Audit Report

Membrane

v1.0

June 15, 2023

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 Membrane.

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/MembraneFinance/membrane-core
Commit	`7535166bfe3370e3ab6b6c38f9a886d8ac2262cc`
Scope	Only the following contracts were in scope: - `contracts/auction` - `contracts/cdp` - `contracts/liq-queue` - `contracts/liquidity_check` - `contracts/oracle` - `contracts/stability-pool`

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

Membrane is a cross-collateral debt protocol built on Cosmos, utilizing the floating-peg stablecoin $CDT and collateralized debt positions.

Pre-determined assets can be "bundled" in a single position, treating their value as a single unit proportional to the respective value and parameters of each underlying asset. This enables vaults to hedge liquidation risk via bundling with less volatile or uncorrelated assets.

To address insolvent debt positions, Membrane employs a three-tiered liquidation mechanism consisting of the Liquidation Queue (LQ), Stability Pool (SP), and open market sales.

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	Certain protocol actions necessitate multiple interactions among multiple contracts within the system, which makes the execution flow difficult to reason about.
Code readability and clarity	Medium	-
Level of documentation	Medium-High	-
Test coverage	Medium-High	`cargo tarpaulin` reports a test coverage for the contracts in scope of 77.62% (4549/5860 lines covered).

Summary of Findings

No	Description	Severity	Status
1	Inhomogeneous time scales used in Osmosis queries lead the protocol to perform calculations on incorrect asset prices	Critical	Resolved
2	Unbounded auction discount increase leads to underflow	Critical	Resolved
3	Parsing error of leftover stability pool repayment amount from the `leftover_repayment` event attribute prevents liquidations	Critical	Resolved
4	Erroneous system time call in the oracle contract prevents the protocol from fetching asset prices	Critical	Resolved
5	Accruing incentives for unstaked stability pool deposits incorrectly sets `last_accrued` and leads to the inability to withdraw, distribute funds, and repay positions	Critical	Resolved
6	Malicious actors could break the liquidation system by spamming bids in the liquidation queue	Critical	Resolved
7	Excessive small stability pool deposits can grief the functionality of the pool	Critical	Resolved
8	Malicious actors can freeze targeted user funds in the contract	Critical	Resolved
9	Users are not able to withdraw their assets or be liquidated if their position contains an asset that has been removed from the oracle	Major	Resolved
10	Auctions are everlasting, and assets that are later added to the auction have accumulated discount applied	Major	Resolved
11	Underflow prevents edit of a collateral asset's maximum LTV parameter	Major	Resolved
12	Failing to distribute debt tokens to stakers	Major	Resolved
13	Lowering an asset's maximum LTV parameter raises the liquidation queue maximum premium without adding additional slots, leading to locked user funds	Major	Resolved

14	Unclaimed stability pool incentives can result in duplicated assets and render the claim function unusable	Major	Resolved
15	Missing owner's input validation	Minor	Resolved
16	Incorrect variable used in equality comparison	Minor	Resolved
17	Oracle centralization risks	Minor	Resolved
18	Possible oracle price caching issues	Minor	Resolved
19	Updating the position contract address in the liquidation queue contract leads to state inconsistency	Minor	Resolved
20	`AddAsset` transaction silently fails	Minor	Resolved
21	Non-production code prevents contracts from working properly and should be removed from the codebase	Minor	Resolved
22	Restaking unstaked stability pool deposits leads to retroactive incentive accumulation	Minor	Resolved
23	Stability pool incentives may be lost when the accumulated amount is near the maximum limit	Minor	Resolved
24	Outdated and incomplete technical documentation	Minor	Acknowledged
25	Liquidation queue bid amount can overreach the bid threshold and is instantly activated	Minor	Resolved
26	Unenforced maximum incentives limit and improper incentives tracking in stability pool	Minor	Resolved
27	The stableswap multiplier is initialized to scale Osmosis stableswap pool liquidity by 1,000%	Minor	Resolved
28	The `assert_credit_asset` function call is unnecessary	Minor	Resolved
29	Errors during calculating the accumulated interest for a stability pool deposit are silently ignored	Minor	Resolved
30	Protocol fees are not sent to the staking contract when selling liquidated collateral on the market	Minor	Resolved
31	Median price calculation incorrectly assumes that prices are ordered	Minor	Resolved
32	Caller-supplied vectors used in the loops can cause out-of-gas errors	Informational	Resolved

33	Additional funds sent to the contract are lost	Informational	Resolved
34	Contracts should implement a two-step ownership transfer	Informational	Resolved
35	Misleading error message while trying to send and repay at the same time	Informational	Resolved
36	Unclear error message during state checking	Informational	Resolved
37	Use of magic numbers decreases maintainability	Informational	Resolved
38	`bigint` crate is affected by CVE-2020-35880	Informational	Resolved
39	Outdated `osmosis-std` dependency	Informational	Resolved

Detailed Findings

1. Inhomogeneous time scales used in Osmosis queries lead the protocol to perform calculations on incorrect asset prices

Severity: Critical

The `get_asset_price` function defined in `contracts/oracle/src/contracts.rs:291` gets asset prices by performing a query on the `GeometricTwapToNowRequest` on Osmosis.

However, since the contract uses timestamps scaled to milliseconds and the `start_time` query parameter requires seconds, the query returns incorrect prices.

Consequently, all the protocol operations performed for an asset price would return wrong results, leading to unexpected behaviors such as a loss of user funds or protocol insolvency.

Recommendation

We recommend passing the timestamp scaled to seconds to the `GeometricTwapToNowRequest` query.

Status: Resolved

2. Unbounded auction discount increase leads to underflow

Severity: Critical

The `get_discount_ratio` function in `contracts/auction/src/contracts.rs:418` determines the discount ratio based on the time elapsed since the auction's initiation and an initial discount specified in `config.initial_discount`.

The discount ratio is calculated by subtracting the sum of the current discount increase (`current_discount_increase`) and the initial discount from 1 (i.e., 100%). `current_discount_increase` increases linearly over time, based on the duration since the auction's start.

However, the `current_discount_increase` is unbounded and increases indefinitely, causing an underflow error in line 434.

Assuming a `config.initial_discount` of 1%, `config.discount_increase_timeframe` set to 60 seconds, and `config.discount_increase` of 1%, the discount becomes 2% after 60 seconds, 3% after

120 seconds, and so on. After 100 minutes in this example, it surpasses 100%, and an underflow error occurs, disrupting the auction.

Consequently, a recently started auction will stop working and necessitate manual intervention by the contract owner to remove the malfunctioning auction. This manual removal will result in an inconsistent state and potentially creates bad protocol debt.

Recommendation

We recommend adding a maximum discount rate to prevent the sum of `current_discount_increase + config.initial_discount` from surpassing 100%.

Status: Resolved

3. Parsing error of leftover stability pool repayment amount from the `leftover_repayment` event attribute prevents liquidations

Severity: Critical

Liquidating an insolvent position utilizes the stability pool as a second line of defense as part of Membrane's three-layer liquidation mechanism. Specifically, the stability pool repays the debt and delivers the leftover repayment amount to the `cdp` contract as part of the `leftover_repayment` event attribute in `contracts/stability-pool/src/contract.rs:621-624`. The value for the `leftover_repayment` event attribute is composed as `"{leftover} {asset_pool.credit_asset.info}"`.

The `handle_stability_pool_reply` reply handler function in the `cdp` contract, located in `contracts/cdp/src/reply.rs`, extracts the leftover repayment amount as a `Uint128` value from the event string attribute. However, the `Uint128::from_str` function in line `252` is unable to parse the value due to the appended `credit_asset.info` string. Consequently, the reply handler errors and the liquidation attempt is unsuccessful.

Recommendation

We recommend removing the credit asset's info (`credit_asset.info`) from the `leftover_repayment` event attribute.

Status: Resolved

4. Erroneous system time call in the oracle contract prevents the protocol from fetching asset prices

Severity: Critical

The `oracle` contract determines the price of an asset by querying the geometric time-weighted average price (TWAP) price in the `get_asset_price` function in `contracts/oracle/src/contracts.rs:291`. Within this function,

`SystemTime::now()` is utilized to define the TWAP's start time in line `318`.

Since system time introduces a non-deterministic factor because it can vary among nodes in the network, it is not supported by Wasmer and returns the error "`time not implemented on this platform`".

Consequently, the protocol is not able to fetch asset prices.

Recommendation

We recommend using block time (`env.block.time`) instead of system time.

Status: Resolved

5. Accruing incentives for unstaked stability pool deposits incorrectly sets `last_accrued` and leads to the inability to withdraw, distribute funds, and repay positions

Severity: Critical

When withdrawing tokens from stability pool deposits, the `accrue_incentives` function in `contracts/stability-pool/src/contract.rs:221` calculates and accrues incentives for each user deposit iteratively. The earned incentives are based on the elapsed time from either the current time or the unstake time, depending on whether the deposit is unstaked. Subsequently, the deposit's `last_accrued` timestamp is updated to the current block time.

However, assigning the current block time to the `last_accrued` timestamp for unstaked deposits results in an underflow error in line 231 of the `accrue_incentives` function during subsequent accruals. This error is caused by subtracting the `last_accrued` value from the smaller `unstake_time` value.

Consequently, calls related to withdrawing deposits, distributing funds from liquidations, and repaying positions will revert.

The same problem also occurs in the `get_user_incentives` function, specifically in line 1172.

Recommendation

We recommend adjusting the `last_accrued` value of unstaked deposits to correspond to the time of the unstaking operation (`unstake_time`) in both the `accrue_incentives` and `get_user_incentives` functions.

Status: Resolved

6. Malicious actors could break the liquidation system by spamming bids in the liquidation queue

Severity: Critical

The liquidation queue contract enables users to place bids in slots for each asset.

Because of the use of vectors for slots and bids, an iteration over them is required in order to perform operations.

Since a maximum number of bids for a slot is not enforced, a malicious actor could spam a large number of small bids to cause the iteration to run out of gas.

One of the iterations that could be targeted can be found in the `read_bids_by_user` function defined in `contracts/liq-queue/src/bid.rs:1035`, which is extensively used throughout the codebase.

Affected messages are `Liquidate`, `ClaimLiquidations`, and `RetractBid`.

This issue could permanently disable liquidations that involve the liquidation queue and hence poses a threat to the health of the protocol.

Recommendation

We recommend enforcing a maximum number of bids that can be registered in slots or implementing a different data structure that does not require performing unbounded iterations.

Status: Resolved

7. Excessive small stability pool deposits can disable the functionality of the pool

Severity: Critical

Depositing debt tokens in the stability pool with the `deposit` function in `contracts/stability-pool/src/contract.rs:181` creates separate deposit entries for each deposit, which are then stored in the `AssetPool.deposits` vector.

Whenever deposits for a specific user are processed, all deposits are iterated and checked if they belong to the user. This inefficient data structure can be exploited by an adversary who creates numerous small deposits, thus increasing the number of iterations needed to process deposits for a particular user. Such action can lead to the exhaustion of the gas limit for a transaction, consequently disabling the stability pool.

Recommendation

We recommend adopting a more efficient data structure that eliminates the need to iterate over all deposits. An alternative approach could involve requiring a minimum amount of debt tokens for deposits and limiting the maximum number of deposits per user.

Status: Resolved

8. Malicious actors can freeze targeted user funds in the contract

Severity: Critical

The `deposit` function in `contracts/cdp/src/positions.rs:45` allows anyone to create a new position on behalf of another user.

This functionality could be exploited by a malicious actor by adding a large number of small value positions for a targeted user.

Since positions are stored in a vector and unbounded iterations are performed on it, a large number of positions will lead to out-of-gas errors.

Also since vectors need to be loaded to memory for iteration, usage of them might cause the node to run out of memory.

Consequently, the targeted user will not be able to withdraw their funds, which will be stuck in the contract.

Recommendation

We recommend adopting a more efficient data structure that eliminates the need to iterate over positions. An alternative could be enforcing a maximum number of positions per user.

Status: Resolved

9. Users are not able to withdraw their assets or be liquidated if their position contains an asset that has been removed from the oracle

Severity: Major

The `get_asset_values` function defined in `contracts/cdp/src/query.rs:515`, interacts with the price oracle contract in order to get asset prices by executing the `Price` query.

Since this query returns an error if one of the requested assets has been deleted from the oracle contract, the entire transaction will revert.

Consequently, accounts that contain an asset that has been removed from the oracle contract's allowlist, cannot withdraw collateral or be liquidated because the `Price` query returns an error and would make the transaction revert.

We classify this issue as major instead of critical since assets can only be removed from the oracle by the owner.

Recommendation

We recommend handling the error or not returning an error when querying the price of removed assets.

Status: Resolved

10. Auctions are everlasting, and assets that are later added to the auction have accumulated discount applied

Severity: Major

In the process of auctioning off bad debt, discounts on assets under auction are initially set to the `config.initial_discount` value and linearly increase over time based on the time elapsed since the auction started.

The `start_auction` function, found in `contracts/auction/src/contracts.rs:140`, initiates a new auction or incorporates assets into an existing one. For newly created auctions, the `start_time` is assigned the current block timestamp, as indicated in lines 181 and 235.

However, auctions that are fulfilled are never removed. Consequently, the auction `start_time` is never updated, and subsequently added assets to the auction will have the accumulated discount applied instead of starting from the initial discount.

Recommendation

We recommend creating separate auctions for each bad debt position and automatically removing auctions once they are fulfilled.

Status: Resolved

11. Underflow prevents edit of a collateral asset's maximum LTV parameter

Severity: Major

The owner-invokable `edit_cAsset` function in `contracts/cdp/src/contract.rs:266` enables the owner to modify parameters for a

collateral asset. Among the adjustable parameters is `max_LTV`, representing the maximum loan-to-value ratio for the asset.

In line `306`, the maximum premium `max_premium` of the asset's liquidation queue is updated. However, the calculation for this new `max_premium` value is incorrect, leading to an underflow error. This is caused by subtracting the `LTV` value (expressed as an integer of atomic units) from `Uint128::new(95u128)`.

Recommendation

We recommend using `Uint128::new(95u128).atomics()` for the left-hand side of the subtraction.

Status: Resolved

12. Failing to distribute debt tokens to stakers

Severity: Major

When repaying debt, repaid debt tokens are either entirely burned or distributed to stakers through the `credit_burn_rev_msg` function in `contracts/cdp/src/positions.rs:1878`.

In cases where the number of burned tokens is zero, a corresponding `OsmoExecuteMsg::BurnTokens` message is still created and dispatched to Membrane's Osmosis proxy contract. However, the `burn_tokens` function in the Osmosis proxy contract reverts in `contracts/osmosis-proxy/src/contract.rs:504` if the number of tokens to burn is zero, which prevents the distribution of debt tokens to stakers.

Recommendation

We recommend including a check to ensure the number of tokens to burn is non-zero before creating the `OsmoExecuteMsg::BurnTokens` message.

Status: Resolved

13. Lowering an asset's maximum LTV parameter raises the liquidation queue maximum premium without adding additional slots, leading to locked user funds

Severity: Major

When the owner of the `cdp` contract modifies an asset's maximum loan-to-value (LTV) ratio `max_LTV` to a lower value, the maximum premium for the asset's liquidation queue is increased. However, the `edit_queue` function in the `liq-queue` contract, found in

`contracts/liq-queue/src/contract.rs:197`, does not create extra slots in the liquidation queue to accommodate the new maximum premium.

Bids users add for the heightened premium will be accepted, but they will be ignored by the `execute_liquidation` function in `contracts/liq-queue/src/bid.rs:337` and skipped instead. Due to the `read_premium_slot` function returning an error for a non-existent slot, bids for those slots cannot be withdrawn anymore. This leads to the user's funds becoming locked and unused in the liquidation queue.

We classify this issue as major instead of critical since it requires lowering an asset's maximum LTV parameter which is an update only the owner is able to do.

Recommendation

We recommend adding additional slots to the liquidation queue when the maximum premium is increased.

Status: Resolved

14. Unclaimed stability pool incentives can result in duplicated assets and render the claim function unusable

Severity: Major

Over time, stability pool deposits accrue incentives. When users withdraw deposits, the accumulated incentives are computed and added to the users' claimable assets.

However, the `withdrawal_from_state` function appends the native token with the `config.mbrn_denom` denom in `contracts/stability-pool/src/contract.rs:489` to the user's claimable assets `claimable_assets` without checking for the token's presence in the claimable assets.

Consequently, the `claimable_assets` vector may contain duplicated assets, causing the claim function to revert due to Osmosis's version of Cosmos SDK's `Coins` type prohibiting duplicate denoms.

Recommendation

We recommend using the `cw-coins` crate to manage coin collections, which prevents duplicate denoms.

Status: Resolved

15. Missing owner's input validation

Severity: Minor

Contracts are missing validation on some owner's input:

- `liq_fee` in `contracts/cdp/src/contract.rs:54` and `packages/membrane/src/cdp.rs:331` should be validated to be in the `[0,1]` range.
- `cpc_multiplier` and `rate_slope_multiplier` in `packages/membrane/src/cdp.rs:348` should be validated to be in a predefined range.
- `max_premium` and `bid_threshold` in `contracts/liq-queue/src/contract.rs:213` should be validated to be in a predefined range.
- `max_LTV` in `contracts/cdp/src/contract.rs:297` should be validated to be in the `(0,100]` range.
- `twap_timeframe`, `initial_discount`, `discount_increase_timeframe` and `discount_increase` in `contracts/auction/src/contract.rs:118` should be validated to be in a predefined range.
- `stableswap_multiplier` in `contracts/liquidity_check/src/contract.rs:203` should be validated to be in a predefined range.
- `incentive_rate`, `max_incentives`, and `unstaking_period` in `contracts/stability-pool/src/contract.rs:161` should be validated to be in a predefined range.

Recommendation

We recommend validating inputs before storing them in contracts.

Status: Resolved

16. Incorrect variable used in equality comparison

Severity: Minor

The `validate_bid_input` function in `contracts/liq-queue/src/bid.rs:1069` contains an erroneous comparison, where the value of `queue.bid_asset.info` is mistakenly compared with itself.

This means that this part of the condition will always return `True`, regardless of the `bid_input` argument sent to the function.

Despite the incorrect comparison, the current implementation does not lead to any immediate issues as the queue for the auction asset is correctly loaded by the `bid_input.bid_for` key.

Recommendation

We suggest replacing `queue.bid_asset.info` with `bid_input.bid_for` in one of the equality comparison sides.

Status: Resolved

17. Oracle centralization risks

Severity: Minor

In the current design, the owner is able to post arbitrary prices to the oracle. Prices are not validated, and any value is accepted.

A compromised owner account or bot may lead to price manipulation exploits, for example, by setting the price of all assets to 0, which would allow the attacker to liquidate all users at their loss.

Moreover, the owner itself could set the wrong prices by mistake.

Recommendation

We recommend performing validation on the posted prices. For instance, there could be a maximum allowed delta per time unit, such that a price of 0 would not be accepted. While this does not fully resolve the centralization issue, privilege abuse would be more involved (and require multiple transactions over a longer time span). This would allow operators and users to react and take counter-measures.

Status: Resolved

18. Possible oracle price caching issues

Severity: Minor

The oracle contract caches asset prices in order to execute queries more efficiently. However, since the `oracle_time_limit` parameter can assume every value, it could lead to misconfigurations where the position contract queries outdated prices.

Prices are calculated as TWAPs, so the caching time retention should be calculated based on the TWAP timespan in order to cache meaningful data.

Recommendation

We recommend implementing a mechanism to calculate the `oracle_time_limit` parameter based on `twap_timeframe`.

Status: Resolved

19. Updating the position contract address in the liquidation queue contract leads to state inconsistency

Severity: Minor

In `contracts/liq-queue/src/contract.rs:168`, the liquidation queue owner is allowed to update the `position_contract` address, and consequently the `bid_asset`.

However, it is not handling the current bids placed in the contract and their relative fund deposits, which would lead to a state inconsistency and the freeze of the fund denominated in the previous `bid_asset`.

Recommendation

We recommend allowing the update of the `position_contract` address only with an ad-hoc migration.

Status: Resolved

20. `AddAsset` transaction silently fails

Severity: Minor

The `add_asset` function, defined in `contracts/oracle/src/contracts.rs:149`, silently fails if the asset is already stored in the contract.

Moreover, in `contracts/oracle/src/contracts.rs:210`, the `added` event's attribute is set to false instead of reverting the transaction.

Consequently, a contract that executes this transaction is not able to correctly handle a failure and users might be misled.

Recommendation

We recommend reverting if an asset cannot be added.

Status: Resolved

21. Non-production code prevents contracts from working properly and should be removed from the codebase

Severity: Minor

The codebase contains some non-production code used for testing purposes which prevents the contracts from working properly in the following locations:

- `contracts/liq-queue/src/contract.rs:57`
- `contracts/cdp/src/liquidations.rs:79`
- `contracts/cdp/src/reply.rs:275`

Recommendation

We recommend using Rust features in order to include or exclude code at build time.

Status: Resolved

22. Restaking unstaked stability pool deposits leads to retroactive incentive accumulation

Severity: Minor

Deposits in the stability pool accrue incentives based on the time elapsed since the last incentive accrual. Unstaked deposits intended to be withdrawn must wait for the unstaking period to expire, during which they do not receive incentives.

However, when an unstaked deposit is restaked using the `restake` function in `contracts/stability-pool/src/contract.rs:519`, the deposit's `unstake_time` is reset to `None`. This results in the deposit retroactively accruing incentives since the last accrual, covering the entire unstaking period.

Recommendation

We recommend accruing incentives prior to restaking to ensure the deposit's `last_accrued` timestamp is updated correctly.

Status: Resolved

23. Stability pool incentives may be lost when the accumulated amount is near the maximum limit

Severity: Minor

The stability pool incentives are subject to a global maximum limit, `config.max_incentives`, which is enforced within the `accrue_incentives` function

in `contracts/stability-pool/src/contract.rs:248`. When the sum of the current accumulated incentives `total_incentives` and the incentives to be accrued `incentives` exceeds the maximum limit, the value of `incentives` is set to 0, resulting in the user not being rewarded.

In cases where the current accumulated incentives are slightly below the maximum limit, and the addition of the new incentives exceeds this limit, newly accrued incentives up to the maximum limit are not distributed and are lost.

Recommendation

We recommend accruing the delta of the maximum limit and `total_incentives` instead of setting `incentives` to 0.

Status: Resolved

24. Outdated and incomplete technical documentation

Severity: Minor

The technical documentation of the tested contracts was limited to the descriptions of `ExecuteMsg` messages, including the conditions they should meet and the actions they perform.

However, it was noticed that in several places, this documentation does not coincide with the implementations and line comments of the contracts. Also, there are functionalities that are not documented at all, such as `Accrue`, `SwapWithMBRN`, `EditcAsset`, `MintRevenue`, `SubmitBid`, `RetractBid`, `Liquidate`, `ClaimLiquidations`, `AddQueue`, and `UpdateQueue`.

Inaccurate or incomplete documentation can mislead users and decrease the maintainability of the protocol due to potential misunderstandings of core concepts, business logic, and execution flow.

Recommendation

We suggest keeping detailed technical documentation up to date with the codebase.

Status: Acknowledged

25. Liquidation queue bid amount can overreach the bid threshold and is instantly activated

Severity: Minor

Submitting a bid to a specific slot in a liquidation queue automatically activates the bid if the current sum of all slot bids falls at or below the bid threshold `queue.bid_threshold`. In

contrast, bids exceeding this threshold must wait for a specified duration before being automatically activated and utilized for liquidations.

However, the bid threshold check in the `submit_bid` function in `contracts/liq-queue/src/bid.rs:80` fails to account for the bid amount. This allows a significant bid to be submitted and activated instantaneously if the current slot's `total_bid_amount` is only slightly below the bid threshold. A liquidator can exploit this to bypass the intended waiting period and to take a majority of the liquidated assets by using a large bid amount.

This issue is also present in line `661`.

Recommendation

We recommend ensuring that the sum of `bid.amount` and `slot.total_bid_amount` is less than or equal to `queue.bid_threshold` before activating the bid.

Status: Resolved

26. Unenforced maximum incentives limit and improper incentives tracking in stability pool

Severity: Minor

Claiming stability pool incentives as a user is done through the `claim` function in `contracts/stability-pool/src/contract.rs:902`. This function calculates the user's newly accrued incentives by invoking the `get_user_incentives` function in line `143` and adding the newly accrued incentives to the user's available claims.

The incentives are constrained by a global maximum limit defined in `config.max_incentives`. However, this upper bound is neither assessed nor enforced within the `get_user_incentives` function. Hence, users can claim in excess of the maximum amount of incentives.

Moreover, the newly accrued incentives are not added to the `INCENTIVES` storage variable, which tracks the total accrued incentives.

Recommendation

We recommend asserting that newly accrued incentives are not exceeding the `config.max_incentives` value, as already correctly implemented in the `accrue_incentives` function in line `258`. Additionally, the newly accrued incentives should be tracked in the `INCENTIVES` storage variable.

Status: Resolved

27. The stableswap multiplier is initialized to scale Osmosis stableswap pool liquidity by 1,000%

Severity: Minor

The `liquidity_check` contract evaluates the AMM liquidity of collateral assets. An owner-configurable multiplier `stableswap_multiplier` is used to scale the liquidity of Osmosis stableswap pools. This multiplier is initialized to `Decimal::percent(10_00)` in `contracts/liquidity_check/src/contracts.rs:18`, representing a 1,000% boost to the stableswap pool liquidity. This high boost value is likely unintended and may result in inflated pool liquidity if the owner does not adjust the multiplier after deployment.

Recommendation

We recommend initializing the stableswap multiplier `stableswap_multiplier` to `Decimal::one()` to prevent inflating the stableswap pool liquidity.

Status: Resolved

28. The `assert_credit_asset` function call is unnecessary

Severity: Minor

The `assert_credit_asset` function is used during the repayment of outstanding debt of a position performed in the `repay` function. It is called with two arguments. One of them is the `basket` struct. The second one is the `credit_asset` variable, which is set when `ExecuteMsg::Repay` is called and passed as a function parameter. The `assert_credit_asset` function verifies that the tokens sent with the transaction match the ones in the `basket` configuration.

However, this check is unnecessary. When `ExecuteMsg::Repay` is called, before going to the `repay` function, the sent parameters are validated. During that validation, the `assert_sent_native_token_balance` function is executed, taking `basket.credit_asset.info` and the funds sent as part of the transaction as parameters. It thereby verifies whether the denominations match and whether the amount of coins is different from zero.

Recommendation

We suggest removing the `assert_credit_asset` function call within the `repay` function, and since this is the only usage of this function, it can be removed entirely.

Status: Resolved

29. Errors during calculating the accumulated interest for a stability pool deposit are silently ignored

Severity: Minor

The `get_user_incentives` function in `contracts/stability-pool/src/contract.rs:1141` calculates the user's incentives by adding the accumulated interest on all deposits calculated by the `accumulate_interest` function. Yet, if the `accumulate_interest` function returns an error for a specific deposit, the interest accumulator remains unchanged, and the returned error is assigned to the error variable. Subsequently, this error is neither propagated to the caller nor logged, which prevents error handling and detection.

Recommendation

We recommend ensuring that any errors from the `accumulate_interest` function are propagated to the caller, which allows error handling.

Status: Resolved

30. Protocol fees are not sent to the staking contract when selling liquidated collateral on the market

Severity: Minor

When liquidating insolvent positions, fees for both the protocol and the party initiating the liquidation are deducted from the position's collateral. Corresponding `BankMsg::Send` messages are then crafted for the transfer of these fees.

However, when selling the liquidated collateral on the market in the `liquidate` function in `contracts/cdp/src/liquidations.rs:234`, the protocol fee messages `protocol_fee_msg` are not added to the response, which implies that these fees are not sent to the staking contract.

Recommendation

We recommend adding the protocol fee messages to the response.

Status: Resolved

31. Median price calculation incorrectly assumes that prices are ordered

Severity: Minor

During the handling of the `Price` query in the oracle contract, when multiple prices are fetched, the median price is calculated in `contracts/oracle/src/contracts.rs:380` and returned.

The current implementation assumes that the `oracle_prices` vector is ordered and takes the middle element in order to return the median.

However, since the vector is not ordered, a random price will be returned instead of the median price.

Recommendation

We recommend reworking the median price calculation in order to return the correct value.

Status: Resolved

32. Caller-supplied vectors used in the loops can cause out-of-gas errors

Severity: Informational

For query functions that use the `QueryMsg` type, limits have been implemented that allow queries without exceeding gas limits.

However, it has been noticed that for functions accepting input from the user of the `Vec<>` type, such limits cannot be set, which in turn may lead to a situation where the use of these message becomes impossible because the loop runs out of gas.

This can have particularly dangerous consequences if queries are used as part of other more advanced operations such as depositing, withdrawing, and related operations.

Examples of such functions are:

- The `get_asset_values` function in `contracts/cdp/src/query.rs:533`.
- The `get_asset_prices` function in `contracts/oracle/src/contract.rs:411`.

Recommendation

We suggest introducing limits that will prevent accepting too large objects as arguments of these functions, or introducing maximum iteration values for loops operating on these arguments.

Status: Resolved

33. Additional funds sent to the contract are lost

Severity: Informational

In `contracts/auction/src/contracts.rs:314`, `contracts/liq-queue/src/bid.rs:48`, and `contracts/stability-pool/src/contracts.rs:97`, a check is performed that ensures that in the transaction, there is a `Coin` with the expected `denom` field.

This validation does not ensure however that no other native tokens have been sent, and any such additional native tokens are not returned to the user, so they will be stuck in the contract forever.

While blockchains generally do not protect users from sending funds to wrong accounts, reverting extra funds increases the user experience.

Recommendation

We recommend checking that the transaction contains only the expected `Coin` and no additional native tokens using https://docs.rs/cw-utils/latest/cw_utils/fn.must_pay.html.

Status: Resolved

34. Contracts should implement a two-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 actually execute the config update.

Recommendation

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

1. The current owner proposes a new owner address that is validated and lowercased.

2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

35. Misleading error message while trying to send and repay at the same time

Severity: Informational

The `mint_revenue` function in `contracts/cdp/src/position.rs:1639` performs a check to prevent the two optional variables `send_to` and `repay_for` being specified simultaneously or not specified at all.

However, if `True`, the error message informs about the second scenario only, that is, `"Destination address is required"`. If both `send_to` and `repay_for` are specified, the error message is wrong.

Recommendation

We recommend expanding the error message to cover both cases or using two distinct error messages.

Status: Resolved

36. Unclear error message during state checking

Severity: Informational

The `cdp` contract uses the `check_debt_increase_state` function to validate `credit_amount` after the `increase_debt` operation. If the values are not as expected, the function returns a `ContractError` indicating `"Possible state error"`.

This means that the caller of the function receives a very generic message, not indicating how to resolve the issue.

Recommendation

We suggest expanding the error message to describe the cause of the error to allow the caller to resolve it.

Status: Resolved

37. 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/cdp/src/rates.rs:387`
- `contracts/cdp/src/positions.rs:1251`
- `contracts/cdp/src/contract.rs:66`
- `contracts/cdp/src/contract.rs:68`

Recommendation

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

Status: Resolved

38. `bigint` crate is affected by CVE-2020-35880

Severity: Informational

In `packages/membrane/Cargo.toml:14`, `bigint` is specified as a dependency. As reported in https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-35880 and https://rustsec.org/advisories/RUSTSEC-2020-0025.html that crate is affected by a Critical CVE with a score of 9.8.

The crate is not maintained anymore and contains several known bugs (including a soundness bug).

Recommendation

We recommend following the recommendation in the CVE to substitute `bigint` with https://crates.io/crates/uint.

Status: Resolved

39. Outdated `osmosis-std` dependency

Severity: Informational

In `contracts/cdp/Cargo.toml:29`, `osmosis-std` is required as a dependency with version `0.1.0`.

However, this version is outdated and could be affected by bugs and issues that have been resolved in up-to-date versions.

Recommendation

We recommend updating the `osmosis-std` dependency to the latest version.

Status: Resolved