Security Audit Report

Escher

v1.0

April 9, 2025

Table of Contents

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 GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Fractal Abstraction Ltd to perform a security audit of Audit of the Escher Finance CosmWasm smart contracts.

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/Escher-finance/cw-liquid-staking
Commit	`261306403a6d24ef2718a40805c0cc22d42ba7f9`
Scope	The CosmWasm contracts in `contracts/liquidstaking` and `contracts/reward` were in scope.
Fixes verified at commit	`fd534c51109f528776368b0ef59bed6ae23ae71e` 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

Escher.Finance is a liquid staking protocol allowing users to stake tokens in return for a liquid staking token, either by directly interacting with the contract or cross-chain via Union.

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**	-
Level of documentation	**Medium-High**	The client provided detailed documentation and a code walkthrough during the kick-off call of the review.
Test coverage	**Low**	`cargo-llvm-cov` reports a `41.66% (1009/2422)` test line coverage for the liquidstaking contract and a `12.40% (30/242)` test line coverage for the reward contract. An end-to-end (E2E) test suite is available, but we could not determine the resulting test coverage.

Summary of Findings

No	Description	Severity	Status
1	Unbonding requests can lead to a Denial of Service (DoS) attack	Critical	Resolved
2	Withdrawing unbonded staking tokens on behalf of a user uses the contract's staking token balance, which can be manipulated to cause the `Ucs03ExecuteMsg::Transfer` message to fail	Critical	Resolved
3	Unauthorized migration of the reward contract allows theft of staking rewards or causes a Denial of Service of the liquid staking contract	Critical	Resolved
4	Slashing penalties are not enforced when processing unbonded funds	Critical	Resolved
5	Attackers can front-run `Unbond` messages to steal user funds	Major	Resolved
6	The first depositor can trigger a share inflation attack to steal user funds	Major	Resolved
7	`OnZkgm` message interface mismatch with Union	Major	Resolved
8	Delegation amount can be zero in specific scenarios, leading to a failed Cosmos SDK `Delegate` message, preventing delegations and redelegations	Major	Resolved
9	Rounding error carryover in the undelegation process can lead to failed undelegation messages, preventing liquid staking tokens from unbonding	Major	Resolved
10	Development-only functions pose a risk to the overall protocol	Major	Resolved
11	Liquid staking tokens are not sent to the user when the contract owner stakes on behalf of a user	Major	Resolved
12	Compounding staking rewards requires a large number of messages, which might exceed the block gas limit and prevent reward processing	Major	Resolved
13	Lack of a strict withdrawal queue of unbonded staking tokens	Major	Resolved
14	Incorrect exchange rate update when total LST	Major	Resolved

	token supply is zero		
15	Redelegation amount can be zero, preventing migration of staking tokens to a new validator set	Major	Resolved
16	Incorrect message interface used for reward contract	Major	Resolved
17	Unrestricted unbonding records can be queried if the `min` pagination parameter is unspecified	Minor	Resolved
18	Unrestricted querying of unreleased unbond records for a specific staker address	Minor	Resolved
19	Inability to query all unbonded records regardless of staker address or release status	Minor	Acknowledged
20	Undelegate amounts can potentially be off-by-one in the unbonding process	Minor	Resolved
21	Unhandled delegation query errors can lead to an incorrect liquid staking token exchange rate	Minor	Resolved
22	Updating the validator set does not account for rounding errors in delegation amounts	Minor	Resolved
23	Missing validations when configuring validators and quote denoms	Minor	Resolved
24	Missing slippage control mechanisms for `Bond` and `Unbond` messages	Minor	Resolved
25	Potential incorrect computing due to ceiling usage	Minor	Resolved
26	Ineffective ownership update in rewards contract	Minor	Resolved
27	Overflow checks are not enabled in production, and the release profile is missing	Minor	Resolved
28	Double sender validations in `split_reward`	Informational	Resolved
29	Use `deps.api.debug` for logging purposes	Informational	Resolved
30	Handle the optional amount parameter in the unbond function to use the current liquid staking token balance if `None` is passed	Informational	Resolved
31	Unneeded `set_contract_version` call	Informational	Resolved
32	Code quality improvements	Informational	Resolved

Detailed Findings

1. Unbonding requests can lead to a Denial of Service (DoS) attack

Severity: Critical

In `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:564-668`, the `process_unbond` function handles an unbond request by burning the liquid staking CW-20 tokens, undelegating the corresponding amount of staking tokens from all validators and storing the unbonding record for later withdrawal.

However, as each LST unbonding request results in multiple validator undelegations, attempting to undelegate from all configured validators based on their weights, the maximum amount of pending undelegations can be reached quickly, even purposefully by an attacker. This leads to a Denial of Service (DoS), where the contract cannot process further unbonding requests.

Furthermore, validator undelegation does not consider pending redelegations caused by previously migrating stake between validators, which effectively locks a portion of the delegated amount for a specific time to accommodate for eventual slashing events.

Recommendation

We recommend implementing a batching mechanism consolidating multiple user unbonding requests into fewer validator undelegations. This would help manage the undelegation entries more efficiently without reaching the maximum limit of pending undelegations.

Additionally, consider ongoing redelegations when determining the available validator delegation amounts.

Status: Resolved

2. Withdrawing unbonded staking tokens on behalf of a user uses the contract's staking token balance, which can be manipulated to cause the `Ucs03ExecuteMsg::Transfer` message to fail

Severity: Critical

In `contracts/liquidstaking/liquidstaking/src/execute.rs:785-788`, when withdrawing the unbonded staking tokens, the `process_unbonding` function sets `funds.amount` to the contract's staking token balance, which can be larger than the expected `undelegate_amount` amount to be transferred. In this case, the Union `Ucs03ExecuteMsg::Transfer` message will fail when checking the token amounts during escrowing, preventing the withdrawal.

This can be exploited by purposefully sending additional staking tokens to the contract to increase the contract's balance. Moreover, this will also occur naturally when multiple matured unbonding records are processed by the chain in the same block, and staking tokens are sent to the contract, exceeding a single unbonding record's `undelegate_amount`.

Recommendation

We recommend modifying the `process_unbonding` function to use `undelegate_amount` instead of `transfer_amount`.

Status: Resolved

3. Unauthorized migration of the reward contract allows theft of staking rewards or causes a Denial of Service of the liquid staking contract

Severity: Critical

In `contracts/liquidstaking/liquidstaking/src/execute.rs:1008-1025`, the `migrate_reward` function allows any user to migrate the reward contract to an arbitrary code ID. This is problematic because it allows an attacker to upload a malicious code ID and migrate the reward contract to it.

Consequently, the attacker could execute arbitrary code, potentially withdrawing staking rewards to steal funds or breaking the integration with the liquid staking contract so that it no longer works.

Recommendation

We recommend implementing strict authorization checks to ensure that only the contract owner can migrate the reward contract.

Status: Resolved

4. Slashing penalties are not enforced when processing unbonded funds

Severity: Critical

In `contracts/liquidstaking/liquidstaking/src/execute.rs:751`, the `process_unbonding` function allows the contract owner to release unbonded funds back to the staker. During the unbonding process, validators may get slashed due to misbehaviors, resulting in the final returned amount being less than intended.

However, slashing mechanisms are not enforced throughout the protocol. If unbonding validators are slashed, the returned amount (unbond_rec.undelegate_amount) should decrease based on the slashed percentage.

Consequently, stakers will receive unbonded funds at the expense of other users, causing bad debt to accrue and a loss of funds scenario.

Recommendation

We recommend computing the undelegated amount after applying the slashing penalty.

Status: Resolved

5. Attackers can front-run Unbond messages to steal user funds

Severity: Major

In contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:632-633, the process_unbond function allows users to redeem their CW20 shares tokens for bonded funds according to the exchange rate.

However, the CW20 token shares are not validated to be provided by the caller. The current approach dispatches a Cw20ExecuteMsg::Burn message to burn the funds directly from the contract balance. This is problematic because if the user initiates two transactions, the first one to transfer the CW20 tokens to the contract and another to call the ExecuteMsg::Unbond message, an attacker can front-run the second transaction to call ExecuteMsg::Unbond first and steal the user's funds.

Recommendation

We recommend modifying the implementation to utilize the Cw20ReceiveMsg entry point to ensure the CW20 tokens are sent by the user in the ExecuteMsg::Unbond message.

Status: Resolved

6. The first depositor can trigger a share inflation attack to steal user funds

Severity: Major

The liquidstaking contract does not prevent the first depositor from being front-run to effectively get fewer shares than planned, from which the attacker will profit.

In contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:515, the number of shares a user receives is computed by dividing the given amount by the exchange rate. The exchange rate is computed by dividing the total_bond_amount variable by the total supply (see line 512). The total_bond_amount variable is computed by summing the total delegated amount, pending rewards, and rewards contract balance, as seen in line 51.

The issue is that if a malicious depositor makes a large enough “donation” to the rewards contract at the right time (inflating the total_bond_amount value), the next depositor will not receive their expected share amount.

Below describes an example exploit scenario that could be followed by an attacker "Mallory", to take advantage of a victim "Alice":

1. Upon identifying that Alice is trying to make the first deposit of 20_000 tokens into the liquidstaking contract, Mallory front-runs her transaction with two calls:
    a. Deposit 1 token to obtain one unit of shares, setting the total_bond_amount and total_supply to one.
    b. Donate 10_000 tokens to the rewards contract to inflate the total_bond_amount value. Note that the total_supply remains as 1.
2. Alice’s transaction gets executed, containing the ExecuteMsg::Bond message, and expects to receive 20_000 shares. When the exchange rate is updated, the donation in the rewards contract is included, making it 10_001. In the calculate_staking_token_from_rate function, the mint amount will be computed as 20_000 / 10_001, causing it to be 1.99980002 and eventually truncated into 1 share. Alice will receive 1 share instead of 20_000 shares.
3. Mallory unbonds their shares to receive half of the total bonded amount, which is 15_000 computed by the calculate_native_token_from_staking_token function as ((30_001 /2) * 1).
4. Mallory receives 25% profit from their deposited tokens at the expense of Alice’s funds.

Note that this issue is only exploitable at the beginning of the contract’s lifecycle and only affects the first user making the deposit. However, as the potential loss of funds can be substantial, we classify it as major.

Recommendation

We recommend implementing virtual shares and assets with sufficient decimal offset. Alternatively, consider minting a number of dead shares to the null address if the total supply is zero.

Status: Resolved

7. `OnZkgm` message interface mismatch with Union

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/msg.rs:99-103`, the `OnZkgm` message does not align with the interface required by Union. Specifically, the `channel_id` field is missing while additional fields `path`, `source_channel_id`, and `destination_channel_id` are present. As a result, the contract is unable to receive and execute the `OnZkgm` message from Union.

This discrepancy may have happened because Union has updated their message interface, which was previously matching as per this commit.

Recommendation

We recommend updating the `OnZkgm` message to match the latest interface.

Status: Resolved

8. Delegation amount can be zero in specific scenarios, leading to a failed Cosmos SDK `Delegate` message, preventing delegations and redelegations

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:357-369`, the `get_delegate_to_validator_msgs` function attempts to delegate `delegate_amount` staking tokens to a specific validator.

However, under certain conditions, `delegate_amount` can be zero. Specifically, this occurs when the provided `delegate_amount` is very small, so when calculating the validator's share with the `calculate_delegated_amount` function, it is rounded down to zero. This situation is particularly likely when redelegating staking rewards, which are typically smaller token amounts.

If `delegate_amount` is zero, the `Delegate` Cosmos SDK message will error, as it requires a positive amount. Note that this is not checked by `wasmd` when translating the cosmwasm message to the corresponding Cosmos SDK message.

Recommendation

We recommend skipping the delegation to the specific validator if `delegate_amount` is zero.

Status: Resolved

9. Rounding error carryover in the undelegation process can lead to failed undelegation messages, preventing liquid staking tokens from unbonding

Severity: Major

In `liquidstaking/liquidstaking/src/utils/delegation.rs:146-189`, the `get_undelegate_from_validator_msgs` function builds a list of undelegation messages for each validator based on their individual weights, given an undelegation amount `undelegate_amount`, as part of the LST unbonding process.

When calculating the undelegation amounts for each validator, the function rounds down the calculated amounts to the nearest whole number with `to_uint_floor` in line `169`. Consequently, this rounding results in a small discrepancy between the total intended undelegation amount and the sum of the validator undelegate amounts. The code attempts to correct this by adding the remaining tokens to the last validator's undelegation amount.

However, if the last validator's actual delegation is less than this adjusted amount, the undelegate message fails because it attempts to undelegate more tokens than are available, causing the transaction to revert.

Recommendation

We recommend capping the last validator's undelegation amount to their actual delegation amount to prevent the undelegation message from failing.

Status: Resolved

10. Development-only functions pose a risk to the overall protocol

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/execute.rs`, the `transfer_to_owner`, `move_to_reward`, `reset`, `transfer`, and `transfer_reward` functions are supposedly intended for development purposes only. These owner-callable functions, i.e., the corresponding messages, allow for actions such as transferring all native balances to the owner or reward contract, resetting the contract state, and arbitrarily transferring tokens via a specified contract.

The presence of these functions poses significant risks for production deployments due to potential or accidental misuse and abuse, undermining the integrity and trustworthiness of the protocol.

Recommendation

We recommend removing the `transfer_to_owner`, `move_to_reward`, `reset`, `transfer`, and `transfer_reward` functions and their corresponding messages from the contract.

Status: Resolved

11. Liquid staking tokens are not sent to the user when the contract owner stakes on behalf of a user

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/execute.rs:57-76`, the `bond` function allows the contract owner to stake on behalf of a user. However, the current implementation results in the owner receiving the minted Liquid Staking Tokens (LST) instead of the intended user. This occurs because both `sender` and `the_staker` are set to the owner, leading to the owner being the recipient of the minted tokens without any transfer to the user.

Recommendation

We recommend modifying the `bond` function to include an option for specifying a recipient address.

Status: Resolved

12. Compounding staking rewards requires a large number of messages, which might exceed the block gas limit and prevent reward processing

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/execute.rs:489-550`, the `process_rewards` function is responsible for compounding staking rewards by withdrawing them from all delegated validators.

However, the current implementation is inefficient as it uses a large number of messages. Specifically, rewards are withdrawn from all delegated validators, and then for each such withdrawal, the reward contract is called, which then delegates the reward across all validators based on their weights.

For example, if there are 10 validators, the contract will send 10 messages to withdraw rewards, then for each of these 10 messages, it will send another 10 messages to delegate the rewards, resulting in a total of 100 messages. This can quickly exceed the block gas limit, preventing reward processing.

Recommendation

We recommend optimizing the reward processing logic by batching the reward delegation.

Status: Resolved

13. Lack of a strict withdrawal queue of unbonded staking tokens

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/execute.rs:751-841`, the `process_unbonding` function allows the Escher team's off-chain service to call with a specific unbonding record `id`, to process the unbonding request and transfer the unbonded staking tokens to the user.

However, this approach does not enforce a strict order for processing unbondings in which the unbonding records have been created, such as First-In-First-Out (FIFO). As a result, the order of unbondings can be manipulated, potentially leading to unfair prioritization or delays for specific users or quickly bypassing slashing penalties.

Recommendation

We recommend implementing a strictly enforced unbonding queue, which processes unbonding requests in a predetermined order, such as FIFO, to ensure fairness and transparency.

Status: Resolved

14. Incorrect exchange rate update when total LST token supply is zero

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/state.rs:65-69`, the `update_exchange_rate` function updates the exchange rate based on the `total_bond_amount` and `total_supply`. However, if `total_supply` is zero, the function does not update the exchange rate, leaving it at its previous value.

This is specifically problematic when, at a later point in time, all LST tokens are burned so that the `total_supply` becomes zero. In this case, the exchange rate will not be updated and reset to 1, and the contract will continue to use the outdated exchange rate for subsequent bond operations.

Recommendation

We recommend setting the exchange rate to 1 when `total_supply` or `total_bond_amount` is zero.

Status: Resolved

15. Redelegation amount can be zero, preventing migration of staking tokens to a new validator set

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:285-338`, the `get_restaking_msgs` function builds the `Redelegate` messages to redelegate the stake from the previous to the new validators.

However, the function does not check if the redelegate `diff_amount` for a validator is zero before adding the message to the `undelegate_msg` array. This will lead to an error in Cosmos SDK when processing the `Redelegate` message, as it requires a positive amount. As a result, redelegating the staking tokens to a new validator set will fail.

Recommendation

We recommend adding a check to ensure that `diff_amount` is greater than zero.

Status: Resolved

16. Incorrect message interface used for reward contract

Severity: Major

In `contracts/liquidstaking/liquidstaking/src/msg.rs:48-55`, the `ExecuteRewardMsg` enum interface is defined, which is used to dispatch messages to the rewards contract.

However, this is incorrect because the reward contract's interface messages are defined in `contracts/reward/src/msg.rs:16-25`.

Consequently, updating the `fee_receiver` and `fee_rate` parameters will fail because the interface does not match the reward contract, as seen in `contracts/liquidstaking/liquidstaking/src/execute.rs:715-727`.

Recommendation

We recommend modifying the implementation so the `liquidstaking` contract utilizes the correct `reward` contract's message interface defined in `contracts/reward/src/msg.rs:16-25`.

Status: Resolved

17. Unrestricted unbonding records can be queried if the `min` pagination parameter is unspecified

Severity: Minor

In `contracts/liquidstaking/liquidstaking/src/query.rs:162-175`, the `query_unbond_record` function sets the `max_bound` pagination upper limit to the user-provided `max` parameter when the lower bound `min` is unspecified, i.e., `None`. This behavior contrasts with the scenario where `min` is specified, in which case `max_bound` is capped at `min + 50` to prevent querying an excessively large number of records.

Recommendation

We recommend implementing a check to limit the `max` value when `min` is `None`. Specifically, if `min.is_none() && max > 50`, set `max_id` to 50. This will ensure the query is restricted to a maximum of 50 records.

Status: Resolved

18. Unrestricted querying of unreleased unbond records for a specific staker address

Severity: Minor

In `contracts/liquidstaking/liquidstaking/src/query.rs:177-193`, the `query_unbond_record` function queries unreleased unbond records for a specific staker address. While a minimum bound is set for the query, the maximum bound is not set to `max_bound`, allowing the retrieval of all unreleased records for a staker without any limit, bypassing the contract's restriction of 50 records.

Recommendation

We recommend using the `max_bound` value to set the upper limit for the range to be queried.

Status: Resolved

19. Inability to query all unbonded records regardless of staker address or release status

Severity: Minor

In contracts/liquidstaking/liquidstaking/src/query.rs:144-230, the query_unbond_record function that processes the UnbondRecord query message currently lacks handling for the scenario where both the staker and released query parameters are None. This omission prevents the ability to query all unbonded records regardless of the staker address and release status, and limits the flexibility of the query.

Recommendation

We recommend querying all unbonded records in a paginated manner if staker and released are both None.

Status: Acknowledged

20. Undelegate amounts can potentially be off-by-one in the unbonding process

Severity: Minor

In contracts/liquidstaking/liquidstaking/src/execute.rs:772-780, the process_unbonding function reduces the actual unbonding amount by one during the unbonding process, due to allowing the contract's current balance of the underlying staking token to be one unit less than the intended unbonding amount.

```
if transfer_amount < (unbond_rec.undelegate_amount - Uint128::one()) {
    return Err(ContractError::NotEnoughAvailableFund {});
}

let mut undelegate_amount = unbond_rec.undelegate_amount;
if transfer_amount < undelegate_amount {
    undelegate_amount = transfer_amount;
}
```

For example, if the undelegate_amount is 100 and the transfer_amount (i.e., the underlying staking token balance) is 99, it will result in the undelegate_amount being reduced by one to 99. As a result, it will undelegate one less token than specified by the unbonding record's undelegate_amount.

We classify this issue as minor because it is very unlikely to occur in practice, as the contract's balance of the underlying staking token would need to be exactly one unit less than the intended unbonding amount.

Recommendation

We recommend implementing a strict equality check between undelegate_amount and transfer_amount to ensure that the full amount intended for unbonding is used.

Status: Resolved

21. Unhandled delegation query errors can lead to an incorrect liquid staking token exchange rate

Severity: Minor

In `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs`, the `get_actual_total_delegated`, `get_actual_total_reward`, and `get_unclaimed_reward` functions handle different kinds of delegation queries. These functions currently default to a value of zero if the query fails due to the failure to check the error case of the returned query result. This can lead to unexpected behavior in the contract's logic.

For instance, in the `get_actual_total_delegated` function, if the `query_all_delegations` call fails, the total delegated amount is set to zero and returned. This can result in incorrect exchange rate calculations and, thus, erroneous contract states.

We classify this issue as minor severity because of the low likelihood of such query failures.

Recommendation

We recommend modifying these functions to return an error if the query fails.

Status: Resolved

22. Updating the validator set does not account for rounding errors in delegation amounts

Severity: Minor

Updating the current validator set with the `UpdateValidators` message redelegates the currently staked tokens from the previous validators to the new validators based on their configured weights. A map containing the pro-rata delegation amounts for each validator is calculated in the `get_validator_delegation_map_base_on_weight` function in `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:216-235`.

However, this function does not account for rounding errors that may occur during the calculation when calling the `calculate_delegated_amount` function. This contrasts the `get_delegate_to_validator_msgs` function implemented in lines `340-393`, which handles such rounding errors by adding the remaining amount, i.e., the rounding error, to the first validator.

The absence of carrying over the rounding error and subsequently adding it to one of the validators affects the accuracy of the delegation process, leading to marginally lower delegation amounts.

Recommendation

We recommend delegating any remaining delegation amount to the first validator to ensure the total delegated amount is fully distributed among validators.

Status: Resolved

23. Missing validations when configuring validators and quote denoms

Severity: Minor

When the contract owner configures validators and quote denoms, no validation ensures there are no duplicate validators and quote denoms supplied in `contracts/liquidstaking/liquidstaking/src/contract.rs:42-49` and `88-90`.

Specifically, providing duplicate key entries would cause the previous value to be overwritten and may cause severe consequences. For example, the `total_delegated_amount` will be incorrectly computed in `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:210` if there are duplicate validator keys, causing incorrect distributions of delegate amounts.

Additionally, the validator's weight should be validated to ensure the weight amount is larger than zero to avoid the transaction failing due to delegating and undelegating zero amounts.

Furthermore, the `UpdateQuoteToken` message in `contracts/liquidstaking/liquidstaking/src/execute.rs:995` should ensure the provided `channel_id` parameter equals to `quote_token.channel_id`, as demonstrated in `contracts/liquidstaking/liquidstaking/src/contract.rs:89`.

We classify this issue as minor because it can only be caused by a misconfiguration from the contract owner, a privileged address.

Recommendation

We recommend applying the following recommendations when instantiating the contract and in the `UpdateValidators` and `UpdateQuoteToken` entry points:

- Validate that there are no duplicate validator addresses and quote token channel IDs.
- Validate addresses with `deps.api.addr_validate`
- Validate that the validator weight is not zero.

- Validate that the channel_id parameter equals quote_token.channel_id in the UpdateQuoteToken message.

Status: Resolved

24. Missing slippage control mechanisms for Bond and Unbond messages

Severity: Minor

In contracts/liquidstaking/liquidstaking/src/contract.rs:112-113, the Bond and Unbond messages do not implement a slippage parameter to enforce slippage control. This is problematic because users cannot specify slippage protection when bonding and unbonding funds, which may cause a loss of funds if an exploit occurs, as demonstrated in the "The first depositor can trigger a share inflation attack to steal user funds" issue.

For comparison, the ZkgmMessage::Bond and ZkgmMessage::Unbond messages implement the slippage and expected parameters for slippage controls.

Recommendation

We recommend implementing slippage control mechanisms for the Bond and Unbond messages.

Status: Resolved

25. Potential incorrect computing due to ceiling usage

Severity: Minor

In contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:140 and contracts/reward/src/helpers.rs:34, the to_uint_ceil function is called to ceil the fraction to compute the delegation amount. This is problematic because the ceiled fraction may result in delegating more funds than intended, causing the transaction to fail due to insufficient funds.

Recommendation

We recommend replacing it with to_uint_floor.

Status: Resolved

26. Ineffective ownership update in rewards contract

Severity: Minor

In `contracts/reward/src/execute.rs:99`, the `update_ownership` function is intended to allow the contract owner to update ownership. However, the `cw_ownable::update_ownership` function call is not implemented to perform the ownership updates, making this function ineffective.

Additionally, the `reward` contract's owner is the `liquidstaking` contract, as seen in `contracts/liquidstaking/liquidstaking/src/contract.rs:60`. Since there is no entry point for the `liquidstaking` contract to call the `update_ownership` function in the reward contract, the ownership cannot be transferred.

Recommendation

We recommend implementing the `cw_ownable::update_ownership` function call and a privileged entry point in the `liquidstaking` contract to update the reward contract's ownership. If ownership does not need to be transferred, consider removing the `update_ownership` function.

Status: Resolved

27. Overflow checks are not enabled in production, and the release profile is missing

Severity: Minor

The liquidstaking contract lacks a release profile in `contracts/liquidstaking/liquidstaking/Cargo.toml`, resulting in `overflow-checks` being disabled in production. Overflow checks are crucial for ensuring that arithmetic operations do not exceed the maximum value that can be stored in a variable, which can lead to unexpected behavior or vulnerabilities in the contract.

Recommendation

We recommend specifying a release profile and enabling overflow checks.

Status: Resolved

28. Double sender validations in `split_reward`

Severity: Informational

In `contracts/reward/src/execute.rs:14-18`, the `split_reward` function performs two authentication validations to ensure the caller is the contract owner and the `config.lst_contract_address`. Implementing double validations is unnecessary, as the sender can only be one of them.

Recommendation

We recommend only performing a single authentication.

Status: Resolved

29. Use `deps.api.debug` for logging purposes

Severity: Informational

In multiple instances of the liquidstaking contract, development log messages are stored in `LOG`. However, it only retains a single log message, which is overwritten each time a new message is logged. This approach is ineffective for maintaining a comprehensive log history and may be a leftover from development.

Similarly, `println!` is used in many instances, which is not recommended for production environments. This logging method is not captured in the contract's execution environment and should be replaced with a more appropriate logging mechanism.

Recommendation

We recommend using `deps.api.debug` for logging purposes.

Status: Resolved

30. Handle the optional `amount` parameter in the `unbond` function to use the current liquid staking token balance if `None` is passed

Severity: Informational

In `contracts/liquidstaking/liquidstaking/src/execute.rs:291-372`, the `unbond` function accepts an `amount` parameter of type `Option<Uint128>`. However, the function immediately calls the `unwrap` function on this parameter without checking if it is `Some`. This defeats the purpose of using an `Option` type, which would allow the caller to pass `None` if they do not want to specify an amount and instead wish to unbond the entire liquid staking token balance.

Recommendation

We recommend either updating the function signature to accept a `Uint128` type directly or handling the `Option` type and providing the current liquid staking token balance as a default value if `None` is passed.

Status: Resolved

31. Unneeded `set_contract_version` call

Severity: Informational

In `contracts/liquidstaking/liquidstaking/src/contract.rs:187`, the `migrate` function calls the `cw2::set_contract_version` function to update the contract name and version. However, this is not required as the contract name and version are already updated when `cw2::ensure_from_older_version` is called, as seen in https://github.com/CosmWasm/cw-minus/blob/v2.0.0/packages/cw2/src/migrate.rs#L31.

Recommendation

We recommend removing the `cw2::set_contract_version` function call.

Status: Resolved

32. Code quality improvements

Severity: Informational

In the following instances, code quality can be improved:

- In `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:167`, the `undelegate_amount_dec` variable can be declared outside the `for` loop as the value remains the same across the iteration.
- The `execute_burn` variable defined in `contracts/liquidstaking/liquidstaking/src/reply.rs:153` is misleading, as the message executed is `Cw20ExecuteMsg::Transfer`, not a burn message.
- The `undelegate_msg` variable defined in `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:303` and `320` is misleading, as the message executed is `StakingMsg::Redelegate`, not an undelegate message.
- The comment for the `get_actual_total_reward` function in `contracts/liquidstaking/liquidstaking/src/utils/delegation.rs:51` is incorrect because it reflects the usage for the `get_actual_total_delegated` function.

Recommendation

We recommend improving the code quality by applying the abovementioned recommendations and updating misleading variables and comments.

Status: Resolved