Audit Report

Zodiac Protocol Periphery

v1.0

March 18, 2024

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

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

Introduction

Purpose of This Report

Oak Security has been engaged by Osmosis Grants Company to perform a security audit of Zodiac Protocol's Periphery 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/zodiac-protocol/periphery
Commit	`8246c247e73ddd88d092840f751d002739199382`
Scope	All contracts were in scope.

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 Zodiac Protocol Periphery contracts extend Zodiac Protocol, which is a DeFi protocol that allows users to manage the risks associated with providing liquidity to AMMs (Automated Market Makers).
These periphery contracts implement the governance functionality, the ZDC token unlocks for Initial Zodiac Builders, the ZDC staking mechanism, vault incentives, and inbound/outbound IBC voting power.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium-High	The protocol uses Stargate messages to communicate with the underlying Cosmos SDK appchain.
Code readability and clarity	Low	-
Level of documentation	Medium	-
Test coverage	Medium-High	`cargo tarpaulin` reports a test coverage for the contracts in scope of 90.42%.

Summary of Findings

No	Description	Severity	Status
1	The staking contract is not able to mint or burn coins	Critical	Resolved
2	Total voting power does not account for the changes in the builder unlock contract and funds in the IBC outbound contract	Critical	Resolved
3	Proposal voting results can be manipulated through flash loans and builder unlock allocations	Critical	Resolved
4	First depositor can be front-run for unfair profit causing direct losses	Critical	Resolved
5	Incorrect total shares determination	Critical	Resolved
6	Replaying `Proliferate` message inflates total voting power	Critical	Resolved
7	Removal of channels automatically removes the associating port and connection identifier	Major	Partially Resolved
8	On-going proposals are rejected after removal of IBC channel	Major	Resolved
9	Emergency governance proposal passes as long as quorum is reached	Major	Resolved
10	Updating the last update timestamp when creating schedules might cause unissued rewards	Major	Resolved
11	`remove_completed_proposal` function incorrectly uses current configuration values	Major	Resolved
12	Possible division by zero error when no liquidity token is deposited	Major	Resolved
13	User's voting power remains after removing channels	Major	Acknowledged
14	`StakePtLp` message does not remove user's deposit when updating the global index	Major	Resolved
15	Insufficient input validations across contracts	Minor	Partially Resolved
16	Owner is allowed to execute arbitrary	Minor	Resolved

	`CosmosMsgs`		
17	Executed proposals can be removed	Minor	Resolved
18	Coin denomination updates lead to locked tokens stuck in the contract	Minor	Resolved
19	Excessive control over funds	Minor	Acknowledged
20	Single-step ownership transfer	Informational	Acknowledged
21	Unused reply ID	Informational	Resolved
22	Lack of event attribute emission	Informational	Acknowledged
23	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
24	Incorrect comments	Informational	Resolved
25	Votes will be considered failed if the response contains a data field	Informational	Acknowledged
26	Excess storage operations performed after removing channels	Informational	Resolved
27	Queries return incomplete configuration	Informational	Resolved
28	Inefficient execution	Informational	Resolved

Detailed Findings

1. The staking contract is not able to mint or burn coins

Severity: Critical

The `BlockBeforeSend Sudo` message, defined in `contracts/periphery/staking/src/contract.rs:229`, is executed by the Osmosis chain when sending tokens through the `Bank` module, and it is used by the protocol to execute custom logic to map tokens to the recipient during a send operation.

This hook is implemented under the assumption that it is not executed when minting or burning tokens.

This assumption does not hold, however, since the mint operation calls the `SendCoinsFromModuleToAccount` function here which calls `SendCoins` here, which implies that the `BlockBeforeSend` hook is executed even during mint and burn operations.

Since `from` is the address of the `tokenfactory` module, an error will occur in line `243` when trying to decrease its zero balance.

This makes the contract unusable.

Please see the `test_mint` test case to reproduce the issue.

Recommendation

We recommend not increasing/decreasing the user's balance during mint/burn operations and not increasing/decreasing the balance of `tokenfactory` module address during the `BlockBeforeSend` sudo message.

Status: Resolved

2. Total voting power does not account for the changes in the builder unlock contract and funds in the IBC outbound contract

Severity: Critical

The `calc_total_voting_power_at` function defined in `contracts/periphery/gov/src/contract.rs:1059` calculates the total voting power at a certain height.

To do so, it fetches the total supply from the staking contract, the number of coins locked in the builder's unlock schedule, and the voting power at the proposal's submission.

This leads to an incorrect voting power because the tokens returned by the builder unlock contract's `BuilderUnlockQueryMsg::State` query can be influenced by new allocations and user withdrawals.

Additionally, it does not account for funds in the IBC outbound contract, leading to a wrong total voting power calculation.

Consequently, the proposals' quorum and results are not correctly computed.

Recommendation

We recommend taking a snapshot of the builder unlock state and accounting for the coins in the IBC outbound contract when calculating the total voting power.

Status: Resolved

3. Proposal voting results can be manipulated through flash loans and builder unlock allocations

Severity: Critical

The `governance` contract accounts for multiple voting sources to determine the outcome of a proposal. However, some of these sources allow attackers to inflate their voting power by taking a flash loan, allowing them to execute governance attacks.

The voting power calculated from the tokens staked in the `voting-power-outbound` contract can be inflated as the data queried in `contracts/periphery/gov/src/contract.rs:1041-1046` reflects the amount on the same block instead of a previous block. An attacker could request a flash loan, `Lock` a large number of tokens in the `voting-power-outbound` contract, `CastVote` in the `governance` contract, `Unlock` the tokens from `voting-power-outbound`, and finally repay the flash loan. Following these steps, they could manipulate the result of any proposal in their favor by paying only the flash loan fee.

Additionally, the builder's unlock contract determines the user's voting power using the `BuilderUnlockQueryMsg::Allocation` query in line `1029`. This is problematic because a user can vote on the proposal, call `ProposeNewReceiver` to transfer the receiver's address to another controlled address, call `ClaimReceiver`, and finally vote on the same proposal again to manipulate the outcome of the poll.

Recommendation

We recommend calculating voting power based on a previous block that cannot be easily manipulated, for example by making use of a `SnapshotMap`. This should be applied to:

- staked tokens in the `voting-power-outbound` contract and
- allocated tokens in the `builder-unlock` contract.

Status: Resolved

4. First depositor can be front-run for unfair profit causing direct losses

Severity: Critical

The `staking` 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/periphery/staking/src/contract.rs:112-118`, the number of shares a user receives depends on `total_shares`, `total_deposit`, and `received_tokens`. However, if a malicious depositor makes a large enough "donation" to the vault at the right time, increasing `total_deposit`, the next depositor will not receive their expected amount of shares.

The below describes a potential 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 `x` tokens into the `staking` contract, Mallory front-runs their transaction with two calls:
    a. Making a minimal initial deposit, let's say `1` token, to obtain one unit of shares and setting `total_shares` to one.
    b. Donating `x/2` tokens to the contract, increasing the `total_deposit` value without increasing `total_shares`.
2. Alice's transaction gets executed containing the `ExecuteMsg::Enter` message, expecting to receive `x` amount of shares. As their deposit is just `1` token below the amount required to get `2` shares, only `1` share will be rewarded.
3. As a result, `total_deposit` would be equal to `1 + 3x/2`, and `total_shares` would be `2`. Making each share worth `1/2 + 3x/4` tokens.
4. Mallory will then be able to return the share to get `1/2 + 3x/4` tokens after having spent just `1 + x/2` tokens, effectively profiting `x/4 - 1/2` tokens.

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 critical.

Recommendation

We recommend implementing virtual shares and assets with sufficient offset to mitigate the issue, as described here.

Status: Resolved

5. Incorrect total shares determination

Severity: Critical

In `contracts/periphery/staking/src/contract.rs:104`, the total shares of `xtoken` are determined via the contract balance. This is problematic because it should be determined by the total supply using the `Supply` bank query instead of the `Balance` query.

Consequently, if the total supply is different from the contract balance, the minted amount will be incorrectly larger than intended, allowing users to withdraw more funds from the protocol and ultimately causing a loss of funds for legitimate stakers.

Recommendation

We recommend determining the total number of shares using the `Supply` bank query.

Status: Resolved

6. Replaying `Proliferate` message inflates total voting power

Severity: Critical

In `contracts/periphery/voting_power_inbound/src/contract.rs:307-311`, the user's voting power is updated, and the total voting power in the `FOREIGN_VOTING_POWER_TOTALS` storage state is increased. However, here is no mechanism in place that prevents voters from replaying the `Proliferate` message to increase the total voting power.

Consequently, malicious users can inflate the total voting power such that it is impossible to reach the quorum during a governance proposal.

Recommendation

We recommend loading and deducting the old voter's power before adding the power and storing it in the `FOREIGN_VOTING_POWER_TOTALS` storage state.

Status: Resolved

7. Removal of channels automatically removes the associating port and connection identifier

Severity: Major

When a channel ID is removed from `contracts/periphery/voting_power_inbound/src/contract.rs:122-140` and `contracts/periphery/voting_power_outbound/src/contract.rs:226-244`, it simultaneously removes the associated port ID and connection ID from their allow lists.

Since one connection can have multiple channels, a malicious channel that got removed will automatically remove the associated connection ID, causing future channels to fail to establish a connection.

Additionally, most of the time, port identifiers are common across multiple chains due to the module name (e.g., "*transfer*"). If a malicious channel uses a common port name and is removed, any future channels that intend to use the same port identifier will fail to be established.

To recover from this situation, a governance proposal needs to be passed to configure and add new connections and port identifiers, which is time-consuming and requires governance participation.

We classify this issue as major because it affects the correct functioning of the system.

Recommendation

We recommend adding a boolean argument to the `RemoveChannel` message that dictates whether the associated port and connection identifier should be removed along with the channel. The associated port and connection identifier should not be removed if the boolean value is false. This change would provide flexibility for governance by allowing the removal of malicious channel IDs without disrupting the establishment of future channels.

Status: Partially Resolved

This issue is partially resolved because the boolean argument has been added, but port identifiers are still removed.

The client states that the port IDs provided to the contract are unique because they point to wasmd contracts so they can be removed.

8. On-going proposals are rejected after removal of IBC channel

Severity: Major

The `governance` contract's `end_proposal` function checks that the current power sources include the set of sources that were present when the proposal started in `contracts/periphery/gov/src/contract.rs:530-536`.

Although this feature's goal is to purge potentially malicious channels from affecting governance, it has the consequence that the proposal is automatically rejected if any of the initial power sources have been removed, even if no votes from the removed channel have been submitted to the proposal.

Recommendation

We recommend identifying IBC votes per channel and excluding them from voting if the channel gets removed, instead of rejecting the proposal entirely.

Status: Resolved

9. Emergency governance proposal passes as long as quorum is reached

Severity: Major

In `contracts/periphery/gov/src/contract.rs:436-439`, the emergency governance proposal is counted as passed if the quorum is reached, irrespective of whether the votes are in favor of or against the proposal.

For example, assume a malicious voter submits a proposal that attempts to remove a legitimate IBC channel. The majority of voters disagree and submit "against" votes so the channel would not be removed. However, since the emergency proposal will pass, as long the quorum is reached, the legitimate IBC channel will be removed eventually.

Recommendation

We recommend updating the implementation to ensure that an emergency proposal passes only when the "for" votes outnumber the "against" votes.

Status: Resolved

10. Updating the last update timestamp when creating schedules might cause unissued rewards

Severity: Major

In `contracts/periphery/vault_incentives_v1/src/contract.rs:170`, `last_update_timestamp` is updated when the owner updates the schedule. This is problematic because the rewards distribution depends on the elapsed staking time, causing a loss of rewards for stakers.

To illustrate, assume the start time is before the current timestamp when the owner creates the schedule. In line `377`, the elapsed time will deduct the current timestamp with the last updated timestamp, not the schedule's start time.

Since the rewards are calculated in line `402` with the denominator as the whole schedule's duration, the stakers will lose a portion of the rewards because the duration from the start date to the last updated timestamp is not included.

Recommendation

We recommend not updating `last_update_timestamp` during the `UpdateSchedule` message.

Status: Resolved

11. `remove_completed_proposal` function incorrectly uses current configuration values

Severity: Major

In `contracts/periphery/gov/src/contract.rs:650-654`, the `remove_completed_proposal` function mutates the proposal status to expired if the current block height exceeds the sum of `proposal.end_block`, `config.proposal_effective_delay`, and `config.proposal_expiration_period`. This implies that current configuration values are used, not the ones at the creation of the proposal.

Consequently, the proposal will expire earlier or later than intended depending on the `proposal_effective_delay` and `proposal_expiration_period` values updated in `update_config` through a successful governance proposal. The expiration block at proposal creation is stored in `proposal.expiration_block`, as seen in line `308`.

Recommendation

We recommend using `proposal.expiration_block` instead.

Status: Resolved

12. Possible division by zero error when no liquidity token is deposited

Severity: Major

In `contracts/periphery/vault_incentives_v1/src/contract.rs:449-451`, the `pt_lp_holders_index` is increased for all staked liquidity token holders if the total yield token supply is zero. However, it does not account for the scenario where the total staked liquidity token in the contract is also zero.

Consequently, the `update_state` function might fail due to a division by zero error when updating indexes.

Recommendation

We recommend validating that `staked_pt_lp` is not zero when increasing `pt_lp_holders_index`.

Status: Resolved

13. User's voting power remains after removing channels

Severity: Major

In `contracts/periphery/voting_power_inbound/src/contract.rs:122-143`, the `RemoveChannel` message resets the `FOREIGN_VOTING_POWER_TOTALS` storage to zero to indicate there will be no voting power in this channel. However, individual voters still have their voting power stored in the `VOTERS` storage state.

Consequently, both `Voter` and `Voters` query messages will still reflect that voters still have outstanding voting power in the removed channel, which is incorrect.

Additionally, if the channel is subsequently readded, `Voters` regain the voting power stored with the old channel since it has not been reset.

Recommendation

We recommend setting the voter's voting power to zero if the channel is removed.

Status: Acknowledged

The client states they will likely release the protocol without a token initially and then roll out the governance/IBC contracts later on, depending on user traction.

14. `StakePtLp` message does not remove user's deposit when updating the global index

Severity: Major

In `contracts/periphery/vault_incentives_v1/src/contract.rs:121-127`, the `StakePtLp` message requires the caller to send `config.pt_lp_token` native denom, which will then call the `update_state` function. This function then queries the total tokens in the contract in line `425` without deducting the user's deposit.

This is problematic because the distributed rewards amount is for the stakers who staked during the elapsed time duration. Since the provided tokens were not staked before, they should not be entitled to a portion of the incentives earned.

Consequently, the overall reward will be lower than intended due to the incorrect denominator increase.

Recommendation

We recommend deducting the user deposit in line `425` to calculate the `staked_pt_lp` value correctly.

Status: Resolved

15. Insufficient input validations across contracts

Severity: Minor

The smart contracts within scope lack sufficient validation before saving configuration values. This could lead to issues that disrupt the correct behavior of the protocol and lead to failing transactions.

- In `contracts/periphery/builder_unlock/src/contract.rs:50`:
  - `token` parameter not validated to be correct before saving upon instantiation.
  - `proposed_receiver` parameter should be `None` for new allocations.
- In `contracts/periphery/gov/src/contract.rs:74-88` and `679-764`, the following parameters lack enough validation both upon instantiation and update:
  - `xtoken` may render the contract inoperable if set to an incorrect denom.
  - `emergency_proposal_required_denom`: if set to zero, emergency proposals could be auto-approved. If set to more than 1, it will never be met, and the proposal cannot pass.
- In `contracts/periphery/staking/src/contract.rs:53`:
  - Lack of denom validation in the `token` parameter upon instantiation.
- `contracts/periphery/voting_power_inbound/src/contract.rs:50-58` and `102-120`, upon instantiation and `AllowVotingPowerSource`, lack of validation of the following parameters could lead to IBC features not working:
  - `compatible_vp_outbound_versions`

    - `allowed_connection_ids`
    - `allowed_port_ids`
- `contracts/periphery/voting_power_outbound/src/contract.rs:54-63`, `192-194`, and `205-223`, upon instantiation, `AllowVotingPowerSource` and `UpdateConfig`, lack of validation of the following parameters could lead to IBC features not working:
    - `compatible_vp_outbound_versions`
    - `allowed_connection_ids`
    - `allowed_port_ids`
- `contracts/periphery/vault_incentives_v1/src/contract.rs:33-57` and `469-538` lack validation upon instantiation and `UpdateConfig` could lead to protocol inoperability:
    - Missing denom validation of `lp_token`, `yield_token`, `principal_token`, and `pt_lp_token`.
    - `pt_lp_pool_id` pool is not validated.
    - `yield_token_allocation_min` should be less than `yield_token_allocation_max`.
    - `yield_token_allocation_min`, `yield_token_allocation_max`, `refracted_lp_ratio_delta_lower_bound`, and `pt_lp_ratio_delta_lower_bound`, `yield_token_allocation` should be validated to be less than 1.
    - `yield_token_allocation` should be validated to be less than than `yield_token_allocation_max` and greater than `yield_token_allocation_min`. Additionally, the validation should be performed similarly to how the configuration update works.

Recommendation

We recommend thoroughly validating all the affected parameters.

Status: Partially Resolved

16. Owner is allowed to execute arbitrary `CosmosMsgs`

Severity: Minor

The `staking` contract allows the `owner` to execute arbitrary `CosmosMsgs` through the `ExecuteMsg::OwnerAction` entry point.

Among others, this message can be used to move contract funds to an arbitrary address through a `BankMsg`. In the event of compromised access keys or a malicious insider, this would allow the sweeping of all the contract funds.

Recommendation

We recommend restricting the allowed messages to the minimum subset needed for the protocol operation.

Status: Resolved

17. Executed proposals can be removed

Severity: Minor

In `contracts/periphery/gov/src/contract.rs:641-660`, the `remove_completed_proposal` function allows removing an expired or rejected proposal from the general proposal list. This implies that it is also possible to remove an `Executed` proposal once it has expired, as seen in line `653`.

Recommendation

We recommend disallowing the removal of executed proposals so users can query their content.

Status: Resolved

18. Coin denomination updates lead to locked tokens stuck in the contract

Severity: Minor

The `voting-power-outbound` contract allows the configuration field `voting_power_denom` to be updated through `UpdateConfig` in `contracts/periphery/voting_power_outbound/src/contract.rs:196-198`.

If it gets updated, users who have locked tokens will receive, upon `Unlock`, tokens with the new denom instead of the ones they locked. This may lead to failures of the unlock operation or to other user's tokens being spent, depending on the balance of the contract. The originally locked tokens will be stuck in the contract.

Similarly, the `vault_incentives_v1` contract allows the configuration field `incentive_denom` to be updated through `UpdateConfig` in `contracts/periphery/vault_incentives_v1/src/contract.rs:493-495`. In this case, the update of the token denom causes the inability for users to get incentives.

We classify this issue as minor since it can only be caused by the owner of the contract.

Recommendation

We recommend making `voting_power_denom` and `incentive_denom` not updatable.

Status: Resolved

19. Excessive control over funds

Severity: Minor

The `builder-unlock` contract allows the owner to directly transfer any amount of unallocated tokens to an arbitrary address through the `execute_transfer_unallocated` function in `contracts/periphery/builder_unlock/src/contract.rs:517-562`.

This mechanism bypasses the expected behavior of the contract where the allocations are expected to be made available depending on time. In case of a malicious insider or a compromised owner key, the attacker would be able to immediately cash out all the remaining funds by reducing allocations and then transferring the funds out.

Recommendation

We recommend removing the `execute_transfer_unallocated` functionality and managing the redistribution of unallocated tokens through the `execute_increase_allocation` function.

Status: Acknowledged

20. Single-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

- The current owner proposes a new owner address that is validated and lowercase.
- The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

21. Unused reply ID

Severity: Informational

The `staking` contract defines an `INSTANTIATE_TOKEN_REPLY_ID` constant. However, the `reply` function in `contracts/periphery/staking/src/contract.rs:268` does not check the ID.

Although not a security issue, an unchecked reply ID may be error-prone if additional reply handling becomes necessary in a future implementation.

Recommendation

We recommend performing pattern matching against the received reply ID, throwing an error if it is not the expected one.

Status: Resolved

22. Lack of event attribute emission

Severity: Informational

The contracts within scope rarely add attributes when returning a response, neither at the end of the execution nor during errors. This could negatively impact off-chain services that try to monitor or keep track of the `actions` performed in the protocol.

Recommendation

We recommend adding enough information as attributes to responses so the performed action and outcome are clearly identified.

Status: Acknowledged

23. The “Migrate only if newer” pattern is not followed

Severity: Informational

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

Recommendation

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

Status: Acknowledged

24. Incorrect comments

Severity: Informational

The `builder-unlock` contract includes some comments from Astroport's codebase that have not been updated to reflect the modified functionalities.

- `contracts/periphery/builder_unlock/src/contract.rs:61`: The entry point does not exist in the current implementation.
- `contracts/periphery/builder_unlock/src/contract.rs:454`: ZDC tokens are not received through a `CW20 receive` function.
- `contracts/periphery/gov/src/contract.rs:668, 687`: Mention Astroport `assembly` contract.

Recommendation

We recommend updating these comments.

Status: Resolved

25. Votes will be considered failed if the response contains a data field

Severity: Informational

In `contracts/periphery/voting_power_inbound/src/contract.rs:170-175`, the sub-message will be considered failed if the response contains data. This approach is error-prone since future code changes that set a data field with a successful response would incorrectly cause errors.

For example, if the IBC inbound contract sets the data to success or failure, it is then handled by the IBC outbound contract leading to the above-mentioned error.

Recommendation

We recommend determining whether the response is successful by checking the "result" attribute key value to be either "success" or "error", as seen in `contracts/periphery/gov/src/contract.rs:206` and line `216`.

Status: Acknowledged

26. Excess storage operations performed after removing channels

Severity: Informational

In `contracts/periphery/voting_power_inbound/src/contract.rs:266-270` and

`contracts/periphery/voting_power_outbound/src/contract.rs:346-379`, excess storage operations are performed towards the `FOREIGN_VOTING_POWER_TOTALS`, `VOTING_POWER_SOURCES`, and `VOTING_POWER_OUTBOUND` storage states.

These operations are not needed because they have already been performed in `contracts/periphery/voting_power_inbound/src/contract.rs:142-143` and `contracts/periphery/voting_power_outbound/src/contract.rs:246`.

Recommendation

We recommend removing these excess storage operations to reduce gas consumption.

Status: Resolved

27. Queries return incomplete configuration

Severity: Informational

In `contracts/periphery/voting_power_inbound/src/contract.rs:427-429` and `contracts/periphery/voting_power_outbound/src/contract.rs:486-489`, the `Config` query only returns the contract owner and voting power denom.

The returned information is incomplete because other configurations, such as `compatible_vp_outbound_versions`, `allowed_connection_ids`, and `allowed_port_ids`, are missing.

Recommendation

We recommend returning `compatible_vp_outbound_versions`, `allowed_connection_ids`, and `allowed_port_ids` in both `Config` queries.

Status: Resolved

28. Inefficient execution

Severity: Informational

The `staking` contract's `Leave` entry point in `contracts/periphery/staking/src/contract.rs:149` does not explicitly check whether the sender has enough tokens locked. As a `Burn` message is forwarded, if the amount to be burnt is larger than the locked amount, the execution will eventually fail.

Although not a security issue, additional gas will be spent compared to validating sufficient funds as soon as possible.

Recommendation

We recommend implementing validation of sufficient locked tokens early during the execution to follow the “fail early” approach, which enhances efficiency and reduces gas consumption.

Status: Resolved

Appendix: Test Cases

1. Test case for [“The staking contract is not able to mint or burn coins”]

```
    #[test]
    fn test_mint(){
        let mut mock_deps = zodiac_mocks::mock_dependencies(&[]);
        let mock_env =
zodiac_mocks::mock_env(zodiac_mocks::MockEnvParams::default());
        let mut mock_info = zodiac_mocks::mock_info(&String::from("owner"));
        clean_setup(&mut mock_deps);

        assert_eq!(
            instantiate(mock_deps.as_mut(), mock_env.clone(), mock_info.clone(),
InstantiateMsg{
                owner: mock_info.sender.to_string(),
                deposit_token: String::from("zdc"),
            }).unwrap().messages,
            vec![
                SubMsg::reply_on_success(compose_stargate_msg(
                    &MsgCreateDenom{
                        sender: MOCK_CONTRACT_ADDR.to_string(),
                        subdenom: String::from(TOKEN_SYMBOL),
                    },

String::from("/osmosis.tokenfactory.v1beta1.MsgCreateDenom"),
                ).unwrap(), INSTANTIATE_TOKEN_REPLY_ID)
            ]
        );

        let resp = MsgCreateDenomResponse{
            new_token_denom: String::from(TOKEN_SYMBOL),
        };

        let mut resp_bytes: Vec<u8> = vec![];
        ProstMessage::encode(&resp, &mut resp_bytes).unwrap();

        assert_eq!(
            reply(mock_deps.as_mut(), mock_env.clone(), Reply{
                id: INSTANTIATE_TOKEN_REPLY_ID,
                result: SubMsgResult::Ok(SubMsgResponse{
                    events: vec![],
                    data: Some(resp_bytes.into()),
                })
            }).unwrap().messages,
            vec![
```

```
                SubMsg::new(compose_stargate_msg(
                    &MsgSetBeforeSendHook{
                        sender: MOCK_CONTRACT_ADDR.to_string(),
                        denom: String::from(TOKEN_SYMBOL),
                        cosmwasm_address: MOCK_CONTRACT_ADDR.to_string(),
                    },
String::from("/osmosis.tokenfactory.v1beta1.MsgSetBeforeSendHook"),
                ).unwrap())
            ]
        );

        let config_response: ConfigResponse =
from_binary::<ConfigResponse>(&query(mock_deps.as_ref(), mock_env.clone(),
QueryMsg::Config{}).unwrap()).unwrap();

        assert_eq!(
            ConfigResponse{
                token: String::from("zdc"),
                xtoken: String::from(TOKEN_SYMBOL),
                owner: mock_info.sender.to_string(),
            },
            config_response
        );

        //try some owner actions
        execute(mock_deps.as_mut(), mock_env.clone(), mock_info.clone(),
ExecuteMsg::UpdateConfig{
            owner: String::from("new_owner")
        }).unwrap();

        let config: Config = CONFIG.load(&mut mock_deps.storage).unwrap();
        assert_eq!(config.owner, String::from("new_owner"));


        assert_eq!(
            execute(mock_deps.as_mut(), mock_env.clone(),
zodiac_mocks::mock_info(&String::from("new_owner")), ExecuteMsg::OwnerAction{
                msg: CosmosMsg::Bank(BankMsg::Send{
                    to_address: mock_info.sender.to_string(),
                    amount: vec![
                        Coin{
                            denom: String::from("abc"),
                            amount: Uint128::MAX,
                        }
                    ]
                })
            }).unwrap().messages,
            vec![
                SubMsg::new(CosmosMsg::Bank(BankMsg::Send{
```

```
                    to_address: mock_info.sender.to_string(),
                    amount: vec![
                        Coin{
                            denom: String::from("abc"),
                            amount: Uint128::MAX,
                        }
                    ]
                }))
            ]
        );

        //update balances
        mock_deps.querier.base.update_balance(MOCK_CONTRACT_ADDR, vec![Coin{
            denom: String::from("zdc"),
            amount: Uint128::from(10000000u32),
        }]);

        mock_info.funds = vec![
            Coin{
                denom: String::from("zdc"),
                amount: Uint128::from(10000000u32),
            }
        ];


        execute(mock_deps.as_mut(), mock_env.clone(), mock_info.clone(),
ExecuteMsg::Enter{}).unwrap();
        let res = sudo(mock_deps.as_mut(), mock_env.clone(),
SudoMsg::BlockBeforeSend { to: mock_info.sender.to_string(), from:
"tokenfactorymodule".to_string(), amount:
mock_info.funds.first().unwrap().to_owned() });
        assert!(res.is_ok());
    }
```