Audit Report

MilkyWay Staking

v1.0

December 12, 2023

Table of Contents

License

Disclaimer

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 Osmosis Grants Company to perform a security audit of MilkyWay 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/milkyway-labs/milkyway-contracts
Commit	`44cbc3326182ac0b74f71e30dd39cbf14ffee384`
Scope	Only `contracts/staking` and `packages/milky_way` were in scope of the audit.
Fixes verified at commit	`57dc9e714efce745c171c74c03f736de41b8b050` 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.
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

MilkyWay offers a liquid staking solution for the Celestia ecosystem, initially deployed and operated on Osmosis. When users liquid-stake their TIA coins with MilkyWay, they receive an on-chain representation of their TIA staking position, known as milkTIA. This empowers Celestia token holders to access liquidity for their staked assets, enabling trading or their use as collateral in various DeFi protocols.

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	**Low-Medium**	-
Code readability and clarity	**High**	-
Level of documentation	**Medium-High**	-
Test coverage	**High**	`cargo tarpaulin` reports a test coverage for the contracts in scope of 91.02% (649/713 lines covered).

Summary of Findings

No	Description	Severity	Status
1	The exchange rate between `milkTIA` and `osmoTIA` cannot be adjusted in the event of a slashing incident	Major	Resolved
2	Repeatedly failed ICS-20 token transfers cannot be recovered	Major	Resolved
3	Users potentially withdraw incorrect token amounts if the received unstaked tokens are accounted for in the wrong batch	Major	Resolved
4	Collected protocol fees cannot be withdrawn	Minor	Resolved
5	Inefficient batch retrieval leads to increased gas costs and potential out-of-gas errors	Minor	Resolved
6	Recovering a large number of timed-out or failed ICS-20 token transfers might run out of gas	Minor	Resolved
7	Inability to mint liquid staking tokens under specific circumstances	Minor	Resolved
8	Queries might become inefficient due to iterating over all items	Minor	Resolved
9	Usage of incomparable `minimum_liquid_stake_amount` amount for liquid unstaking	Minor	Resolved
10	Missing configuration parameter validation in the `instantiate` function	Informational	Resolved
11	Duplicate tests impact maintainability	Informational	Resolved
12	“Migrate only if newer” pattern is not followed	Informational	Resolved
13	Missing additional query value	Informational	Resolved
14	Unused `treasury_address` parameter	Informational	Resolved
15	Remove `pub` keyword for internal functions	Informational	Resolved
16	Emit additional info	Informational	Resolved
17	Mitigate centralization risk	Informational	Resolved

18	Validate the circuit breaker policy	Informational	Resolved
19	`UpdateConfig` does not allow adding or removing an operator	Informational	Acknowledged
20	Configuration is loaded twice from storage in `execute_submit_batch`	Informational	Resolved
21	Missing slippage protection for minting `milkTIA` tokens may result in the user receiving fewer tokens than anticipated	Informational	Resolved
22	`execute_revoke_ownership_transfer` may not be effective	Informational	Resolved
23	Overflow checks not enabled for release profile	Informational	Acknowledged

Detailed Findings

1. The exchange rate between `milkTIA` and `osmoTIA` cannot be adjusted in the event of a slashing incident

Severity: Major

The liquid staking token `milkTIA` is supposed to be collateralized by at least 100% of staked `TIA`, specifically, `osmoTIA` tokens. In the event of a slashing incident, the staked `TIA` tokens will be slashed, and thus, the amount of tokens collateralizing `milkTIA` will be reduced. Consequently, the exchange rate between `milkTIA` and `osmoTIA` must be adjusted to reflect the new, lower collateralization ratio.

However, contrary to the outlined procedure for the slashing incidence process in the MilkyWay architecture design documentation, there is currently no mechanism in place to adjust the exchange rate, only functionality to pause the contract via the circuit breaker.

Moreover, a stale exchange rate may be problematic to external protocols that integrate `milkTIA`.

As the slashing incidence process involves communication between two separate blockchains, Osmosis and Celestia, the introduced time delay allows malicious actors to exploit the outdated exchange rate in the meantime. However, due to the ability of an operator to pause the contract via the circuit breaker, we classify this issue as major.

Recommendation

We recommend implementing the outlined procedure for the slashing incidence process by adding a new message that allows permissioned actors to adjust the exchange rate accordingly.

Additionally, we recommend exposing the contract's pause status, stored in `Config.stopped`, via the `QueryMsg::Config` query to allow external protocols to retrieve the current status and act appropriately.

Status: Resolved

The client added the ability for the admin to adjust the exchange rate by modifying the `total_native_token`, `total_liquid_stake_token`, and `total_reward_amount` values.

2. Repeatedly failed ICS-20 token transfers cannot be recovered

Severity: Major

The `osmoTIA` tokens are transferred from the staking contract on Osmosis to Celestia via ICS-20 token transfers. The in-flight packets are stored in `INFLIGHT_PACKETS` when handling the submessage reply in the `handle_ibc_reply` function, implemented in `contracts/staking/src/execute.rs:818-855`. This allows timed-out or failed transfers to be recovered and retried via the `ExecuteMsg::RecoverPendingIbcTransfers` message.

However, if a recovered transfer fails again, the transfer cannot be recovered again. This is due to the `MsgTransfer` message not being added as a submessage and, thus, the `handle_ibc_reply` reply handler not being called. Consequently, the in-flight packet is not stored, and repeated attempts to recover the transfer will not work as the packet is not found in the `INFLIGHT_PACKETS` storage map.

The same issue is also found in the `receive_rewards` function in `contracts/staking/src/execute.rs:718`, where the `ibc_transfer_msg` is not added as a submessage with the appropriate reply handler. As an actor can not purposefully force the IBC packet to fail, we classify this issue as major.

Recommendation

Consider using the same logic as in the `execute_liquid_stake` function by using submessages and handling the reply.

Status: Resolved

3. Users potentially withdraw incorrect token amounts if the received unstaked tokens are accounted for in the wrong batch

Severity: Major

The `receive_unstaked_tokens` function in `contracts/staking/src/execute.rs:721-790` is called as part of the IBC hooks, made possible through the `memo` field included in every ICS-20 token transfer packet. The MilkyWay off-chain coordinator initiates such a token transfer once the unbonding period of 21 days on Celestia has passed and the unstaked `TIA` tokens are ready to be transferred to the staking contract on Osmosis and received as `osmoTIA` tokens.

The corresponding batch, i.e., the oldest submitted batch, is determined in lines `757-760` by iterating over all batches stored in `BATCHES` and taking the first batch that matches the status `BatchStatus::Submitted`. The batch's `received_native_unstaked` value is then set to the received amount of unstaked tokens in line `782`. This value is expected to include

the accumulated staking rewards or, if a slashing event occurred, the slashed amount of tokens.

Subsequently, `received_native_unstaked` is used in the `execute_withdraw` function in lines `382-385` to calculate the amount of underlying `osmoTIA` tokens to withdraw.

However, as ICS-20 token transfers use an unordered IBC channel, the order in which the unstaked `osmoTIA` tokens are received by the staking contract is not guaranteed. Consequently, if the off-chain coordinator sends multiple unstaked token transfers to the staking contract, batches could receive the incorrect token amount, resulting in users withdrawing incorrect amounts of `osmoTIA` tokens.

Recommendation

We recommend specifying the batch ID in the `ExecuteMsg::ReceiveUnstakedTokens` message to ensure that the correct batch is updated.

Status: Resolved

4. Collected protocol fees cannot be withdrawn

Severity: Minor

Staking rewards are collected on Celestia, transferred to the MilkyWay staking contract on Osmosis, and processed within the `receive_rewards` function in `contracts/staking/src/execute.rs:652-719`. Protocol fees are calculated and deducted from the received staking rewards and accounted for in the `total_fees` storage variable in line `706`. The remaining staking rewards are then transferred back to Celestia to be staked again.

However, due to missing functionality, the collected protocol fees cannot be claimed by the protocol and remain locked in the staking contract.

Recommendation

We recommend adding a mechanism to withdraw the protocol fees from the staking contract.

Status: Resolved

5. Inefficient batch retrieval leads to increased gas costs and potential out-of-gas errors

Severity: Minor

The `receive_unstaked_tokens` function determines the oldest submitted batch in `contracts/staking/src/execute.rs:757-760` by iterating over all batches stored in

the `BATCHES` storage map until the first batch with the status `BatchStatus::Submitted` is found.

Over time, as more and more batches are added, the number of batches to iterate over will increase, potentially leading to increased gas costs and, in the worst case, to an out-of-gas error. However, new batches are only created after some pre-configured time, initially set by the MilkyWay team to three days, has elapsed. Consequently, we classify this issue as minor as creation of new batches is limited.

Similarly, in the `execute_liquid_unstake` function in `contracts/staking/src/execute.rs:175-180`, the pending batch can be retrieved from `PENDING_BATCH_ID` instead of iterating the `BATCHES` storage map.

Recommendation

We recommend specifying the submitted batch ID as a parameter to the `receive_unstaked_tokens` function and the `ExecuteMsg::ReceiveUnstakedTokens` message.

Additionally, we recommend retrieving the current pending batch ID from `PENDING_BATCH_ID` in the `execute_liquid_unstake` function.

Status: Resolved

6. Recovering a large number of timed-out or failed ICS-20 token transfers might run out of gas

Severity: Minor

The `recover` function in `contracts/staking/src/execute.rs:543-586` retrieves all timed-out and failed ICS-20 token transfer packets from the `INFLIGHT_PACKETS` storage map without the use of pagination. Consequently, if the number of in-flight packets grows very large, the function might run out of gas, and token transfers cannot be recovered anymore.

Recommendation

We recommend adding pagination to the recover function to allow for the recovery of a large number of in-flight packets.

Status: Resolved

7. Inability to mint liquid staking tokens under specific circumstances

Severity: Minor

The `compute_mint_amount` function in `contracts/staking/src/helpers.rs:38-53` calculates the amount of liquid staking tokens, `milkTIA`, to mint based on the current exchange rate between the derivative and the underlying asset. Specifically, the mint amount is calculated as `total_liquid_stake_token * native_to_stake / total_native_token`. The case where `total_native_token` is zero is handled in line `48` by minting `milkTIA` tokens 1:1 to the osmoTIA tokens.

However, if `total_liquid_stake_token` is zero while `total_native_token` is non-zero, the minting functionality does not work anymore as the resulting mint amount is zero, and the `execute_liquid_stake` in `contracts/staking/src/execute.rs:101-103` will revert with an error.

This scenario can happen if a slashing event occurs, the number of `osmoTIA` tokens is significantly reduced, and batches are submitted that cause the `total_native_token` and `total_liquid_stake_token` values to be asymmetrically reduced, resulting in `total_liquid_stake_token` to become zero while `total_native_token` is non-zero.

Recommendation

We recommend checking if `total_liquid_stake_token` is zero and minting `milkTIA` tokens 1:1 in this case as well.

Status: Resolved

8. Queries might become inefficient due to iterating over all items

Severity: Minor

The `Batches`, `PendingBatch`, `ClaimableBatches`, `IbcQueue`, and `IbcReplyQueue` queries, implemented in `contracts/staking/src/query.rs`, utilize the `range` function to iterate over all items stored in the respective storage maps. In case the number of items grows very large, the query might become inefficient due to iterating and returning all items.

Recommendation

We recommend adding the ability to paginate the results of the queries by adding `min` and `max` parameters to the queries that allow to specify the range of items to return.

Status: Resolved

9. Usage of incomparable `minimum_liquid_stake_amount` amount for liquid unstaking

Severity: Minor

In `contracts/staking/src/execute.rs:168`, the `execute_liquid_unstake` function reuses `minimum_liquid_stake_amount` to check for the minimum unstake amount. However, the `minimum_liquid_stake_amount` value is being used for staking, and it refers to the underlying asset. In unstaking, the user provides an amount of staked assets. The price of these two might not be 1:1. Hence, `minimum_liquid_stake_amount` cannot be reused in this instance.

Recommendation

We recommend using an additional config parameter `minimum_liquid_unstake_amount`.

Status: Resolved

10. Missing configuration parameter validation in the `instantiate` function

Severity: Informational

The `instantiate` function in `contracts/staking/src/contract.rs:48-143` initializes and stores the staking contract's configuration in the `CONFIG` storage variable based on the provided parameters. However, contrary to the `update_config` function in `contracts/staking/src/execute.rs:589-650`, no validation on the provided parameters is performed, potentially allowing for invalid configurations to be stored.

Recommendation

We recommend extracting the validation logic from the `update_config` function and reusing it in the `instantiate` function.

Status: Resolved

11. Duplicate tests impact maintainability

Severity: Informational

The tests for the helper functions in `contracts/staking/src/helpers.rs:102` are found to be duplicated in `contracts/staking/src/tests/helper_tests.rs`, impacting the maintainability of the codebase.

Recommendation

We recommend removing the duplicated code.

Status: Resolved

12. "Migrate only if newer" pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

It is recommended to follow the migrate "only if newer" pattern defined in the CosmWasm documentation.

Status: Resolved

13. Missing additional query value

Severity: Informational

The `query_config` function in `contracts/staking/src/query.rs:12` is missing a few parameters, such as `stopped`, `protocol_fee_config`, `multisig_address_config`, and `minimum_liquid_stake_amount`. It is best practice to allow users to query and verify the values.

Recommendation

We recommend adding the ability to query the values.

Status: Resolved

14. Unused `treasury_address` parameter

Severity: Informational

The staking contract's `instantiate` function defines and initializes the `treasury_address` configuration variable in `contracts/staking/src/contract.rs:87`. However, the value is not being used in the codebase.

Recommendation

We recommend either removing the `treasury_address` variable or adding functionality that utilizes it.

Status: Resolved

15. Remove `pub` keyword for internal functions

Severity: Informational

The `pub` keyword found in line `contracts/staking/src/helpers.rs:5` and `20` is used to publicly expose the function. However, in the codebase, the functions are only used internally.

Recommendation

We recommend removing the `pub` keyword.

Status: Resolved

16. Emit additional info

Severity: Informational

In the `execute_liquid_stake` function in `contracts/staking/src/execute.rs:75`, users who wish to stake will provide an `amount` of the underlying asset `osmoTIA` and receive `mint_amount` of the liquid staking derivative `milkTIA`. When the function completes successfully, it emits the user's staked `amount` of `osmoTIA`. However, the amount of underlying assets provided might not be the same as the staked asset received.

Recommendation

We recommend emitting an additional `mint_amount`.

Status: Resolved

17. Mitigate centralization risk

Severity: Informational

The implementation contains several centralization risks, making the admin or operator a single point of failure:

- Admin and only admin can add or remove validators
- Admin and only admin can update config
- Admin and only admin can add or remove operators
- Any single operator can stop the protocol
- Admin and only admin can resume the protocol

Recommendation

We recommend implementing traditional security mechanisms such as using multisigs, timelocks, or governance mechanisms to mitigate the centralization risk.

Status: Resolved

The client pointed out that the contract admin will be a multi-sig, and the admin role will move to be a DAO.

18. Validate the circuit breaker policy

Severity: Informational

Circuit breaker functionality implements the following policy:

- The admin cannot stop the protocol
- Any single operator can stop the protocol
- Admin and only admin can resume the protocol

At the same time, the admin can always send the `CircuitBreaker` message if needed via a controlled operator, which the admin can add via the `UpdateConfig` function.

Consequently, prohibiting the admin from stopping the contract is ineffective within this policy.

Recommendation

We recommend validating this policy within the threat model.

Status: Resolved

The client decided to adjust the circuit breaker functionality to allow the admin to pause the contract.

19. `UpdateConfig` does not allow adding or removing an operator

Severity: Informational

The implementation of the `UpdateConfig` function in `contracts/staking/src/execute.rs:621` requires the admin to provide all current

operator addresses to the `UpdateConfig` message in addition to the addresses that should get added or removed. This method is not misuse-resistant since it allows the admin to rewrite all operators in case of accidental error.

Recommendation

We recommend adding functionality that can be used to add or remove specified operators from the list.

Status: Acknowledged

The client assessed that adding and removing individual operators is not needed.

20. Configuration is loaded twice from storage in `execute_submit_batch`

Severity: Informational

In the `execute_submit_batch` function, the configuration is loaded twice from the `CONFIG` storage variable in `contracts/staking/src/execute.rs:236` and `274`.

Recommendation

We recommend removing line `contracts/staking/src/execute.rs:274` to only read the configuration once from storage.

Status: Resolved

21. Missing slippage protection for minting `milkTIA` tokens may result in the user receiving fewer tokens than anticipated

Severity: Informational

In `contracts/staking/src/execute.rs:101`, the `execute_liquid_stake function` considers the case when the calculated `mint_amount` of `milkTIA` tokens equals zero due to “an issue with rounding”.

However, it does not consider other cases when `mint_amount` is close to zero for the same reason as stated before or due to a previous exchange rate adjustment in response to a slashing event on Celestia.

Recommendation

We recommend providing a mechanism allowing users to specify the minimally acceptable number of minted tokens.

Status: Resolved

22. `execute_revoke_ownership_transfer` may not be effective

Severity: Informational

The `execute_revoke_ownership_transfer` function in `contracts/staking/src/execute.rs:500-513` is supposed to "Revoke transfer ownership, callable by the owner".

At the same time, if the owner calls `execute_transfer_ownership`, a new owner can immediately call `execute_accept_ownership`. Subsequently calling the `execute_revoke_ownership_transfer` function will not be effective to revoke the ownership that has just been transferred.

Recommendation

We recommend adding a timelock to the `execute_transfer_ownership` function.

Status: Resolved

23. Overflow checks not enabled for release profile

Severity: Informational

`contracts/staking/Cargo.toml` does not enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Acknowledged

The client acknowledged the finding, reasoning that the root-level `Cargo.toml` has overflow checks enabled.