Security Audit Report

MilkyWay Staking Updates

v1.0

May 15, 2025

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 Staking Updates.

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	`8e1b719aa1cdfcddb105ce8542dd6489d8af3d19`
Scope	The scope was restricted to the changes in the `contracts/staking` and `contracts/treasury` directories since our last audit, which was performed at commit `57dc9e714efce745c171c74c03f736de41b8b050`.
Fixes verified at commit	`bafbad53589260c804a91fe7dd7483bbadc21512` 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

The MilkyWay Staking Updates audit features changes to the `staking` and `treasury` contracts. The `staking` contract will be migrated to introduce new design configurations, and the `treasury` contract will manage tokens and allow for effective treasury balance by selling or buying tokens based on predefined rules.

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	-
Code readability and clarity	Medium	Code readability can be improved by refraining from using variables that start with an underscore (e.g., `contracts/staking/src/execute.rs:327`).
Level of documentation	Medium-High	-
Test coverage	Medium-High	`cargo-llvm-cov` reports an `82.59% (1656/2005)` line coverage for the `staking` contract and an `82.49% (311/377)` line coverage for the `treasury` contract.

Summary of Findings

No	Description	Severity	Status
1	Packet Forward Middleware vulnerability allows bypassing sender authentication during IBC hooks	Critical	Resolved
2	Migration may fail due to out-of-gas errors	Major	Resolved
3	Division by zero and incorrect redemption and purchase rates when no liquid stake token in the protocol may lead to transaction failures and loss of funds	Major	Resolved
4	Registering `ibc_callback` in `spend_funds` without `SudoMsg::IBCLifecycleComplete` implementation causes IBC hook callback failures	Major	Resolved
5	Migration will fail due to the incorrect contract version set	Major	Resolved
6	Missing validation for DAO treasury fee rate leads to potential failure of `ExecuteMsg::ReceiveRewards`	Minor	Resolved
7	Specifying duplicate packet IDs causes more funds to be distributed	Minor	Resolved
8	Redemption and purchase rates are posted with stale rates	Minor	Resolved
9	Migrating the treasury contract does not set the contract name and version	Minor	Resolved
10	Incorrect received reward amount check leads to lost treasury fees	Minor	Resolved
11	Panics in `update_oracle_msgs` block critical transactions	Minor	Resolved
12	Missing packet status validations during recovery with `selected_packets` may lead to duplicate packet transmission	Minor	Resolved
13	Lack of sanity checks on unbonding and batch periods may indefinitely lock user funds	Minor	Resolved
14	Insufficient validation of IBC denominations	Minor	Resolved
15	Lack of validation for allowed swap routes could	Minor	Resolved

	lead to loss of funds		
16	Lack of maximum slippage check would allow the account in the trader role to lose potentially all funds to an illiquid route or MEV	Minor	Acknowledged
17	Inability for a non-admin account to recover IBC transfer packets with multiple denominations may cause funds to be indefinitely stuck	Minor	Resolved
18	Incorrect access control for `ExecuteMsg::LiquidStake` allows smart contracts to perform liquid staking	Informational	Resolved
19	Optimize non-empty batch check in the `ExecuteMsg::SubmitBatch` handler	Informational	Resolved
20	Avoid the usage of magic numbers and string literals	Informational	Resolved
21	Two chains with the same prefix but connected via IBC cause confusion	Informational	Resolved
22	Resolve and remove development phase comments before production release	Informational	Resolved
23	Returning incorrect attributes may confuse off-chain indexers and smart contracts	Informational	Resolved
24	Admin can resume a non-paused contract	Informational	Resolved
25	Inconsistent address validation during migration	Informational	Resolved
26	Duplicate functions are unnecessary	Informational	Resolved
27	Misleading comments and documentation	Informational	Resolved
28	Spelling mistakes in comments	Informational	Resolved

Detailed Findings

1. Packet Forward Middleware vulnerability allows bypassing sender authentication during IBC hooks

Severity: Critical

The `receive_rewards` function in `contracts/staking/src/execute.rs:834-843` and the `receive_unstaked_tokens` function in `contracts/staking/src/execute.rs:925-939` implement sender authentications on IBC hook calls. This assumes that the message originates from a specific user on the counterparty chain.

However, if the counterparty chain has the Packet Forward Middleware (PFM) enabled, anyone can forge the sender address by exploiting the vulnerability described here. Since the Celestia chain has the PFM module enabled, this vulnerability is present in the current environment, allowing unauthorized users to bypass the sender authentication, potentially leading to unintended behavior or security risks.

While the necessity for authenticating the sender in the `receive_rewards` function is not evident, it is very critical in the `receive_unstaked_tokens` function. An attacker could front-run the coordinator and impersonate the `staker_address` by sending a `ReceiveUnstakedTokens` message immediately after the `next_batch_action_time` has passed, forcing the batch status change to `BatchStatus::Received` in line `975`. This prevents the coordinator from updating the batch, effectively blocking users from accessing their unstaked tokens. This vulnerability can lead to a denial of service for users.

Recommendation

We recommend implementing the following recommendations:

- For the `receive_rewards` function, the sender validation can be removed if authentication is not necessary. If authentication is necessary, we recommend disabling the IBC hook call and checking `info.sender` instead.
- For the `receive_unstaked_tokens` function, we recommend disabling the IBC hook call and authenticating the caller by checking `info.sender` instead. If this is not possible, we recommend considering one of the following options:
    - Updating the `resume_contract` logic to update the expected amount in pending or submitted batches when there is a validator slash, and thus verify that the amount received matches the expected amount for the batch.
    - Implement a monitoring system and a mechanism to recover from potential attacks where an attacker would have frontrun the coordinator.

Status: Resolved

2. Migration may fail due to out-of-gas errors

Severity: Major

In `contracts/staking/src/migrations/v1_1_0.rs:21-55`, the `migrate` function performs an unbounded iteration over `v1_0_0::INFLIGHT_PACKETS` and `v1_0_0::IBC_WAITING_FOR_REPLY` to migrate the entries to the new state design.

The issue is that the transaction will fail due to an out-of-gas error if there are many entries in the `v1_0_0::INFLIGHT_PACKETS` and `v1_0_0::IBC_WAITING_FOR_REPLY` states. This would cause the migration to fail, preventing the protocol from working as intended.

Recommendation

We recommend migrating the state entries by pagination within batches. This offloads the migration effort over multiple transactions, preventing out-of-gas errors.

To ensure that no new entries are added during the migration, consider locking the contract by automatically setting a `migration_pending` boolean when the migration starts. Once both the `v1_0_0::INFLIGHT_PACKETS` and `v1_0_0::IBC_WAITING_FOR_REPLY` states have no more entries, unset the `migration_pending` boolean to complete the migration.

Status: Resolved

3. Division by zero and incorrect redemption and purchase rates when no liquid stake token in the protocol may lead to transaction failures and loss of funds

Severity: Major

In `contracts/staking/src/helpers.rs:188-201`, the `get_rates` function is used to update the oracle contract as well to provide the `purchase_rate` value within the `query_state` handler response.

There are two issues with the implementation:

1. `total_native_token` is not checked to be non-zero before being used in `Decimal::from_ratio` in line `198`, which will panic if the denominator is zero. If this happens, every `ExecuteMsg` handler that performs an oracle update will fail, locking the contract up.
2. If `total_liquid_stake_token` is zero, the tuple of `(Decimal::zero(), Decimal::zero())` is returned, assigning zero to both the redemption rate and the purchase rate in the oracle and query results. The actual values should both be `Decimal::one()`, setting it to zero can mislead users viewing the query results or

cause users of the oracle potential loss of funds, e.g., collateral valued at zero and their position liquidated.

Recommendation

We recommend returning the tuple `(Decimal::one(), Decimal::one())` if `total_liquid_stake_token` or `total_native_token` is zero.

Status: Resolved

4. Registering `ibc_callback` in `spend_funds` without `SudoMsg::IBCLifecycleComplete` implementation causes IBC hook callback failures

Severity: Major

In `contracts/treasury/src/execute.rs:128`, the `ExecuteMsg::SpendFunds` handler issues an IBC Transfer if a `channel_id` is specified along with a receiver address that has the "`celestia`" prefix.

The memo field is set to `{ "ibc_callback": "<treasury_contract_address>" }`, which requests the `x/ibc-hooks` module to issue a `SudoMsg::IBCLifecycleComplete` callback to the specified contract once the IBC transfer packet is acknowledged.

However, the treasury contract does not implement this `sudo` entry point, leading to a failure in the IBC hook callback. This call from the IBC hooks can be observed in the following references:

- https://github.com/osmosis-labs/osmosis/blob/v29.0.0/x/ibc-hooks/wasm_hook.go#L336
- https://github.com/osmosis-labs/osmosis/blob/v29.0.0/x/ibc-hooks/wasm_hook.go#L371

Consequently, this failure can disrupt the intended IBC operations and lead to unhandled callbacks, potentially affecting the contract's functionality.

Recommendation

We recommend addressing the handling of IBC callbacks as follows:

- If handling IBC callbacks is not intended, remove their usage.
- Otherwise, implement the `sudo` entry point to properly handle the `SudoMsg::IbcLifecycleComplete` message. This will ensure the contract can process IBC callbacks correctly and maintain expected functionality.

Status: Resolved

5. Migration will fail due to the incorrect contract version set

Severity: Major

To upgrade from the previously deployed contract version `v0.4.20`, the contract must first be upgraded to `v1.0.0` and then to `v1.1.0` by issuing the `MigrateMsg::V0_4_20ToV1_0_0` and `MigrateMsg::V1_0_0ToV1_1_0` messages, as seen in `contracts/staking/src/contract.rs:284-297`.

However, the migration will fail to migrate the contract from `v1.0.0` to `v1.1.0`. This is because after the `MigrateMsg::V0_4_20ToV1_0_0` migration has been completed, `set_contract_version` in line `301` will be called to incorrectly set the contract version to `v1.1.0`. When the `MigrateMsg::V1_0_0ToV1_1_0` message is called, the transaction will error in line `277` because the contract tries to migrate to the same version.

Recommendation

We recommend setting the contract version to `v1.0.0` after completing the `MigrateMsg::V0_4_20ToV1_0_0` migration. Additionally, we recommend pausing the contract and resuming it once the migration process has been completed.

Status: Resolved

6. Missing validation for DAO treasury fee rate leads to potential failure of `ExecuteMsg::ReceiveRewards`

Severity: Minor

In `contracts/staking/src/types.rs:25-33`, the `UnsafeProtocolFeeConfig::validate` function does not implement validation checks for the `UnsafeProtocolFeeConfig::dao_treasury_fee` value. This allows values exceeding `100_000` to be set, implying a fee rate greater than 100%.

If that occurs, in `contracts/staking/src/execute.rs:858-870`, the `ExecuteMsg::ReceiveRewards` message handler will always fail with the misleading `ContractError::ReceiveRewardsTooSmall` error instead of an overflow error.

We classify this issue as minor severity because it can only be caused by the admin, who is a privileged user.

Recommendation

We recommend introducing a constant (for example, `MAX_TREASURY_FEE`) that is at most `100_000` and raising an error in the `validate` function if `dao_treasury_fee` exceeds that limit.

Status: Resolved

7. Specifying duplicate packet IDs causes more funds to be distributed

Severity: Minor

In `contracts/staking/src/execute.rs:701-713`, the `recover` function iterates over the `selected_packets` parameter to refund the IBC packets to the receiver. However, if there are duplicate packet IDs, the computed amount will be more than intended (see line `760`), causing excess funds to be sent.

We classify this issue as minor severity because it can only be caused by the admin, who is a privileged user.

Recommendation

We recommend deduping the `selected_packets` parameter to avoid duplicates.

Status: Resolved

8. Redemption and purchase rates are posted with stale rates

Severity: Minor

In `contracts/staking/src/execute.rs:240`, the `execute_liquid_stake` function calls `update_oracle_msgs` to post rates to the MilkyWay oracle contract. This is performed by computing the rates retrieved from the `STATE` storage, as seen in `contracts/staking/src/helpers.rs:189`.

However, the `update_oracle_msgs` function is called before the `STATE` persists in storage (see `contracts/staking/src/execute.rs:245`). This means the rates will be computed using the outdated `total_native_token` and `total_liquid_stake_token` fields, causing the oracle contract to receive incorrect posted rates.

Recommendation

We recommend calling the `update_oracle_msgs` function after the `STATE` is persisted in storage.

Status: Resolved

9. Migrating the treasury contract does not set the contract name and version

Severity: Minor

In `contracts/treasury/src/contract.rs:112`, the `migrate` function does not update the contract name and version with the `set_contract_version` function. This causes the contract to use the old name and version, which is incorrect.

Recommendation

We recommend calling the `set_contract_version` function.

Status: Resolved

10. Incorrect received reward amount check leads to lost treasury fees

Severity: Minor

In `contracts/staking/src/execute.rs:864`, the `ExecuteMsg::ReceiveRewards` handler validates whether the rewards are sufficient to incur fees via the `amount.checked_sub(fee).is_err()` condition and return the `ContractError::ReceiveRewardsTooSmall` error.

This validation is incorrect for the intended error, as the condition is only evaluated to `true` if the calculated fee amount is larger than the reward, which indicates a subtraction overflow that only occurs if the fee rate is 100% or greater.

The received amount is too small for the fee rate to be applied when the `fee` calculation in lines `859-862` evaluates to zero. For example, when `ProtocolFeeConfig::dao_treasury_fee` is `3_000` (3%) and the reward amount is `33`, `(3,000 * 33) / 100_000` evaluates to zero due to integer division, leading to no fee being deducted and a loss of revenue.

We classify this issue as minor because the `ExecuteMsg::ReceiveRewards` message that contains the actual rewards can only be issued by a trusted party, which is the `NativeChainConfig::reward_collector_address`. This reduces the likelihood of this flaw being exploited (i.e., `ExecuteMsg::ReceiveRewards` messages sent so frequently that no fees are ever received).

Recommendation

We recommend returning the `ContractError::ReceiveRewardsTooSmall` error if the calculated `fee` amount is equal to zero, thereby ensuring that the fee deduction is not unintentionally bypassed.

Status: Resolved

11. Panics in `update_oracle_msgs` block critical transactions

Severity: Minor

In `contracts/staking/src/execute.rs:109`, the `oracle_address` from the `protocol_chain_config` is unwrapped. Since this is an `Option<Addr>`, if the `oracle_address` is not defined, the function will panic, blocking the transaction without proper error handling. This issue affects the `update_oracle_msgs` function, which is part of the happy path for several critical operations:

- `execute_liquid_stake`
- `execute_submit_batch`
- `execute_withdraw`
- `receive_rewards`
- `resume_contract`

If `protocol_chain_config.oracle_address` is set to `None`, the above calls to the staking contract will fail.

We classify this issue as minor severity because only the admin can configure an empty `oracle_address`, who is a privileged user.

Recommendation

We recommend addressing the handling of the `oracle_address` as follows:

- If an `oracle_address` is mandatory, it should not be an `Option<Addr>`, but an `Addr`. The `UnsafeProtocolChainConfig.validate` function should correctly convert the `String` to an `Addr`.
- If an `oracle_address` is not mandatory, the code should reflect this and handle the absence of a redemption or purchase rate oracle address gracefully, ensuring that the function does not fail.

Status: Resolved

12. Missing packet status validations during recovery with `selected_packets` may lead to duplicate packet transmission

Severity: Minor

In `contracts/staking/src/execute.rs:707-710`, the `recover` function does not validate the packet statuses passed from the `selected_packets` parameter. This oversight allows the admin to re-emit IBC packets that are still in the `PacketLifecycleStatus::Sent` status, potentially leading to duplicate packet transmissions.

We classify this issue as minor severity because it can only be caused by the admin, who is a privileged user.

Recommendation

We recommend enhancing the `recover` function to validate the packet status when `selected_packets` are specified. Specifically, ensure that only packets with `PacketLifecycleStatus::AckFailure` or `PacketLifecycleStatus::TimedOut` statuses can be re-emitted. This will prevent the re-emission of packets that are still in transit or have not yet failed.

Status: Resolved

13. Lack of sanity checks on unbonding and batch periods may indefinitely lock user funds

Severity: Minor

The `UnsafeNativeChainConfig::validate_function` in `contracts/staking/src/types.rs:69-72` does not validate the `unbonding_period` and `batch_period` fields, which are configured in `contracts/staking/src/contract.rs:75` and `contracts/staking/src/execute.rs:815-817`.

As no upper bound is enforced, a very large value (e.g., 100 years) can be configured for either period, causing user funds to remain locked for that duration.

We classify this issue as minor severity because it can only be caused by the admin, who is a privileged user.

Recommendation

We recommend introducing a `MAX_UNBONDING_PERIOD` constant with a reasonable limit and validating it during the contract instantiation and `update_config` phases, such that `unbonding_period <= MAX_UNBONDING_PERIOD` and `batch_period <=`

`unbonding_period`. Implementing these sanity checks ensures that users do not have their funds inadvertently locked.

Status: Resolved

14. Insufficient validation of IBC denominations

Severity: Minor

In `contracts/staking/src/helpers.rs:217-224`, the `validate_ibc_denom` function performs insufficient validation on IBC denominations that could lead to unintended protocol behavior.

The current implementation only checks that the denomination starts with "`ibc/`" and that the remainder is 64 characters long. This validation does not ensure that the IBC denomination corresponds to the expected token from the native chain as specified in `NativeChainConfig::token_denom`.

Consequently, two issues can occur:

1. If the IBC denomination is completely invalid (e.g., contains invalid characters), the transaction will fail when the `execute_liquid_stake` handler attempts to issue an IBC transfer message, which occurs when the contract sends the received IBC assets to the `NativeChainConfig::staker_address` for staking.
2. More critically, if the IBC denomination is valid but corresponds to a token from the "native" chain that is not the expected `NativeChainConfig::token_denom`, the transaction will succeed. This would result in the protocol incorrectly minting Liquid Staked Tokens (LSTs) to the sender of an incorrect token.

Recommendation

We recommend enhancing the validation to ensure that the IBC denomination corresponds specifically to the expected token from the native chain. The validation should verify that the substring after "`ibc/`" equals:

```
sha256("transfer/{source_channel}/{native_staking_denom}")
```

where:

- `source_channel` is `ProtocolChainConfig::ibc_channel_id`.
- `native_staking_denom` is `NativeChainConfig::token_denom`.

This approach aligns with the IBC specification for denomination traces as implemented in the Cosmos IBC-Go module.

Status: Resolved

15. Lack of validation for allowed swap routes could lead to loss of funds

Severity: Minor

In `contracts/treasury/src/contract.rs:52`, the `instantiate` function sets `Config::allowed_swap_routes` without validation. Similarly, in `contracts/treasury/src/execute.rs:259-263`, the `ExecuteMsg::UpdateConfig` handler does not perform any validation of updated swap routes.

If a non-existent route is mistakenly whitelisted, an attacker could create the required pools with minimal liquidity, influence the trader account to swap through that route, then withdraw liquidity, leading to unfavorable rates or fund loss.

We classify this as minor severity as only the contract deployer or admin can whitelist swap routes, who is a privileged user.

Recommendation

We recommend validating the existence of each route before accepting it by making an `EstimateSwapExactAmountInRequest` query and returning an informative error if the query fails.

Status: Resolved

16. Lack of maximum slippage check would allow the account in the trader role to lose potentially all funds to an illiquid route or MEV

Severity: Minor

The `execute_swap_exact_amount_in` function in `contracts/treasury/src/execute.rs:150-185` and the `execute_swap_exact_amount_out` function in lines `196-231` allow the trader to manually select the amount of min tokens out / tokens in max for a swap, as well as the routes used.

A malicious trader could potentially select routes and values for `token_out_min_amount` or `token_in_max_amount` in such a way that funds from the treasury would get lost, and for a potential personal gain.

We classify this issue as minor severity because it can only be caused by the trader, who is a privileged user nominated by the admin.

Recommendation

We recommend setting up monitoring of the trades initiated by the trader so that any malicious attempt can be detected and the privileges of the trader can be revoked if necessary.

Status: Acknowledged

The client states that they will implement the monitoring in the off-chain program.

17. Inability for a non-admin account to recover IBC transfer packets with multiple denominations may cause funds to be indefinitely stuck

Severity: Minor

In `contracts/staking/src/execute.rs:735-746`, the `ExecuteMsg::RecoverPendingIbcTransfers` handler verifies that all packets being recovered have the same denomination. The function returns an error if multiple denominations are detected among the packets.

When a non-admin user has pending IBC transfers with different denominations that have failed or timed out, they cannot retry these transfers on their own. This forces the user to rely on admin intervention to manually select packets of the same denomination. This means that users' funds can remain indefinitely stuck if the admin is unresponsive or uncooperative.

Furthermore, an attacker could potentially exploit this design by using the `mint_to` parameter of the `ExecuteMsg::LiquidStake` message to cause transfers of multiple denominations (both the LST and staking asset) to be sent to the `NativeChainConfig::staker_address` for staking. If both of those transfers fail or timeout, only the admin would be able to retry the transfers by carefully selecting packets of the same denomination. The goal of this attack would be to 'grief' the admin entity.

We classify this issue as minor since, in the current design, only the staker address can receive IBC transfers from the contract in different denominations. Additionally, it is unlikely that an attacker could deterministically cause IBC transfers of both the LST and staking assets to fail at the staker address.

Recommendation

We recommend modifying the function to group packets by denomination, then merging each group into a new IBC transfer.

Status: Resolved

18. Incorrect access control for `ExecuteMsg::LiquidStake` allows minting to non-normal accounts

Severity: Informational

The `execute_liquid_stake` function in `contracts/staking/src/execute.rs:144-151` intends that only normal accounts (i.e., not smart contracts or ICAs) may be the recipient of the LSTs minted in the `ExecuteMsg::LiquidStake` handler. However, the current implementation does not enforce this restriction effectively.

Specifically, any user can bypass the restriction by passing an address in the `mint_to` parameter, which is not checked to be a "normal account".

After confirmation with the client, we downgraded this issue to informational severity. Instead, the client has decided to allow any account to execute the `ExecuteMsg::LiquidStake` message without any restrictions on the account type specified in the `mint_to` field.

Recommendation

Since the client has decided to allow any account to execute the `ExecuteMsg::LiquidStake` message without any restrictions, we recommend removing the validation in `contracts/staking/src/execute.rs:144-151` to avoid any confusion when reading the code.

Status: Resolved

19. Optimize non-empty batch check in the `ExecuteMsg::SubmitBatch` handler

Severity: Informational

In `contracts/staking/src/execute.rs:373-381`, the code unnecessarily reads from storage to check if there are any unstake requests for a batch when this information is already available in memory.

The current implementation performs a storage read operation on the `Map` returned by the `unstake_requests` function to determine if the pending batch is empty or not. This storage read is unnecessary because the `batch` object, which was previously loaded from the `BATCHES` storage in line `355`, already contains a count of unstake requests via the `batch.unstake_requests_count` field.

Consequently, performing unnecessary storage reads increases gas costs and reduces the efficiency of the contract.

Recommendation

We recommend returning the `ContractError::BatchEmpty` error if the `batch.unstake_requests_count == 0` and avoiding the storage read.

Status: Resolved

20. Avoid the usage of magic numbers and string literals

Severity: Informational

The following are instances where numbers and string literals are used directly in the code:

- In `contracts/staking/src/execute.rs:862`, the number literal `100_000u128` is used to represent the fee rate denominator.
- In `contracts/staking/src/execute.rs:146`, the number `39` is used to represent the length of a native user address minus the prefix.
- In `contracts/treasury/src/execute.rs:108`, `105`, and `119`, the chain prefixes `osmo` and `celestia`, as well as the port `transfer` are hardcoded.

The use of magic numbers and string literals reduces maintainability going forward, as new readers of the code have to work out the meaning and intent behind the usage.

Recommendation

We recommend creating named constants for all the instances of magic values outlined above. For the chain prefixes, we recommend using configurations like `NativeChainConfig` and `ProtocolChainConfig` to be consistent with the staking contract.

Status: Resolved

21. Two chains with the same prefix but connected via IBC cause confusion

Severity: Informational

In `contracts/staking/src/execute.rs:171`, there is a check for when the native chain and the protocol chain have the same prefix (e.g., "`osmo`").

Moreover, in `contracts/staking/src/execute.rs:172-175`, the code derives two variables: `mint_to_is_protocol` and `mint_to_is_native`, with the latter being currently unused, to determine whether the minted tokens should be later sent via IBC or not.

As seen in the protocol chain config validation and the `stake_sub_message` created in line `230`, the code currently enforces an IBC connection between the native chain and the protocol chain.

This setup suggests that the protocol and native chains are distinct, even if they share the same prefix. This logic could lead to confusion, as it implies the possibility of IBC communication between two chains with identical prefixes, which is unusual.

If the intention is to handle the scenario where the native and protocol chains are the same (e.g., Osmosis), the current implementation would not manage this correctly, as it would attempt to perform IBC from Osmosis to Osmosis.

Recommendation

We recommend clarifying the intended use case for handling chains with the same prefix.

- If the goal is to support the same chain for both native and protocol chains, update the code to handle this scenario appropriately, removing the IBC message and IBC mandatory settings in the protocol chain configuration.
- If the intention is to support two different chains with the same prefix, add comments to clarify this logic and ensure the implementation aligns with this requirement.
- If there is a requirement to always have an IBC connection between the native and protocol chains, and those chains should never have the same prefix, consider adding the validation rule to ensure `NativeChainConfig::account_address_prefix != ProtocolChain Config::account_address_prefix`.

Following the clarification, we recommend updating the code to determine whether the minted tokens should be sent via IBC or on the protocol chain accordingly.

During the fixes review process, the client confirmed that this is to handle the case with two different chains connected via IBC but with the same prefix.

Status: Resolved

22. Resolve and remove development phase comments before production release

Severity: Informational

The following are instances where there are development-related comments such as “TODO”, “DEPR”, and other such comments:

- In `contracts/staking/src/contract.rs:235`, the comment “DEPR” appears to imply that the `QueryMsg::AllUnstakeRequests` and `QueryMsg::AllUnstakeRequestsV2` messages are for temporary development

purposes only. Also, in line `242`, the comment "dev only, depr" appears to imply the same for `QueryMsg::IbcQueue` and `QueryMsg::IbcReplyQueue` messages.

- In `contracts/staking/src/contract.rs:54`, there is a comment in the `instantiate` method, reading "*TODO: determine if info.sender is the admin or if we want to pass in with msg*". This decision is crucial and should be resolved before the contract is deployed to production.

Recommendation

We recommend resolving and removing the comments along with any associated deprecated code referenced above prior to the production release.

Status: Resolved

23. Returning incorrect attributes may confuse off-chain indexers and smart contracts

Severity: Informational

In `contracts/staking/src/ibc.rs:78`, the `receive_timeout` function incorrectly returns an attribute action with the value `receive_ack`.

Similarly, in `contracts/treasury/src/execute.rs:234`, the `action` attribute returned for the `execute_swap_exact_amount_out` function is `swap_exact_amount_in` instead of `swap_exact_amount_out`.

Additionally, in `contracts/treasury/src/contract.rs:58`, the `instantiate` function returns the `owner` attribute as `info.sender`, while the admin could be different (as configured in `msg.admin`).

Consequently, these discrepancies may confuse off-chain indexers or smart contracts calling the contract, leading to incorrect handling of events.

Recommendation

We recommend the following updates to ensure clarity and consistency in event attributes:

- For the `receive_timeout` function, update the attribute to `receive_timeout` instead of `receive_ack`. Consider adding the `channel` as an attribute, similar to how it is handled in the `receive_ack` function.
- For the `execute_swap_exact_amount_out` function, ensure the `action` attribute correctly reflects `swap_exact_amount_out`.

- For the `instantiate` function, update the returned `owner` attribute to `admin` instead of `info.sender`.

Status: Resolved

24. Admin can resume a non-paused contract

Severity: Informational

In `contracts/staking/src/execute.rs:1013`, the `resume_contract` function allows the admin to resume the contract, even if it is not currently paused. This can be problematic because a paused contract is typically monitored, and resuming it should only be allowed if it is indeed paused.

The `resume_contract` function also allows the admin to modify critical information, such as `total_native_token`, `total_liquid_stake_token`, and `total_reward_amount`.

Although the admin can also pause the contract via the circuit breaker, adding a validation to ensure the contract is paused before resuming can serve as a precaution against potential admin abuse.

Recommendation

We recommend implementing a validation to verify that the contract is paused before processing the `resume_contract` function. This will prevent the admin from resuming the contract when it is not paused, adding an extra layer of security and ensuring proper contract management.

We also recommend implementing monitoring of value changes in the `resume_contract` function so that a malicious admin cannot pause and resume instantly in a new state without triggering alerts.

Status: Resolved

25. Inconsistent address validation during migration

Severity: Informational

In `contracts/staking/src/migrations/v1_0_0.rs:36-61`, the `migrate` function validates most addresses against their respective prefixes but omits validation for `monitors` addresses against the `protocol_account_address_prefix`.

Since the `oracle_address` is properly validated against the same prefix that would be used for monitors, and these prefixes should be identical, this presents minimal risk.

Recommendation

We recommend adding validation of the `monitors` addresses for consistency.

Status: Resolved

26. Duplicate functions are unnecessary

Severity: Informational

The `validate_address` function in `contracts/treasury/src/helpers.rs:3` is identical to `contracts/staking/src/helpers.rs:43`. This is unnecessary and only one should be used across the codebase.

Recommendation

We recommend removing one of the `validate_address` functions.

Status: Resolved

27. Misleading comments and documentation

Severity: Informational

The following are instances where there are misleading comments or documentation that do not reflect actual behaviour as per the code:

1. In `contracts/staking/README.md:120-122`, the documentation incorrectly lists only `paginated` as a parameter for the `RecoverPendingIbcTransfers` message. In reality, the message also includes the parameters `selected_packets` and `receiver`.

2. In `contracts/staking/src/execute.rs:344`, there is a comment that mentions that `execute_submit_batch` is "*Called automatically during liquidUnstake*", which is incorrect, as neither `ExecuteMsg::SubmitBatch` is issued, nor `execute_submit_batch` is called in `execute_liquid_stake`.

3. In `contracts/staking/src/helpers.rs:208`, the error message does not align with the actual condition.

4. In `contracts/staking/src/execute.rs:697`, a comment reads "*Fallback to staker address in case the sender was None*". The sender can never be `None`, and we believe the developer meant the "in case the receiver was None".

Additionally, in `contracts/staking/README.md`, there are stale definitions of `InstantiateMsg`, `ExecuteMsg`, and `QueryMsg`, which do not match the actual implementations in the code:

- The `InstantiateMsg` structure in the README includes many fields that do not exist in the actual code.
- The `ExecuteMsg` enum in the README has outdated parameters and missing variants. For example, the README shows `SubmitBatch` with a `batch_id` parameter, but the actual implementation does not have this parameter. The README also does not document all available message types that exist in the actual code.
- The `QueryMsg` enum in the README is incomplete, and several query types that are present in the implementation are missing, such as `BatchesByIds`, `PendingBatch`, and `UnstakeRequests`.
- The query response JSON examples do not match the actual response types if serialized. For instance, the `BatchResponse` type structure in the code contains different fields than what's shown in the README examples.

Consequently, these discrepancies can mislead developers or users, leading to improper usage and potential errors when integrating or interacting with the smart contract.

Recommendation

We recommend updating the documentation to accurately reflect the code or updating the code to accurately reflect the documentation.

1. For the `RecoverPendingIbcTransfers` message, ensure that `selected_packets` and `receiver` are included in the documentation and described appropriately to provide clear guidance on their usage.
2. Consider removing the comment or updating `execute_liquid_stake` to actually call `execute_submit_batch` if necessary.
3. Consider updating the message to be "*denom length is less than or equal to 3*".
4. Consider updating the comment to "*in case the receiver was None*".
5. Ensure all type definitions in `contracts/staking/README.md` match those in the actual code, as well as the example JSON responses.

Status: Resolved

28. Spelling mistakes in comments

Severity: Informational

The comment in `contracts/staking/src/migrations/v1_1_0.rs:38` should be "*Migrate the ibc messages waiting for reply*", not "*replay*".

The markdown subtitle in `contracts/treasury/README.md:48` should be "*Spend funds*", not "*Sped*".

Recommendation

We recommend fixing the spelling mistakes described above.

Status: Resolved