

4. Missing check that `foreign_asset_id` is specifically of the `ForeignAsset` type

Severity: Minor

In `pallets/prices/src/lib.rs:149-157`, the `set_foreign_asset` extrinsic enables the `UpdateOrigin` origin to establish a mapping within the pallet that associates a `foreign_asset_id` with an `asset_id`.

However, it does not verify that the provided `foreign_asset_id` is specifically of the `ForeignAsset` type; it is only identified as a `CurrencyId`.

As a result, the `UpdateOrigin` can map any asset type, not just foreign assets.

We classify this issue with minor severity since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend verifying that the provided `foreign_asset_id` is specifically of the `ForeignAsset` type.

Status: Resolved

The client removed `set_foreign_asset` and `get_special_asset_price` from the codebase.

5. Default mantissa precision can lead to an incorrect price

Severity: Minor

In `pallets/prices/src/lib.rs:179-186`, the `get_asset_mantissa` function is designed to calculate the mantissa for a given asset by attempting to retrieve the asset's decimal precision from multiple sources.

However, if it fails to find any decimal information for the specified `asset_id`, it automatically assumes a default precision of `12` decimals.

As a result, this approach poses a risk, as the function plays a crucial role in asset valuation, and an incorrect mantissa calculation would lead to the determination of an erroneous price.

Recommendation

We recommend returning an error if the decimal precision for a particular asset cannot be retrieved.

Status: Acknowledged

The client states that a misconfiguration of the decimal precision is unlikely since it is set during the asset registration in the `bifrost-asset-registry` pallet.

6. Missing validation for the `lend_token_id`

Severity: Minor

In `pallets/lend-market/src/lib.rs:656-680`, the `force_update_market` function enables the `UpdateOrigin` to assign a new `lend_token_id` to a specified market.

However, it fails to verify whether the `lend_token_id` is not already a market, a validation that is performed in the `ensure_lend_token` method in line `1843`.

As a consequence, this oversight could lead to inconsistencies in market configuration since the lend token is not designed to be used in a market.

We classify this issue with minor severity since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend verifying whether the `lend_token_id` is not already utilized in a market.

Status: Resolved

7. The `force_update_market` extrinsic could break existing markets

Severity: Minor

In `pallets/lend-market/src/lib.rs:659`, the `force_update_market` function permits the `UpdateOrigin` to forcibly replace an existing market with another, without ensuring the coherence of the provided data.

This poses a risk since `UpdateOrigin` is allowed to modify markets with any data, including malicious ones that could break invariants.

As a consequence, the current approach could result in the acceptance of markets with a `collateral_factor` or `liquidation_threshold` outside of the `[0,1]` range, which could break market operations and potentially cause a loss of funds.

We classify this issue as minor since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend removing the `force_update_market` extrinsic and utilizing the `update_market` one to handle market updates.

Status: Resolved

8. Static calculation of weights for the `update_liquidation_fee_collateral` extrinsic

Severity: Minor

In `pallets/lend-market/src/interest.rs:1123-1130`, the `update_liquidation_fee_collateral` extrinsic takes a `collaterals` vector as a parameter.

However, the calculation of the extrinsic weights does not take into account the length of the `collaterals` vector.

As a result, the `update_liquidation_fee_collateral` extrinsic will charge the same fee for vectors of different sizes leading to an uncorrelated relation between fees and computational resource usage.

We classify this issue as minor since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend dynamically calculating the weight of the `update_liquidation_fee_collateral` extrinsic based on the `collaterals` vector's cardinality.

Status: Resolved

9. Missing validation of `liquidate_incentive_reserved_factor` during market update

Severity: Minor

In `pallets/lend-market/src/lib.rs:381` the `add_market` function checks the correct allowed ranges for the provided market parameters, including `liquidate_incentive_reserved_factor`. This value is forced to be in the `(0,1)` range.

However, in the `update_market` function in `pallets/lend-market/src/lib.rs:610`, this validation is not performed.

As a consequence, when updating the market, it is possible to set the `liquidate_incentive_reserved_factor` value outside the `(0,1)` range, which will have implications in the form of incorrect calculations of incentives when liquidating positions within the `liquidated_transfer` function in `pallets/lend-market/src/lib.rs:1757`.

We classify this as minor since only a privileged account can perform the aforementioned operation.

Recommendation

We recommend validating the `liquidate_incentive_reserved_factor` in the `update_market` function.

Status: Resolved

10. Incorrect interest is calculated if requested before market initialization

Severity: Minor

In `pallets/lend-market/src/interest.rs:58`, the `get_market_status` function calculates and returns information about the queried market.

However, in case the market has not been used yet, the `last_accrued_interest_time` variable, when retrieved in line `75` from the `LastAccruedInterestTime` on-chain storage, would default to zero.

Consequently, when `get_market_status` is called without the market being previously initialized, the interest accrual period is inaccurately computed from January 1, 1970, despite the asset not being used yet leading to a wrong calculation.

Recommendation

We recommend handling the case of `last_accrued_interest_time` equal to zero in the `get_market_status` function.

Status: Resolved

11. Inefficient active markets search

Severity: Informational

The `ensure_active_market` function, defined in `pallets/lend-market/src/lib.rs:1801-1806`, searches for a market by comparing identifiers one by one in a loop through all markets with an asymptotic complexity of `O(n)`.

However, the same operation could be performed in constant time by using the `market` method, defined in `pallets/lend-market/src/lib.rs:1946`.

Consequently, this would cause the execution to incur higher costs due to unnecessary iteration.

Recommendation

We recommend replacing the iteration with a call to the `market` function.

Status: Resolved

12. Inefficient market data aggregation

Severity: Informational

The `get_lf_base_position` function, defined in `pallets/lend-market/src/lib.rs:1145-1151` iterates through all active markets, aggregating collateral for underlying assets.

However, the markets queried are not retained, despite being re-queried later by the `current_collateral_balance` function in line `1407`.

Consequently, this would cause the execution to incur higher costs due to unnecessary calculations.

Recommendation

We recommend either passing the query results down to the called functions without alteration, or redesigning the storage layout to only retrieve the necessary data at the point of call.

Status: Acknowledged

13. Redundant storage queries

Severity: Informational

The `pallets/lend-market/src/lib.rs` file exhibits redundant uses of on-chain storage.

For instance, the `AccountDeposits` storage, introduced at line `315`, undergoes redundant queries in lines `945-946`, `1280-1283`, `1312-1315`, and `2105-2106`. In each case, the storage is first checked for the key's presence, followed by a separate query to fetch the key's value.

Another example of redundant queries can be seen in line `1146`, where the storage query `Self::liquidation_free_collaterals` is called on every iteration of the loop despite having the same value on each iteration.

Minimizing the number of storage queries would enhance both performance and code clarity.

Recommendation

We recommend employing the `OptionQuery` type for business logic that needs to check for the existence of a key.

Additionally, duplicated queries in the same scope should be avoided.

Status: Acknowledged

14. Code duplication

Severity: Informational

The `pallets/lend-market/src/lib.rs` file contains multiple code duplicates:

1. The code fragments in lines `1144-1152` and `1158-1166` are nearly identical, with the only distinction being the function called on the iterated values: `collateral_asset_value` (line `1149`) in one instance and `liquidation_threshold_asset_value` (line `1163`) in the other.

2. The code fragments in lines `1312-1325` and `1280-1293` are nearly identical, with the only distinction being the type parameter: `BalanceOf` in one instance and `FixedU128` in the other.

Code duplication undermines maintainability, thereby expanding the potential for security vulnerabilities.

Recommendation

We recommend refactoring the codebase to avoid duplications. Generic type parameters and function-type parameters could be used to streamline the data flow.

Status: Acknowledged

15. The `get_special_asset_price` function consistently returns `None`

Severity: Informational

In `pallets/prices/src/lib.rs:188`, the `get_special_asset_price` function is designed to return a `TimeStampedPrice` value, but it consistently returns `None` instead.

As a consequence, this behavior is misleading and renders the function ineffective.

Recommendation

We recommend eliminating the function `get_special_asset_price` if it's not being utilized.

Status: Resolved

16. Inefficient `reduce_reserves` extrinsic execution in case of `reduce_reserves` parameter equal to zero

Severity: Informational

In `pallets/lend-market/src/lib.rs:1045`, the `reduce_reserves` extrinsic takes `reduce_amount` as a parameter.

However, if this amount is zero, the function will perform all operations unnecessarily, which is inefficient and may be misleading to the function caller.

Recommendation

We recommend verifying whether `reduce_amount` is greater than zero and, if not, returning an error that terminates the transaction.

Status: Resolved

17. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `pallets/lend-market/src/farming.rs:28`
- `pallets/lend-market/src/rate_model.rs:173`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

18. State change events are emitted even if no change has occurred

Severity: Informational

In `pallets/lend-market/src/lib.rs:778` and `pallets/lend-market/src/lib.rs:678`, when updating market parameters, the event about success is emitted regardless of whether anything has been changed.

If none of the optional values are specified, then the function will end without changing the state, but the `MarketRewardSpeedUpdated` or `UpdatedMarket` event will be emitted, which is inconsistent with the function's logic.

Recommendation

We recommend emitting an event only in case the state has been updated.

Status: Acknowledged

19. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below:

- In `pallets/lend-market/src/lib.rs:1894` there should be an `Overflow` error, not an `Underflow` one, causing `exchange_rate` to be between `(0.02, 1)` range, so the division of `amount` value and `exchange_rate` could cause overflow only.
- In `pallets/lend-market/src/lib.rs:959`, it should be used `asset`, instead of `assert`.
- In `pallets/leverage-staking/src/lib.rs:129`, `NotSupportTokenType` should be reworded to `NotSupportedTokenType`.
- In `pallets/lend-market/src/interest.rs:141`, the inequality should be greater than or equal to 0.02, not only greater, based on code implementation.

Recommendation

We suggest following the aforementioned suggestions.

Status: Resolved