Audit Report

Sai

v1.1

February 4, 2025

Table of Contents

License

Disclaimer

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 Liquiditea Corp. to perform a security audit of Sai.

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/NibiruChain/SAI
Commit	`6188a3c03b137e103cc86287e810526de1b122df`
Scope	All contracts were in scope.
Fixes verified at commit	`cef86d057d0faa64857931cb50b684f0ca8156d5` 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

Sai is a perpetual futures exchange on the Nibiru blockchain. It enables traders to take leveraged long or short positions on perpetual contracts without expiry dates, using constant-product AMM pools for price discovery.

The platform currently employs an isolated margin model for each trading pair, minimizing cross-position risks while the Perp Fund acts as a backstop to cover funding payments and bad debt, enhancing the platform's resilience during extreme market conditions.

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	Technical documentation was not available.
Test coverage	Medium-High	The test coverage reported by `cargo tarpaulin` is `76.69%`.

Summary of Findings

No	Description	Severity	Status
1	Attackers can double their tokens by using locked deposits	Critical	Resolved
2	Rewards are not distributed when traders increase position size	Critical	Resolved
3	Attackers can abuse the `burn_from` parameter to burn tokens from other users	Critical	Resolved
4	Incorrect day boundary check allows immediate excess point rewards	Critical	Resolved
5	Incorrect sender field set when minting and burning `tokenfactory` denoms	Critical	Resolved
6	Missing market trading validations when updating trades	Major	Resolved
7	Denial of service due to zero-amount fee validation	Major	Resolved
8	Strict timing requirement for collateral redemption	Major	Acknowledged
9	Linear price impact approximation with config parameter as market depth undermines protocol risk management	Major	Partially Resolved
10	Missing cooldown implementation allows unlimited take profit and stop loss updates	Minor	Acknowledged
11	Potential denial of service error due to unbounded `PAIRS` iteration	Minor	Resolved
12	Potential denial of service error due to unbounded `pair_groups` iteration	Minor	Acknowledged
13	Contracts are not compliant with CW2 Migration specification	Minor	Resolved
14	Missing validation in the Perp contract `instantiate` function	Minor	Resolved
15	Missing validation when updating `MAX_DAILY_ACC_PNL_DELTA` and `MAX_SUPPLY_INCREASE_DAILY_P` states	Minor	Resolved
16	Perp contract ownership cannot be transferred	Minor	Resolved

17	Missing validation in the Vault contract `instantiate` function	Minor	Resolved
18	Missing validation in the Oracle contract `instantiate` function	Minor	Resolved
19	Missing validation in the `CreatePermissionGroup` message	Minor	Partially Resolved
20	Missing validation in the `AdminExecuteMsg::SetFees` message	Minor	Resolved
21	Missing validation in the `UpdateExpirationTime` message	Minor	Resolved
22	Missing validation in the `AddMinter` message	Minor	Resolved
23	Group ID sequence is not enforced leading to collisions	Minor	Partially Resolved
24	Incorrect group permissions guard	Minor	Resolved
25	Maximum supply update causes sharp supply increases at day boundary	Minor	Acknowledged
26	Missing validation in `CreatePair` message	Minor	Partially Resolved
27	Centralization risks	Minor	Acknowledged
28	Use of magic numbers decreases maintainability	Informational	Acknowledged
29	Existing `PAIR_NAMES` values can be overwritten	Informational	Acknowledged
30	Events are not emitted	Informational	Acknowledged
31	Miscellaneous comments	Informational	Partially Resolved

Detailed Findings

1. Attackers can double their tokens by using locked deposits

Severity: Critical

In `contracts/vault/src/contract.rs:149`, the `DepositWithDiscountAndLock` function in the Vault contract allows users to lock collateral for a specified period in exchange for premium shares.

However, since shares are minted to the user during the deposit, attackers can exploit this by initiating a `MakeWithdrawRequest` immediately after the `DepositWithDiscountAndLock` execution, allowing them to withdraw their collateral without waiting for the lock period to end.

When the lock expires, users can then use the `UnlockDeposit` function to redeem the shares again, effectively doubling the withdrawal amount.

This leads to unauthorized redemption of shares and poses a risk of significant fund depletion from the vault.

Recommendation

We recommend minting shares only after the lock expiration in the `UnlockDeposit` function.

Status: Resolved

2. Rewards are not distributed when traders increase position size

Severity: Critical

In `contracts/perp/src/update_position_size.rs:157-168`, the `process_opening_fees` function is called to distribute the trigger and staking reward fees as `BankMsg::Send` messages. These messages are returned as part of the function's return values, as seen in `contracts/perp/src/fees/mod.rs:166-191`.

However, the `increase_position_size` function ignores the return values from the `process_opening_fees` function. This causes the fee distribution messages to fail to be dispatched, resulting in a loss of fees for the catalyst and protocol.

Recommendation

We recommend distributing the fee messages returned from the `process_opening_fees` function.

Status: Resolved

3. Attackers can abuse the `burn_from` parameter to burn tokens from other users

Severity: Critical

In `contracts/vault-token-minter/src/contract.rs:149`, the `ExecuteMsg::Burn` message allows the caller to specify the `burn_from` parameter as any address when dispatching `MsgBurn` to the `tokenfactory` module.

This is problematic because the `burn_from` parameter burns tokens from other users' accounts instead of the caller's account.

Consequently, an attacker can exploit this issue to burn tokens from other accounts, such as the `perp` contract, causing a loss of funds.

Recommendation

We recommend setting the `burn_from` parameter value to the `env.contract.address`. Since the caller has already sent the required tokens as validated in `contracts/vault-token-minter/src/contract.rs:134-140`, the tokens can be directly burnt from the contract itself as its balance has already increased.

Status: Resolved

4. Incorrect day boundary check allows immediate excess point rewards

Severity: Critical

In `contracts/perp/src/fees/mod.rs:427-492,558-560`, the `update_trader_points` function checks if `current_day` is greater than `trader_info.last_day_updated` to determine if points should be awarded.

However, the `get_current_day` function divides the `timestamp` seconds by `86400` without considering partial days. This means that a trade held for only one second after a day boundary would be regarded equally to one held for a full day.

This is critical because it allows traders to receive more points than intended by executing high-volume trades around day boundaries, directly impacting the reward distribution system.

Recommendation

We recommend modifying the daily points update logic to compare the actual time difference between updates rather than just the day number.

Status: Resolved

5. Incorrect sender field set when minting and burning `tokenfactory` denoms

Severity: Critical

In `contracts/vault-token-minter/src/contract.rs:113` and `144`, the `tokenfactory` module's `MsgMint` and `MsgBurn` messages are dispatched with the `sender` set as the transaction caller (i.e., `info.sender`). This is incorrect because the message is dispatched with the sender as the contract itself, as seen in lines `126` and `157`.

Consequently, the `ExecuteMsg::Mint` and `ExecuteMsg::Burn` messages will fail, resulting in a denial of service in the protocol.

Recommendation

We recommend setting the `MsgMint` and `MsgBurn` messages' sender field as `env.contract.address`.

Status: Resolved

6. Missing market trading validations when updating trades

Severity: Major

The `TRADING_ACTIVATED` state in `contracts/perp/src/trading/state.rs:41` represents the market state of whether users can initiate trades. For example, when users call the `UpdateOpenLimitOrder` message, the trade can only be updated during the `TradingActivated::Activated` phase, as seen in `contracts/perp/src/trade.rs:167-174`.

However, this is not enforced in the `IncreasePositionSize`, `DecreasePositionSize`, and `UpdateLeverage` messages. If the contract owner closed or paused the `TRADING_ACTIVATED` state, users could still change their trade configurations, which is not intended.

Recommendation

We recommend applying the `TradingActivated.can_open` validation in the `IncreasePositionSize`, `DecreasePositionSize`, and `UpdateLeverage` messages.

Status: Resolved

7. Denial of service due to zero-amount fee validation

Severity: Major

In `contracts/perp/src/trade.rs:756-766`, the `handle_trade_pnl` function dispatches a `BankMsg::Send` message to the trader if the fee amount is larger or equal to zero. This is problematic because if the fee amount is zero, the bank message will fail as Cosmos SDK does not allow sending zero native tokens.

Consequently, the `unregister_trade`, `increase_position_size`, and `decrease_position_size` functions will fail, causing a denial of service for users when interacting with the protocol.

Recommendation

We recommend only sending the fee if the amount is larger than zero.

Status: Resolved

8. Strict timing requirement for collateral redemption

Severity: Major

In `contracts/vault/src/contract.rs:574`, the `redeem` function allows users to retrieve collateral assets.

However, executing this action requires precise timing. Specifically, a user must initially send a `MakeWithdrawRequest` message and then wait 1 to 3 epochs, with each epoch lasting 72 hours, before initiating the redemption process.

The `redeem` function must be called within the first 48 hours of the target epoch, after which the collateral becomes inaccessible.

Should the user miss this restricted time window, they are forced to delete the `withdrawalRequest`, create a new one, and wait an additional 1 to 3 epochs to attempt the redemption again.

Recommendation

We recommend extending the allowable redemption window to include periods beyond the specified target epoch.

Status: Acknowledged

The client states that the protocol ensures fairness and stability by restricting withdrawal requests to the first two days of each epoch and aligning PnL updates with the last day. This prevents LPs from reacting to PnL changes to minimize their risk unfairly, which could compromise the vault's ability to cover trader profits and shift risks to remaining stakers. By timing PnL measurements after the withdrawal window, the protocol ensures withdrawal decisions are independent of the latest data, promoting equitable risk-sharing and vault stability.

9. Linear price impact approximation with config parameter as market depth undermines protocol risk management

Severity: Major

In `contracts/perp/src/trading/state.rs:392-394`, it is checked that `price_impact_p` is larger than `MAX_OPEN_NEGATIVE_PNL_P`.

However, this check relies on the `_get_trade_price_impact` function, which uses a simplistic linear approximation based on a config parameter for the depth/liquidity to calculate price impact.

The documentation already acknowledges several limitations of this approach and suggests future improvements, including:

- A dynamic adjustment of market depth based on recent trading activity.
- Support for asymmetric price impact
- More sophisticated time-weighting algorithms for open-interest calculation.

This is major because oversimplified price impact calculations could lead to the failure of the `MAX_OPEN_NEGATIVE_PNL_P` check, undermining the protocol's risk management.

Recommendation

We recommend implementing all the listed measures. At the minimum, the market depth should not be a linear function with only one config parameter.

Status: Partially Resolved

10. Missing cooldown implementation allows unlimited take profit and stop loss updates

Severity: Minor

In the documentation, it states that take profit (`tp`) and stop loss (`sl`) values can be modified after opening trade with a cooldown period implemented via `tp_last_updated_block` and `sl_last_updated_block`.

However, in the code implementation, these cooldown blocks are not used to restrict updates to `tp` and `sl` values. This allows traders to update their take profit and stop loss values without any time restrictions between updates.

While dynamic limit updates are widespread in traditional finance, cooldown periods are recommended in blockchain environments to mitigate front-running risks and reduce unnecessary triggering costs for the protocol due to frequent updates. A dynamic stop loss could be implemented by setting the limits as a percentage (or weighted average) of a price provided by an oracle.

Recommendation

We recommend implementing the cooldown mechanism by adding validation to check the number of blocks that have passed since the last update.

Status: Acknowledged

11. Potential denial of service error due to unbounded `PAIRS` iteration

Severity: Minor

In `contracts/perp/src/price_impact/mod.rs:61-71`, the `transfer_price_impact_oi_for_pairs` function iterates over all the pairs retrieved from `contracts/perp/src/contract.rs:296-307`. This is problematic because if too many `PAIRS` are configured, the transaction will fail due to an out-of-gas error.

Furthermore, a permanent denial of service may occur since there is no entry point to remove existing `PAIRS` from the storage.

We classify this issue as minor because only the contract owner can configure the `PAIRS` state via the `AdminExecuteMsg::SetPairs` entry point, which is a privileged address.

Recommendation

We recommend implementing a privileged entry point to remove existing `PAIRS` from the storage state or a maximum number of `PAIRS` that can be configured.

Status: Resolved

12. Potential denial of service error due to unbounded `pair_groups` iteration

Severity: Minor

In `contracts/perp/src/borrowing/mod.rs:108`, the iteration over all the `pair_groups` might fail due to an out-of-gas error if too many `pair_groups` are configured.

Furthermore, a permanent denial of service may occur since there is no entry point to remove existing `PAIR_GROUPS` from the storage.

We classify this issue as minor because only the contract owner can configure the `PAIR_GROUPS` state.

Recommendation

We recommend implementing a privileged entry point to remove existing `PAIR_GROUPS` from the storage state or a maximum number of `PAIR_GROUPS` that can be configured.

Status: Acknowledged

13. Contracts are not compliant with CW2 Migration specification

Severity: Minor

The following contracts do not adhere to the CW2 Migration specification standard:

- Perp
- Vault Token Minter
- Vault

This may lead to unexpected problems during contract migration and code version handling.

Recommendation

We recommend following the CW2 standard in all the contracts. For reference, see https://docs.cosmwasm.com/core/entrypoints/migrate#example.

Status: Resolved

14. Missing validation in the Perp contract `instantiate` function

Severity: Minor

During the execution of the `instantiate` function of the Perp contract, defined in `contracts/perp/src/contract.rs:60-81`, the contract fails to validate input addresses for `owner`, `oracle_address`, and `staking_address`.

These addresses are stored directly without checks, which could allow arbitrary strings or incorrect addresses to be registered without verification.

Recommendation

We recommend implementing strict input validation for addresses within the `instantiate` function.

Status: Resolved

15. Missing validation when updating `MAX_DAILY_ACC_PNL_DELTA` and `MAX_SUPPLY_INCREASE_DAILY_P` states

Severity: Minor

In `contracts/vault/src/contract.rs:1487-1516`, the `update_max_daily_acc_pnl_delta` and `update_max_supply_increase_daily_p` function allows the manager to update the `MAX_DAILY_ACC_PNL_DELTA` and `MAX_SUPPLY_INCREASE_DAILY_P` states.

However, no validation ensures that the `new_max_daily_acc_pnl_delta` is not less than `crate::constants::MIN_DAILY_ACC_PNL_DELTA` or the `new_max_supply_increase_daily_p` is not larger than `crate::constants::MAX_SUPPLY_INCREASE_DAILY_P`.

Consequently, misconfiguring these values may cause unintended transaction failures and incorrect computation of `CURRENT_MAX_SUPPLY`.

We classify this issue as minor because only the manager can configure these values, which is a privileged address.

Recommendation

We recommend applying the validations similar to the contract instantiation phase in `contracts/vault/src/contract.rs:57` and `60-61`.

Status: Resolved

16. Perp contract ownership cannot be transferred

Severity: Minor

The `perp` contract does not implement any entry points to transfer the contract ownership. This is problematic because if the current contract owner is compromised, it is impossible to update the owner to a different address to minimize the attack surface.

Additionally, this is inconsistent with the Oracle, Vault, and Vault Token Minter contracts, where the ownership can be transferred via the `UpdateOwnership` message.

Recommendation

We recommend implementing the `UpdateOwnership` feature in the `perp` contract.

Status: Resolved

17. Missing validation in the Vault contract `instantiate` function

Severity: Minor

In `contracts/vault/src/contract.rs:49`, the `instantiate` function of the Vault contract lacks comprehensive validation checks.

Addresses, including `owner`, `manager`, `admin`, `perp_contract`, `vault_token_minter_contract`, and `oracle`, are not validated, which may lead to incorrect or unintended configurations.

Additionally, `min_lock_duration` should be constrained to values greater than 0 and less than `MAX_LOCK_DURATION`.

Similarly, the `withdraw_lock_thresholds_percentage` parameter also requires validation to ensure it falls within the valid range of `[0,1]`.

Recommendation

We recommend implementing strict validation for all specified address parameters within the instantiate function to ensure they conform to expected address formats and prevent misconfigurations.

Status: Resolved

18. Missing validation in the Oracle contract `instantiate` function

Severity: Minor

In `contracts/oracle/src/contract.rs:31`, the `instantiate` function within the Oracle contract lacks validation for the `owner` address, which could lead to improper assignment or functionality issues.

Additionally, the `expiration_time` parameter is not validated to ensure it is greater than zero and set within a reasonable interval, potentially resulting in unpredictable behavior if set incorrectly.

Recommendation

We recommend implementing strict validation for all specified address parameters within the instantiate function to ensure they conform to expected address formats and prevent misconfigurations.

Status: Resolved

19. Missing validation in the `CreatePermissionGroup` message

Severity: Minor

In `contracts/oracle/src/contract.rs:142-151`, the `CreatePermissionGroup` message, provided addresses are stored directly in the contract without validation, which may lead to invalid or duplicate entries.

This lack of validation increases the risk of configuration errors and potential misuse of permission groups.

Recommendation

We recommend implementing validation to ensure all addresses in the vector are correctly formatted and unique.

Status: Partially Resolved

20. Missing validation in the `AdminExecuteMsg::SetFees` message

Severity: Minor

In `contracts/perp/src/contract.rs:224-228`, the `AdminExecuteMsg::SetFees` message, provided fees are stored directly in the contract without validation, which may lead to invalid entries.

Recommendation

We recommend implementing validation to all fees are within reasonable bounds typically, between zero and one.

Status: Resolved

21. Missing validation in the `UpdateExpirationTime` message

Severity: Minor

In `contracts/oracle/src/contract.rs:130-134`, the `UpdateExpirationTime` message handler lacks validation for the `expiration_time` parameter.

Without proper validation, `expiration_time` could be set to zero or an excessively large value, potentially leading to unintended contract behavior.

Recommendation

We recommend implementing validating `expiration_time` to be in a reasonable interval.

Status: Resolved

22. Missing validation in the `AddMinter` message

Severity: Minor

In `contracts/oracle/src/contract.rs:130-134`, in the `AddMinter` message handler, the provided address is added directly to the `WHITELISTED_MINTER` list without prior validation.

This lack of validation could lead to invalid or unintended entries in the whitelist.

Recommendation

We recommend implementing address validation to ensure that only correctly formatted addresses are added to the `WHITELISTED_MINTER` list, preserving the integrity of the whitelist.

Status: Resolved

23. Group ID sequence is not enforced leading to collisions

Severity: Minor

In `contracts/oracle/src/contract.rs:152-159`, the `UpdatePermissionGroup` message in the Oracle contract allows the owner to modify permissions for a specified `group_id`.

However, there is no check to verify if the `group_id` is already registered.

As a result, this function could unintentionally create a new permission group with an arbitrary `group_id`, potentially leading to collisions with the `PERMISSION_GROUP_ID_SEQ` and disrupting the integrity of the permissions structure.

Additionally, the `new_addresses` field is not validated, allowing for potential invalid entries or duplicate addresses within the group.

Recommendation

We recommend implementing validation to confirm that `group_id` exists before updating permissions, thus avoiding the accidental creation of new groups and potential ID collisions.

Additionally, validate and deduplicate all entries in `new_addresses`.

Status: Partially Resolved

24. Incorrect group permissions guard

Severity: Minor

In the Oracle contract, specifically in In `contracts/oracle/src/contract.rs:60-67`, the `SetPrice` and `CreatePair` messages within `OraclesExecuteMsg` are intended to be executable by either the owner or participants of specified permission groups.

However, the `is_authorized` function is incorrectly invoked with group ID `0`, which is restricted solely to the owner.

This oversight prevents participants of other permission groups from executing these messages, effectively disabling the contract's functionality for creating and managing permissioned groups.

Recommendation

We recommend modifying the is_authorized function call to check against the appropriate permission group for each message, ensuring that both the owner and group participants are correctly authorized.

Status: Resolved

25. Maximum supply update causes sharp supply increases at day boundary

Severity: Minor

In `contracts/vault/src/contract.rs:407-428`, the `try_update_current_max_supply` function updates the maximum supply once per day.

The daily increase is applied immediately, causing a sharp jump in the maximum supply at the day boundary.

This is minor because while it creates volatile supply dynamics, it does not compromise the protocol's security. The sudden increase in supply at day boundaries could lead to temporary market inefficiencies and affect user behavior and trading mechanics around these known supply expansion times.

Recommendation

We recommend implementing a more granular supply increase mechanism by reducing the update interval from daily to hourly or per block.

Status: Acknowledged

26. Missing validation in `CreatePair` message

Severity: Minor

In `contracts/oracle/src/contract.rs:87-118`, the `CreatePair` message in the Oracle contract has several validation gaps that can lead to potential inconsistencies and misuse.

Firstly, the `name`, `base`, and `quote` strings are not validated for length, which may allow excessively long values and could impact storage efficiency.

Additionally, the `permission_group` assigned to `PairInfo` is not verified to ensure it exists, potentially resulting in permission errors.

Furthermore, the contract allows multiple `PairInfo` entries with the same `base` and `quote`, and duplicate entries in `PAIR_NAMES` can overwrite existing records, risking unintended overwrites and potential disruption of existing `PairInfo` data.

Recommendation

We recommend implementing the following validations:

- Enforce a maximum length for `name`, `base`, and `quote` strings to ensure data integrity.
- Verify that the specified `permission_group` exists before assigning it to `PairInfo`.
- Prevent duplicate `PairInfo` entries with the same base and quote to avoid redundancy.
- Safeguard `PAIR_NAMES` against overwrites by checking for existing entries before creating new pairs with the same name.

Status: Partially Resolved

27. Centralization risks

Severity: Minor

The smart contracts in scope are designed to rely on a trusted party to perform privileged operations such as setting the Oracle price or withdrawing all assets from the Vault.

Consequently, the overall security of the system depends on the trusted parties, particularly in relation to key management and operations.

If the key management system responsible for authorizing these operations is compromised or mismanaged, several risks could emerge:

- Loss of funds, the `ExecuteMsg::WithdrawCoins` and `AdminExecuteMsg::WithdrawFunds` allow the admin to deplete the funds of the Vault and Perp contracts.

- Price manipulation, the admin can set arbitrary prices in the Oracle contract potentially leading to financial imbalance or unfair trading conditions.

Recommendation

We recommend enforcing strict key management as well as evaluating the removal of the aforementioned privileged operations.

Status: Acknowledged

28. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `contracts/perp/src/price_impact/state.rs:27-29`
- `contracts/oracle/src/contract.rs:274`
- `contracts/oracle/src/contract.rs:242`
- `contracts/vault/src/contract.rs:587`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

29. Existing `PAIR_NAMES` values can be overwritten

Severity: Informational

In `contracts/oracle/src/contract.rs:112`, the `OraclesExecuteMsg::CreatePair` message saves the `name` key with the pair ID value to the `PAIR_NAMES` state. However, it does not validate that the key does not hold any existing value before overwriting it.

Although we did not find a security vulnerability introduced by this issue, overwriting the `PAIR_NAMES` state could have unforeseen consequences when third parties query `OracleQueryMsg::GetPairByName`.

Recommendation

We recommend validating that the state key is empty before updating the `PAIR_NAMES` state.

Status: Acknowledged

30. Events are not emitted

Severity: Informational

In `contracts/perp/src/trade.rs:413-427`, the events mutated inside the `_close_trade` function are not included in the `unregister_trade` function's response.

Similarly, the `store_trade` function in `contracts/perp/src/trade.rs:700-709` returns a response with events and attributes (see `contracts/perp/src/trade.rs:503-505`). However, it is not included in the `register_trade` function's response, causing the events to be ignored.

Recommendation

We recommend emitting the events with the `add_events` function in the return statements.

Status: Acknowledged

31. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

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

- The Vault Token Minter contract stores the total supply of the token in `TOTAL_SUPPLY`. However, since it leverages the TokenFactory module to manage tokens, it could directly use a query to Bank to retrieve the total supply without storing redundant information. We recommend not storing redundant data that can be fetched by native Cosmos SDK modules.

- In `contracts/vault-token-minter/src/contract.rs:50-56` and `contracts/vault-token-minter/src/contract.rs:73-81`, the

`WHITELISTED_MINTER` is loaded two times from the storage. We recommend loading it only one time to reduce gas usage.

- In `contracts/perp/src/trade.rs:64-93`, the `validate` method is executed twice for `trade`. We recommend executing it only once to reduce gas usage.

- In `contracts/vault/src/contract.rs:743`, the `collateralization_percentage` function performs an unnecessary computation by executing the subtraction between the same value causing the formula to always return `1 + ACC_REWARDS_PER_TOKEN` in case `acc_pnl_per_token_used` is negative.

Status: Partially Resolved