Security Audit Report

Milkyway Celestia Restaking

v1.0

December 16, 2024

Table of Contents

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.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by DECENTO LABS PTE. LTD to perform a security audit of Milkyway Celestia Restaking.

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 targets:

Repository	https://github.com/milkyway-labs/milkyway
Commit	`1f8faae052efa52e1f7334551ce0252edaf34731`
Scope	Only the `x/liquidvesting` module was in the scope of this audit.
Fixes verified at commit	`816e90ad7e2f96d04a7a0bdb61695487021e10aa` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/milkyway-labs/celestia-parser
Commit	`aa0685161de8f93f0a39a0d7e4b01e6164540d36`
Scope	All files were in scope.
Fixes verified at commit	`1326ec64ade4c21100a61a63d6c5527056f28d0a` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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 Celestia Restaking implements functionality for investors to restake their vesting tokens across other protocols for more rewards. The Celestia Parser observes staking transaction requests to notify the `x/liquidvesting` module to mint or burn token representations to the user. These tokens can be restaked in the `x/restaking` module when the user's insurance fund amount is sufficient to cover slashing penalties.

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**, **Partially Resolved**, 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-High	-
Code readability and clarity	Medium-High	-
Level of documentation	Medium-High	-
Test coverage	Low-Medium	go cover reports a `74.9%` test coverage in `x/liquidvesting`, which excludes proto-generated files. For more information on how we compute test coverage, please see this section in the appendix. The `celestia-parser` repository reports a code coverage of `11.7%`.

Summary of Findings

No	Description	Severity	Status
1	Malicious chains can send malicious packets to steal insurance funds	Critical	Resolved
2	Excess funds are credited to the user's treasury fund	Critical	Resolved
3	Vesting tokens will not be burnt if the user performs full undelegation	Critical	Resolved
4	Incorrect rounding direction when decreasing insurance fund	Critical	Resolved
5	`MsgDepositInsurance` will fail due to blacklisted module address	Major	Resolved
6	Non-atomic execution between `SignAndBroadcastMsgExec` and database operations	Major	Resolved
7	Improper error handling in `GetValidatorCommissionsInRange`	Minor	Resolved
8	Updating insurance percentages causes unintended behaviors	Minor	Resolved
9	Storage writes to `KVStore` while an iterator exists	Minor	Resolved
10	Disable transfers between restaking targets	Informational	Resolved
11	Unnecessary validations	Informational	Resolved
12	Impractical validation	Informational	Resolved
13	Lack of validation for `height` parameter in `GetContextWithHeight`	Informational	Resolved
14	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Malicious chains can send malicious packets to steal insurance funds

Severity: Critical

In `x/liquidvesting/hooks/ibc_hooks.go:33`, the `onRecvIcs20Packet` function does not validate that the packet's destination channel ID is the intended channel ID of Celestia and MilkyWay. This is problematic because a malicious chain can send a fake `FungibleTokenPacketData` packet and spoof itself as Celestia to arbitrarily increase the attacker's insurance funds in lines `90-107`.

The attacker will set `MsgDepositInsurance` to trick the `onRecvIcs20Packet` function into crediting real-world value tokens to them. After that, the attacker calls the `WithdrawInsuranceFund` message and steals funds that belong to other depositors, causing a loss of funds.

Recommendation

We recommend validating the packet's destination channel ID as the intended channel ID established between Celestia and Milkway.

Status: Resolved

2. Excess funds are credited to the user's treasury fund

Severity: Critical

In `modules/distribution/utils.go:159`, the insurance deposit amount is determined from the `missingFromInsuranceFund` variable. However, while the variable represents the difference in the amount required to reach the insurance percentage threshold, there is a possibility that the user rewards are less than the shortage amount, but this scenario is not handled.

Consequently, more funds may be distributed to the user's insurance fund at the expense of other users' rewards, causing a loss of funds.

Recommendation

We recommend computing the amount to distribute as `min(missingFromInsuranceFund, userRewards)`.

Status: Resolved

3. Vesting tokens will not be burnt if the user performs full undelegation

Severity: Critical

In `modules/staking/periodic_operations.go:25-32`, the user's staking representation is processed based on their delegated amount stored in the database. The user's delegation amount is queried for every block via a `QueryDelegationRequest` message and is only stored if the `DelegationResponse` is valid (see `modules/staking/utils.go:56-69`).

The issue is that if a user has an existing delegation and performs a full undelegation, their position will be removed completely. This means the `DelegationResponse` will return `nil` during `QueryDelegationRequest` queries, preventing the database from updating the user's delegated amount.

Consequently, the `getUserLatestTotalDelegatedAmount` function will incorrectly return the user's old delegated amount, causing the Celestia Parser to wrongly assume the user still has an active delegation. This prevents the `MsgBurnVestedRepresentation` message from being dispatched, ultimately allowing users to receive restaking rewards without active delegations.

Recommendation

We recommend modifying the implementation to account for the above full undelegation scenario. For example, if the user's staking position cannot be queried, the user's delegated amount should be stored as zero in the database.

Status: Resolved

4. Incorrect rounding direction when decreasing insurance fund

Severity: Critical

In `x/liquidvesting/keeper/send_restriction.go:83-86`, the `SendRestrictionFn` decreases a user's insurance fund's `Used` amount, as a transfer of liquid vesting tokens is occurring from a restaking target to a non-restaking target.

Because the detracted amount required is calculated via the `getRequiredAmountInInsuranceFund` function, which always rounds up through usage of the `ceil` function, the following scenario can arise:

1. Assume the module's `InsurancePercentage` is set to `2%` and that Alice's insurance fund has `insuranceFund.Used() == 0`.
2. Alice transfers `100` tokens to a restaking target:
    a. The `required` variable is calculated as `ceil(100 * 2 / 100) = 2`
    b. The `isToRestakingTarget` case is executed, calling `insuranceFund.AddUsed(2)`

3. Alice withdraws `1` token from the staking target:
    a. The `required` variable is calculated as `ceil(1 * 2 / 100) = 1`
    b. The `isFromRestakingTarget` case is executed, calling `insuranceFund.DecreaseUsed(1)`

At this point, Alice has achieved a `99` amount deposit within the staking target, with only `1` token being accounted for as used. Note that Alice can also repeat step 3 to achieve a `98` and `0` tokens case, wiping the used insurance fund altogether while holding a `98` amount deposit on the restaking target.

Recommendation

We recommend modifying the implementation such that in the `isFromRestakingTarget` case, the `required` variable is computed with the `floor` function so the user's insurance fund is decreased correctly. Additionally, the protocol must allow such a value to be `0` only when the withdrawal from the restaking target is not partial, as it would allow repeated withdrawals that do not decrease the used assets within the insurance fund.

Status: Resolved

5. `MsgDepositInsurance` will fail due to blacklisted module address

Severity: Major

In `x/liquidvesting/hooks/ibc_hooks.go:47`, the `onRecvIcs20Packet` function can only be called if the packet's receiver is the `x/liquidvesting` module's address. This is problematic because the `x/liquidvesting` module is not configured as a whitelisted address in `app/app.go:1484-1496`.

Consequently, all `MsgDepositInsurance` IBC messages will fail because the `x/liquidvesting` module is not allowed to receive funds, preventing the protocol from working as intended.

Recommendation

We recommend whitelisting the `x/liquidvesting` module's address in the bank keeper.

Status: Resolved

6. Non-atomic execution between `SignAndBroadcastMsgExec` and database operations

Severity: Major

In `modules/staking/periodic_operations.go:92-107` and `modules/distribution/periodic_operations.go:116-195`, the `SignAndBroadcastMsgExec` operation and subsequent database updates are not performed atomically. This approach introduces a significant risk of data inconsistency, as it's possible for `SignAndBroadcastMsgExec` to succeed while database operations fail. In such cases, double-spending is possible because the protocol state might not accurately reflect the latest transaction outcome.

For instance:

- The method `SignAndBroadcastMsgExec` executes successfully, leading to updates in the MilkyWay chain.
- However, if subsequent calls to update the database fail, the database will not accurately reflect this update.

Note that the `SignAndBroadcastMsgExec` method uses the `CheckTx` method, which does not guarantee inclusion into a block even if it returns no error. This is because passing `CheckTx` means the transaction meets the requirements to be added to the mempool. However, even if the implementation used the `DeliverTx` method and the execution of the sent message on the MilkyWay would be successful, one of the consequent database operations might fail, resulting in the execution of the message on the chain and processing of the same message on the next iteration again.

Recommendation

We recommend the following improvements to enhance the implementation most straightforwardly, assuming that the `celestia-parser` is deployed on a single node. First, commit the changes to the database using the transaction management mechanism; if this completes successfully, send the message using a method based on the `DeliverTx` method call. Then, check whether the message was executed. If it was not executed, roll back the database changes.

Alternatively, consider implementing a two-phase commit-type protocol adapted to the system design.

Status: Resolved

7. Improper error handling in `GetValidatorCommissionsInRange`

Severity: Minor

In `database/commissions.go:80`, the `GetValidatorCommissionsInRange` function in the database layer handles errors incorrectly. Currently, it does not check for specific error cases, such as `sql.ErrNoRows`, which can lead to unclear or misleading error handling. When no rows are returned, `sql.ErrNoRows` is raised, and handling this specific error allows for more accurate error reporting and debugging.

Recommendation

We recommend handling the `sql.ErrNoRows` error as it is performed in all other functions in the code (e.g., in `database/commissions.go:64`).

Status: Resolved

8. Updating insurance percentages causes unintended behaviors

Severity: Minor

In `x/liquidvesting/keeper/msg_server.go:155`, the `UpdateParams` function allows the authority to update `Params.InsurancePercentage`, which is the percentage of insurance funds required for restaking.

However, if the authority increases the percentage value, an underflow error will occur in the `DecreaseUsed` function because it tries to release a greater insurance amount than intended (see `x/liquidvesting/keeper/send_restriction.go:85`). On the other hand, if the authority decreases the percentage value, the `Used` insurance fund amount will not be reflected automatically, preventing users from withdrawing any excess amount via the `WithdrawInsuranceFund` message.

We classify this issue as minor because it can only caused by the authority, which is a privileged account.

Recommendation

We recommend only allowing the insurance percentage to be updated via state migrations so the users' insurance amount can be updated concurrently.

Status: Resolved

9. Storage writes to `KVStore` while an iterator exists

Severity: Minor

In `x/liquidvesting/keeper/burn.go:185-196`, the `DequeueAllBurnCoinsFromUnbondingQueue` method deletes elements from the `store` while an iterator exists over its domain. However, the iterator type's contract states that this should not occur.

Recommendation

Although we did not find a security vulnerability introduced by this issue, we recommend storing all to-be-deleted keys within a slice within the for loop's body and flushing the deletes to store only after `iter.Close` is called.

Status: Resolved

10. Disable transfers between restaking targets

Severity: Informational

In `x/liquidvesting/keeper/send_restriction.go:38-45`, the `SendRestrictionFn` function allows for the `isToRestakingPlatform` and `isFromRestakingPlatform` conditions to hold, which should not be allowed. No impact is warranted if such a transfer is attempted, as `x/liquidvesting/keeper/send_restriction.go:77` ensures that non-zero value transfers are caught within this edge case.

Recommendation

We recommend disallowing the shown case by returning an error if both flags are `true`.

Status: Resolved

11. Unnecessary validations

Severity: Informational

In `modules/distribution/handle_blocks.go:69-71`, the `getValidatorEarnedCommissionsAtHeight` method performs a validation to verify that `height > 1`. This check is unnecessary as the shown method is only used within `refreshCommissions`, which has already verified this exact condition at `modules/distribution/handle_blocks.go:27-29`.

Recommendation

We recommend removing the unnecessary validation.

Status: Resolved

12. Impractical validation

Severity: Informational

In `x/liquidvesting/types/ibc-hooks.go:46-50`, the `ValidateBasic` function of `MsgDepositInsurance` checks that `deposit.Amount.Denom` must be equal to `msg.Amounts[i].Amount.Denom`. This validation is impractical because the `deposit` variable is accessed from `msg.Amounts`, indicating that the value will always be the same and that the condition will never be false.

Recommendation

We recommend removing the validation.

Status: Resolved

13. Lack of validation for `height` parameter in `GetContextWithHeight`

Severity: Informational

In `utils/context.go:9-10`, the `GetContextWithHeight` function currently accepts the `height` parameter without any validation. Passing an invalid `height`, such as a negative number or values outside the valid range for the chain, can potentially lead to unexpected behavior or errors within the resulting context.

This lack of validation introduces inconsistencies, as some caller functions check `height` before calling `GetContextWithHeight` (e.g., `modules/distribution/handle_blocks.go:69`), while others do not (e.g., `modules/staking/utils.go:53`).

Recommendation

We recommend validating the `height` parameter consistently.

Status: Resolved

14. Miscellaneous comments

Severity: Informational

The following are some recommendations to improve the overall code quality and readability:

- In `modules/staking/utils.go:67`, log an error if `res.DelegationResponse` is `nil`.
- In `modules/staking/periodic_operations.go:89` and `modules/distribution/periodic_operations.go:117`, check if the length of `msgs` is more than zero before calling `SignAndBroadcastMsgExec`. This is required to prevent sending an empty transaction that will fail.
- Review, address, and adequately propagate correct context instead of `context.Background` context to ensure that all contexts satisfy requirements (e.g., `modules/distribution/utils.go:130`, `utils/context.go:10`).
- The `if` statement in `modules/staking/config.go:68` does not reflect the statement of the error message. The error should be returned if `c.UpdateRepresentationsFrequency` is less than or equal to one second.
- Fix spelling mistakes in `client/config.go:12` and `x/liquidvesting/keeper/msg_server.go:113`.
- In `modules/distribution/monitoring.go:11`, check if the `CounterOpts` type value is correct: the help and name values are inconsistent.
- In `client/config.go`, consider refactoring `CelestiaClientConfigFromJunoConfig` and `MilkyWayClientConfigFromJunoConfig` functions, which perform nearly identical operations on different config sections. This duplication could be error-prone if one function is updated and the other is not.

Recommendation

We recommend applying the aforementioned recommendations.

Status: Resolved

Appendix

1. Command to compute test coverage excluding proto-generated files

To compute the test coverage accurately without including code generated by protoc-gen-gogo and protoc-gen-grpc-gateway, file extensions that end with `.pb.go` or `.pb.gw.go` need to be excluded.

The following command is executed in the `x/liquidvesting` directory to compute the test coverage:

```
go test -coverprofile=coverage.out ./... && grep -vE ".pb.go|.pb.gw.go"
coverage.out | go tool cover -func=/dev/stdin | grep total | awk '{print $3}'
```