Security Audit Report

Zephyrus

v1.1

November 24, 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 Moonkitt Labs to perform a security audit of Zephyrus 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/moonkitt-lab/zephyrus_wasm
Commit	`6556f9d5f119807d6cf674d599f4cb68381e6d66`
Scope	All contracts were in scope.
Fixes verified at commit	`a1766a296bc4cc85796b0f8ef20e8921b42be1bd`

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

Zephyrus is a decentralized governance protocol that enables users to lock tokens into vessels and either vote directly or delegate voting power to specialized operators called Hydromancers who participate in proposal voting on the Hydro platform.

The protocol manages time-weighted voting shares (TWS), processes tribute distributions from winning proposals, and handles commission splits between users, Hydromancers, and the protocol treasury.

Users can create multiple vessels with different lock durations, switch between self-control and delegation modes, and claim rewards based on their proportional voting power contribution to successful proposals.

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-High	-
Level of documentation	Medium	-
Test coverage	Medium-High	Coverage: 70.93%

Summary of Findings

No	Description	Severity	Status
1	Unlimited tribute additions enable `execute_claim` denial of service	Critical	Resolved
2	Unlimited vessel assignments enable denial of service against Hydromancers	Critical	Resolved
3	Attackers can claim rewards on behalf of Zephyrus, causing user rewards to be stuck	Critical	Resolved
4	Token dust spam enables denial of service on balance queries	Critical	Resolved
5	Decommissioning vessels deletes unclaimed rewards permanently	Major	Acknowledged
6	Users bypass Hydromancer commissions by taking control before claiming	Major	Resolved
7	Reward calculations use current Hydromancer instead of historical voting Hydromancer	Major	Resolved
8	Missing validation for empty admins	Minor	Resolved
9	Admins cannot be updated	Minor	Resolved
10	Usage of `saturating_sub` may hide unexpected errors	Minor	Resolved
11	Incorrect usage of processed tribute value for computing user funds	Minor	Resolved
12	Accumulated dust from rounding lacks sweep mechanism	Informational	Acknowledged
13	Anti-pattern using `is_some` before unwrap	Informational	Resolved
14	Duplicate user ID retrieval functions	Informational	Resolved
15	Unused `consolidate_coins` function	Informational	Resolved
16	Inconsistent error types between `StdError` and `ContractError`	Informational	Acknowledged

17	Unnecessary computation before conditional check	Informational	Resolved
18	Use iterator chains to optimize collection operations in `auto_maintenance`	Informational	Resolved
19	`calculate_rewards_amount_for_vessel_on_tribute` violates maintainability standards	Informational	Resolved
20	Redundant `vessel_id` assignment in reply handler	Informational	Resolved
21	Missing lock duration validation causes failed Hydro transactions	Informational	Resolved
22	Commission updates apply immediately without user notification period	Informational	Acknowledged
23	Commission rate validation allows exactly 100% during updates	Informational	Resolved
24	Unimplemented `query_voting_power` function	Informational	Resolved
25	Missing entry point to manage hydromancers	Informational	Acknowledged
26	Unused `lock_ids` variable during vessel processing	Informational	Resolved
27	Inefficient manual clearing of mappings	Informational	Resolved
28	Unoptimized conditional check order in voting power calculation	Informational	Resolved
29	Duplicate `hydro_lock_ids` are not deduplicated in `execute_change_hydromancer`	Informational	Resolved
30	Lack of role-based access controls for the pausing mechanism	Informational	Acknowledged
31	Event logging lacks sufficient detail	Informational	Resolved
32	Panic calls violate error handling best practices	Informational	Resolved

Detailed Findings

1. Unlimited tribute additions enable `execute_claim` denial of service

Severity: Critical

The `query_hydro_outstanding_tribute_claims` function in `contracts/main/src/contract.rs:226-232` retrieves all tributes for a round. Based on Hydro's `add_tribute` functionality, anyone can add as many tributes as they want for any proposal, and what is more, they can even add tributes retroactively, meaning that tributes can be sent to proposals in past rounds.

Malicious users can spam thousands of dust tributes, creating an enormous claim list, which could lead to out of gas conditions when outstanding tributes are being processed.

Attack scenario:

1. Zephyrus votes for a proposal and that proposal wins
2. Attacker calls Hydro's `add_tribute` thousands of times (he could have randomly guessed that this will be a winning proposal, or he can monitor the mempool and see which the winning proposal will be)
3. The first user to call `execute_claim` will invoke `query_hydro_outstanding_tribute_claims`
4. Contract must iterate thousands of tributes in `process_outstanding_tribute_claims`
5. Transaction exceeds gas limit
6. All users permanently unable to claim that round's reward
7. Attacker can continue doing this at minimal cost

Recommendation

We recommend implementing an admin controlled flow, where either admins or hydromancers call `process_outstanding_tribute_claims` using paginated chunks for the tribute data coming from `query_hydro_outstanding_tribute_claims`.

Status: Resolved

2. Unlimited vessel assignments enable denial of service against Hydromancers

Severity: Critical

The `state::get_vessels_by_hydromancer` function in `contracts/main/src/state.rs:508-524` retrieves all vessels for a Hydromancer without limit. When `execute_hydromancer_vote` is called, it must iterate through every assigned vessel.

Attack scenario:

1. Attacker creates thousands of dust locks as there is no minimum amount limit.
2. Attacker assigns all locks to target Hydromancer using `execute_change_hydromancer` or simply transfers their NFTs to Zephyrus to the specific Hydromancer.
3. When the Hydromancer attempts to vote, the contract iterates over thousands of vessels
4. Transaction exceeds gas limit and reverts
5. Hydromancer is permanently unable to vote

The issue compounds in `contracts/main/src/reply.rs:110-129` where tribute claims iterate all Hydromancers and their vessels.

Recommendation

We recommend enforcing a maximum cap for vessels per Hydromancer.

Status: Resolved

3. Attackers can claim rewards on behalf of Zephyrus, causing user rewards to be stuck

Severity: Critical

In `contracts/main/src/contract.rs:236-250`, the `execute_claim` function distributes user rewards only when `outstanding_tributes_result` returns `Some(_)`. If it returns `None`, which could occur when someone externally triggers a claim on behalf of the contract (i.e., by calling `claim_tribute` in Hydro with the `voter_address` parameter set to Zephyrus's contract address), then user rewards will not be distributed.

Below is an example of an attack scenario:

1. Rewards are available for users to claim.
2. An attacker (or any user) calls `claim_tribute` on the Hydro contract, setting `voter_address` to Zephyrus's contract address.
3. The rewards are sent to the contract, and Hydro marks them as already claimed.

4. When users subsequently call `execute_claim`, `outstanding_tributes_result` becomes `None`, and no rewards are distributed even though they should be.

Consequently, users will successfully call `execute_claim` but receive no rewards, leading to a loss of rewards.

Recommendation

We recommend modifying the implementation so that rewards are still distributed even if someone externally claims on behalf of Zephyrus.

For example, the contract can utilize CosmWasm's `query` method on Hydro's `TRIBUTE_CLAIMED_LOCKS` state to verify whether rewards have been claimed on behalf of Zephyrus and process the corresponding distributions accordingly.

Status: Resolved

4. Token dust spam enables denial of service on balance queries

Severity: Critical

The contract queries all token balances in two locations without any limits:

- In `contracts/main/src/contract.rs:271`, the `query_all_balances(contract_address.clone())?` call enumerates all token balances held by the contract inside function `process_outstanding_tribute_claims`
- In `contracts/main/src/contract.rs:681`, the `BankQuery::AllBalances` is used to construct `DecommissionVesselsReplyPayload` containing all fetched `previous_balances` inside function `execute_decommission_vessels`
- A third iteration occurs in `contracts/main/src/reply.rs:525`, where `balance_query` relies on `BankQuery::AllBalances`. This instance was identified by the Client.

It is possible to attack the contract by sending numerous dust amounts of different token denoms to the contract address, which can be achieved using `x/tokenfactory denoms`.

Each additional denom increases the gas cost of operations that enumerate balances, eventually causing out of gas condition.

Recommendation

We recommend implementing a whitelist of expected token denoms or processing only specific tokens rather than enumerating all balances.

Status: Resolved

5. Decommissioning vessels deletes unclaimed rewards permanently

Severity: Major

The `handle_unlock_tokens_reply` function in `contracts/main/src/reply.rs:573-579` calls `state::remove_vessel` for each unlocked vessel.

The `remove_vessel` function in `contracts/main/src/state.rs:565-617` deletes all vessel data including:

- `VESSELS.remove(storage, hydro_lock_id)`
- `VESSEL_SHARES_INFO.remove(storage, (round_id, hydro_lock_id))`
- `USER_VESSELS.update(...)`

When users have unclaimed rewards from previous rounds, the vessel data required for reward calculation is permanently deleted. Users lose all historical rewards with no recovery mechanism.

Recommendation

We recommend modifying `execute_decommission` to check for and automatically claim all pending rewards before vessel removal, or maintain a separate `HISTORICAL_VESSELS` map for reward claims.

Status: Acknowledged

Users can only decommission their own vessels, and Zephyrus will have a frontend safeguard that checks for pending unclaimed rewards before decommissioning. When unclaimed rewards are detected, the frontend automatically batches a claim message prior to the decommission transaction, preventing unintended reward forfeiture.

6. Users bypass Hydromancer commissions by taking control before claiming

Severity: Major

The `calculate_rewards_amount_for_vessel_on_tribute` function in `contracts/main/src/helpers/rewards.rs:210-230` checks the vessel's current control state. This is problematic for when a user changes their vessel's ownership.

Attack scenario:

1. Alice delegates vessel to Hydromancer in round 1
2. Hydromancer votes for a proposal
3. Alice executes `take_control` in the next rounds
4. Alice claims rewards for round 1

5. System sees that the vessel is under user control and gives Alice rewards as if her vessel was under her control for that round
6. Hydromancer loses rightful commission

Recommendation

We recommend storing control state per round `VESSEL_ROUND_CONTROL: Map<(RoundId, VesselId), bool>` and checking historical state during claims.

Status: Resolved

7. Reward calculations use current Hydromancer instead of historical voting Hydromancer

Severity: Major

The `calculate_rewards_amount_for_vessel_on_tribute` function in `contracts/main/src/helpers/rewards.rs:206` retrieves the current vessel state. When calculating voting power in lines `245-250`, the function uses the vessel's current `hydromancer_id`. If a user changes Hydromancers between voting and claiming, an issue will arise:

1. Round 1: Vessel controlled by Hydromancer A with 1000 TWS
2. Hydromancer A votes, accumulating 1000 TWS for proposal
3. User changes vessel to Hydromancer B with 5000 TWS
4. User claims Round 1 rewards
5. Calculation incorrectly uses Hydromancer B's 5000 TWS instead of Hydromancer A's 1000 TWS

This inflates or deflates rewards based on the new Hydromancer's voting power rather than the actual voting power used.

Recommendation

We recommend storing Hydromancer assignments per round in a new `VESSEL_ROUND_HYDROMANCER` map to preserve historical voting context.

Status: Resolved

8. Missing validation for empty admins

Severity: Minor

In `contracts/main/src/contract.rs:62-67`, the `instantiate` function does not validate that `msg.whitelist_admins` is non-empty before processing. If an empty list is

provided, the contract may deploy without an admin address configured, preventing proper administrative operations and management.

Consequently, deploying the contract with an empty `whitelist_admins` list may result in a non-functional or inaccessible contract state, as no entity would be authorized to perform administrative tasks.

Recommendation

We recommend adding a validation check to ensure `msg.whitelist_admins` is not empty.

Status: Resolved

9. Admins cannot be updated

Severity: Minor

In `contracts/main/src/contract.rs:62-67`, the `instantiate` function initializes `whitelist_admins` but provides no mechanism to update or modify the list after deployment. If any admin account becomes compromised or loses access to its private key, the contract cannot be properly managed or recovered, effectively locking administrative control.

Recommendation

We recommend implementing a permissioned entry point that allows authorized entities to update admin addresses.

Status: Resolved

10. Usage of `saturating_sub` may hide unexpected errors

Severity: Minor

In `contracts/main/src/reply.rs:89`, the `handle_claim_tribute_reply` function calculates `balance_expected_adjusted` using `balance_expected.saturating_sub(total_distributed)`.

This approach is unsafe because `saturating_sub` silently returns `0` if `total_distributed` exceeds `balance_expected`, which should never occur under correct logic.

However, if such a case occurs, accounting errors will be masked, potentially leading to the contract operating with an incorrect balance.

Recommendation

We recommend replacing `saturating_sub` with `checked_sub` and returning an explicit error if an overflow occurs.

Status: Resolved

11. Incorrect usage of processed tribute value for computing user funds

Severity: Minor

In `contracts/main/src/query.rs:234-241`, when a tribute has already been processed, the `query_vessels_rewards` function fetches the stored processed tribute via `state::get_tribute_processed` and then passes that value into `calculate_protocol_comm_and_rest` to derive `users_funds`.

This is incorrect because `state::mark_tribute_processed` stores the `users_and_hydromancers_funds` amount in `contracts/main/src/reply.rs:213`, not the original rewards amount.

Consequently, the `query_vessels_rewards` function will return a lower `users_funds` value than intended, underreporting claimable rewards and misleading users.

Recommendation

We recommend using the value returned by `state::get_tribute_processed` directly as `users_funds` when the tribute has already been processed.

Status: Resolved

12. Accumulated dust from rounding lacks sweep mechanism

Severity: Informational

The `to_uint_floor` function is used in multiple locations:

- `contracts/main/src/helpers/rewards.rs:310, 312, 359, 479`
- `contracts/main/src/query.rs:288`
- `contracts/main/src/reply.rs:147`

Each rounding operation loses fractional amounts that accumulate in the contract. With no sweep mechanism, these funds become permanently locked.

Recommendation

We recommend implementing an `execute_sweep_dust` function restricted to admins that transfers accumulated rounding differences to the fee recipient.

Status: Acknowledged

Zephyrus acknowledges that rounding operations may result in dust accumulation. However, implementing a sweep mechanism is not feasible because the contract cannot reliably distinguish between legitimately unclaimed user funds and accumulated rounding errors. Given this constraint, Zephyrus accepts that dust will accumulate in the contract.

13. Anti-pattern using `is_some` before `unwrap`

Severity: Informational

`is_some` is used before `unwrap` In `contracts/main/src/helpers/rewards.rs:214-215`.

Recommendation

We recommend replacing it with `if let Some(vessel_harbor) = vessel_harbor {`.

Status: Resolved

14. Duplicate user ID retrieval functions

Severity: Informational

Two functions provide identical functionality in `contracts/main/src/state.rs`:

- `get_user_id_by_address` in lines `246-248`
- `get_user_id` in lines `287-290`

Both retrieve user IDs by address with only error type differences.

Recommendation

We recommend removing `get_user_id_by_address` and use only `get_user_id` throughout the codebase.

Status: Resolved

15. Unused `consolidate_coins` function

Severity: Informational

The `consolidate_coins` function in `contracts/main/src/helpers/vectors.rs:24-37` is never called anywhere in the codebase.

Recommendation

We recommend removing the unused function.

Status: Resolved

16. Inconsistent error types between `StdError` and `ContractError`

Severity: Informational

The codebase inconsistently uses `StdResult/StdError` and `Result<_, ContractError>` across different modules, for example:

- Lines `112-119`: `mark_tribute_processed` and `get_tribute_processed` return `StdResult`
- Lines `132-136`: `add_new_rewards_to_hydromancer` uses `StdError::generic_err` instead of `ContractError`
- Lines `221-226`: `insert_new_user` uses `StdError::generic_err("User already exists")`
- Lines `246-248`: `get_user_id_by_address` returns `StdResult<UserId>`

This forces implicit error conversions throughout the codebase and makes error handling inconsistent.

Recommendation

We recommend standardizing all internal functions to return `Result<T, ContractError>` for consistent error handling. Only use `StdResult` for CosmWasm entry points that require it by specification.

Status: Acknowledged

17. Unnecessary computation before conditional check

Severity: Informational

In `contracts/main/src/helpers/rewards.rs:126-129`, voting power is calculated before checking if it is needed.

Recommendation

We recommend moving the conditional check before the multiplication.

Status: Resolved

18. Use iterator chains to optimize collection operations in `auto_maintenance`

Severity: Informational

The `collect_vessels_needing_auto_maintenance` function in `contracts/main/src/helpers/auto_maintenance.rs` uses an underperforming looping pattern when filtering out the vessels that opted in for `auto_maintenance`.

Recommendation

We recommend utilizing iterator chains, which will remove the need for sorting.

Status: Resolved

19. `calculate_rewards_amount_for_vessel_on_tribute` violates maintainability standards

Severity: Informational

The `calculate_rewards_amount_for_vessel_on_tribute` function in `contracts/main/src/helpers/rewards.rs:191-205` accepts 11 parameters, making it error-prone and difficult to maintain.

Recommendation

We recommend creating a `VesselRewardContext` struct to encapsulate related parameters.

Status: Resolved

20. Redundant `vessel_id` assignment in reply handler

Severity: Informational

In `contracts/main/src/reply.rs:318`, the `vessel_id` is assigned inside the loop after a conditional check. The assignment occurs for every non-skipped vessel when it could be extracted once.

Recommendation

We recommend moving the assignment before the conditional `let vessel_id = vessel_shares_info.lock_id;` at line `315`.

Status: Resolved

21. Missing lock duration validation causes failed Hydro transactions

Severity: Informational

The `execute_update_vessels_class` function in `contracts/main/src/contract.rs:573-618` sends `hydro_lock_duration` directly to Hydro without validation.

Unlike `execute_receive_nft` which calls `validate_lock_duration` in lines `397-401`, invalid durations cause Hydro to reject the transaction. Users waste gas on failed transactions with unclear error messages.

Recommendation

We recommend validating the `lock_duration` before sending it to Hydro.

Status: Resolved

22. Commission updates apply immediately without user notification period

Severity: Informational

The `execute_update_commission_rate` function in `contracts/main/src/contract.rs:172-192` updates the commission rate immediately at line `189`. Users have no time to react to unfavorable rate changes before they affect pending rewards. This allows admins to suddenly increase rates and capture more rewards without warning.

Recommendation

We recommend implementing a timelock mechanism with a 7-day delay between commission rate announcement and activation, allowing users to withdraw or adjust positions.

Status: Acknowledged

23. Commission rate validation allows exactly 100% during updates

Severity: Informational

The instantiate function in `contracts/main/src/contract.rs:77-78` validates commission rates using `>= Decimal::one()`.

However, the `execute_update_commission_rate` function in `contracts/main/src/contract.rs:180` validates with `> Decimal::one()`.

This inconsistency allows setting exactly `Decimal::one()` (100% commission) through updates but not during instantiation. A 100% commission rate would drain all user rewards to the protocol.

Recommendation

We recommend changing line `180` to use `>= Decimal::one()` to maintain consistency and prevent 100% commission rates. Additionally, we recommend implementing a sanity threshold so the commission cannot exceed some predefined maximal value.

Status: Resolved

24. Unimplemented `query_voting_power` function

Severity: Informational

In `contracts/main/src/query.rs:76-78`, the function `query_voting_power` is declared but not implemented. Leaving this function as a `todo!()` placeholder will cause the contract to panic if it is ever invoked, interrupting execution in production deployments.

Recommendation

We recommend implementing the `query_voting_power` function or removing it entirely if it is not required.

Status: Resolved

25. Missing entry point to manage hydromancers

Severity: Informational

Across the codebase, no entry point allows administrators to add or remove hydromancers. This limits the system's flexibility and prevents dynamic updates, which may be necessary if new hydromancers join or existing ones need to be revoked.

Recommendation

We recommend implementing a permissioned entry point that allows administrators to add or remove hydromancers.

Status: Acknowledged

Zephyrus will launch with a single Hydromancer. The ability to add or remove Hydromancers shall be implemented in a future release.

26. Unused `lock_ids` variable during vessel processing

Severity: Informational

In `contracts/main/src/reply.rs:303`, the `lock_ids` vector is initialized and populated in line `419` via `lock_ids.push(vessel.hydro_lock_id)`, but is never used afterwards. It is neither included in emitted events nor returned in the response, making its purpose unclear and potentially omitting useful information for off-chain tracking or debugging.

Recommendation

We recommend emitting the `lock_ids` as part of the related event data or removing the variable entirely if it serves no purpose.

Status: Resolved

27. Inefficient manual clearing of mappings

Severity: Informational

In `contracts/main/src/state.rs:1076-1084`, the `clear_distribution_tracking` function iterates through all keys in `TRIBUTE_DISTRIBUTED_AMOUNTS` and removes them one by one.

This is inefficient as CosmWasm's `Map` storage API provides a `clear` method that can remove all entries directly, which is more concise and efficient.

Recommendation

We recommend replacing the manual key iteration and removal with a direct call to `TRIBUTE_DISTRIBUTED_AMOUNTS.clear(storage)`.

Status: Resolved

28. Unoptimized conditional check order in voting power calculation

Severity: Informational

In `contracts/main/src/helpers/rewards.rs:114-131`, the condition `if *locked_round < locked_rounds` is evaluated after computing `voting_power_contribution`.

This causes unnecessary computation for rounds that will be skipped anyway.

Recommendation

We recommend moving the conditional check `if *locked_round < locked_rounds` before performing the voting power calculations to reduce gas consumption.

Status: Resolved

29. Duplicate `hydro_lock_ids` are not deduplicated in `execute_change_hydromancer`

Severity: Informational

In `contracts/main/src/contract.rs:794`, providing duplicate `hydro_lock_ids` (e.g., `vec![0, 0, 0]`) to the `execute_change_hydromancer` function will inflate the `HYDROMANCER_TW_SHARES_BY_TOKEN_GROUP_ID` state when passed to the `initialize_vessel_tws` function call in line `827`.

While Hydro's current `query_lockups_shares` implementation mitigates this using an internal `HashSet`, this behavior could change in the future, potentially reintroducing the vulnerability.

Recommendation

We recommend deduplicating `hydro_lock_ids` in the `execute_change_hydromancer` function before sending them to Hydro. This ensures consistent safety regardless of Hydro's internal implementation changes.

Status: Resolved

30. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The codebase implements a pausing mechanism, which is in line with best practices. However, all of the administrative functions of the contract are centralized in the admin role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn on and off the pausing mechanism.

Status: Acknowledged

Zephyrus plans to implement role-based access control for the pausing mechanism in a future release.

31. Event logging lacks sufficient detail

Severity: Informational

In `contracts/main/src/contract.rs:234`, the claim event only logs the action attribute without including function parameters such as `tranche_id` and other relevant arguments. This reduces the ability to track and analyze contract interactions through event logs, making monitoring more difficult.

Recommendation

We recommend adding function arguments to the event attributes to improve observability.

Status: Resolved

32. Panic calls violate error handling best practices

Severity: Informational

In `contracts/main/src/contract.rs:503`, as well as multiple other files such as `./contracts/main/src/state.rs`, `reply.rs`, `query.rs` and `./contracts/main/src/helpers/tws.rs` the code uses `expect` to handle invalid states, which causes a panic if the expectation is violated.

Using panics instead of returning proper errors reduces code maintainability and is discouraged as a generic best practice.

Recommendation

We recommend replacing all `expect` calls with proper error handling using `Result` or `Error` types and descriptive error messages.

Status: Resolved