Review and Audit Report

NAMI Protocol – Rebalancer Contract

Final Findings Report

July 27th, 2024

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

ENTROPIC

DISCLAIMER

This audit report is provided "as is" without any representations or warranties, express or implied. Entropic Labs, LLC and the author(s) of this report make no representations or warranties in relation to this audit report or the information and materials provided herein.

This report does not constitute financial, investment, or legal advice. It is solely created for the use of the client and should not be relied upon by any other parties. The client assumes all risks associated with the use of the audited smart contract.

Entropic Labs, LLC and the author(s) shall not be held liable for any direct, indirect, special, incidental, consequential, or other losses of any kind, in tort, contract, or otherwise (including but not limited to loss of revenue, income or profits, or loss of use or data), arising out of or in connection with any use of or reliance on this audit report.

STRUCTURE

This report is structured in a code review phase followed by a comprehensive audit phase. Each section of this report contains associated information on vulnerabilities or code suggestions. Not all findings from the code review phase may be included in this report, but have been implemented before the audit phase began.

Issues and suggestions are segmented into the following categories:

- **Critical**: An issue that is actively exploitable or present, causing severe losses or a severe denial of service.
- Major: An issue that may manifest itself or be exploitable, causing loss of funds, state inconsistencies, denial of service, or other errors.
- Minor: An issue that may lead to other exploitable vulnerabilities, now or in the future, or a pattern that seriously increases computation costs, or violates best practices.
- Informational (Info): Opinionated changes or improvements to the codebase that the author highly suggests the client consider, which may or may not directly impact contract security.

Whilst audits are an excellent method to uncover issues, they do not guarantee the correctness of a contract. This audit *does not* cover any changes made outside the scope or submission timeline of the codebase.

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

SCOPE OF WORK

This code review and audit report was commissioned by NAMI Protocol, on the codebase of their NAMI Rebalancer smart contract. The code review familiarized the NAMI team with CosmWasm development best practices, whilst a detailed audit helps productionize the NAMI protocol smart contracts.

The code review portion of this report was conducted on the following codebase:

Repository: https://github.com/NAMIProtocol/rebalancer_smart_contract
Revision: `3489210fad4bb05f1c6bd1fffa1ddf137a375f0e` (on branch main)

The audit portion of this report was conducted on the following codebase(s):

Repository: https://github.com/NAMIProtocol/rebalancer_smart_contract
Revision: `ae97ab8c602ee3ab8aa1400119f2f6c401f52568` (on branch main)
Revision: `eff9ee5c1b8ea289ba6d9dd75ce635b6512ef511` (on branch main)

CODEBASE REMARKS

Codebase Complexity: **Medium**
Codebase Test Coverage: 83.09% (TARPAULIN)

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

ENTROPIC

Code Review

SUMMARY

CR-01	Use CosmWasm APIs for Time	**Critical**	**Resolved**
CR-02	Pass Mutable References of State Instead of Loading from Store	**Critical**	**Resolved**
CR-03	Prevent Running Expensive Queries and Calculations Twice	**Major**	**Resolved**
CR-04	Rounding and Decimal Normalization Should Be Done Client-side	**Major**	**Resolved**
CR-05	Use QueryMsg Interfaces Instead of Raw JSON	Minor	**Resolved**
CR-06	Prevent Unnecessary Clones, Reducing Cost	Minor	**Noted**
CR-07	Reduce Use of `.unwrap()`	Minor	**Resolved**
CR-08	Store `receipt_denom` in Config	Minor	**Resolved**
CR-09	Don't Unnecessarily Load Items from Storage Inside Helpers	**Major**	**Resolved**
CR-10	Remove Old Version of `kujira_ghost` and use `kujira::ghost`	Minor	**Resolved**
CR-11	Use Standard Utilities for Constructing Funds	Info	**Resolved**
CR-12	Use Clippy Lints	Info	**Resolved**
CR-13	Use Doc-Comments over Comments	Info	**Resolved**

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

ENTROPIC

Detailed Findings

CR-01 - Use CosmWasm APIs for Time

Critical

Description: WASM code is executed inside a sandbox that has no external interaction by default. This includes CosmWasm. Using `chrono` or other time libraries will not work inside wasm. Instead, use the BFT time provided by Tendermint and proxied by CosmWasm's `env.block.time`.

Recommendation: Use CosmWasm's `env.block.time`.

Resolved

CR-02 - Pass Mutable References of State Instead of Loading from Store

Critical

Description: Certain functions, such as `take_fee`, or `cost_reached` load and modify the contract state, and may persist the modifications by calling `State::save`. However, any functions calling `take_fee` may hold references to contract state that do not update alongside these modifications. This can cause calculations to be done on stale data, leading to major inconsistencies.

Recommendation: Pass mutable references of the state to avoid stale data.

Resolved

CR-03 - Prevent Running Expensive Queries and Calculations Twice

Major

Description: Prevents duplicated calculations by caching results in variables.

Recommendation: Cache results to avoid redundant calculations.

Resolved

CR-04 - Rounding and Decimal Normalization Should Be Done Client-side

Major

Description: The contract should be the most accurate source of truth for all data. Thus, any rounding should be done by frontends or other consuming client-side programs, instead of in the contract.

Recommendation: Perform rounding and normalization client-side.

Resolved

CR-05 - Use QueryMsg Interfaces Instead of Raw JSON

Major

Description: Uses typed interfaces to prevent manual construction of QueryMsgs, giving stronger type safety.

Recommendation: Use typed QueryMsg interfaces.

Resolved

CR-06 - Prevent Unnecessary Clones, Reducing Cost

Minor

Description: Prevent unnecessary clones to reduce cost.

Recommendation: Avoid unnecessary cloning of data.

Noted

CR-07 - Reduce Use of .unwrap()

Minor

Description: Unwrap should be used in instances where an error is not expected, and it's alright for a program to panic with an error. Instead, we should return errors.

Recommendation: Replace `.unwrap()` with proper error handling.

Resolved

CR-08 - Store `receipt_denom` in Config

Minor

Description: The `State` should be used for dynamic state, tracking balances as they change. The `receipt_denom` does not change after contract instantiation, and should be placed in the `Config` struct instead.

Recommendation: Store `receipt_denom` in the `Config` struct.

Resolved

CR-09 - Don't Unnecessarily Load Items from Storage Inside Helpers

Major

Description: Unnecessary storage loads increase the storage costs incurred by the contract.

Recommendation: Avoid unnecessary storage loads in helper functions.

Resolved

CR-10 - Remove Old Version of `kujira_ghost` and Use `kujira::ghost`

Minor

Description: For compatibility reasons, enforces that one version of the kujira crate is used by using the re-exported `kujira::ghost` as opposed to `kujira_ghost` directly.

Recommendation: Use the re-exported `kujira::ghost`.

Resolved

CR-11 - Use Standard Utilities for Constructing Funds

Informational

Description: Fund construction can be simplified by using existing utilities.

Recommendation: Use standard utilities from `cw-utils` for constructing funds.

Resolved

CR-12 - Use Clippy Lints

Informational

Description: `clippy` lints are a more strict set of lints, recommended to improve code quality and find unnecessary calls.

Recommendation: Address `clippy` lint warnings.

Resolved

CR-13 - Doc-Comments Over Comments

Informational

Description: Rust has built-in support for “doc comments”, which are associated with a function, member, module, etc. Use these for documenting what a struct member may do, instead of regular comments which have no semantic meaning to the language.

Recommendation: Use doc-comments instead of regular comments.

Resolved

Audit (Phase 1)

SUMMARY

AP-01	Incorrect Implementation of Swap Actions	**Critical**	**Resolved**
AP-02	Token Withdrawal Message Constructed Without Sorting Funds	**Critical**	**Resolved**
AP-03	Precision Information is Hardcoded to 6 Decimal Places per Denomination	**Major**	**Resolved**
AP-04	Possible Execution Errors Due to Lack of Zero-Balance Check	**Critical**	**Resolved**
AP-05	Incorrect Swap Amount Calculations	**Major**	**Resolved**
AP-06	Protocol Performance Fee Collection Relies on Unmodified `State`	Minor	**Resolved**
AP-07	Non-KUJI Staker Performance Fees Are Locked in Contract	**Major**	**Resolved**
AP-08	Use Taker Fees from Orderbook Config for Cost Calculations	**Major**	**Resolved**
AP-09	No Explicit Validation of `from` or `to` Denomination During Swaps	**Major**	**Resolved**
AP-10	Replace Expensive Swap Cost Logic with Worst-case Approximation	Minor	**Resolved**
AP-11	Unnecessary Extra Computation in `calc_pool_size`	Info	**Resolved**
AP-12	Arbitrary Tokens Returned During Withdrawals	Info	**Noted**
AP-13	Fee Address is Statically Defined	Info	**Resolved**
AP-14	Use Ghost Amounts as Basis for Calculating USD Values	Info	**Resolved**
AP-15	Ghost Token Value Should Use mul_ceil During Withdrawal	Minor	**Resolved**
AP-16	`denom_addresses` Checks and Loads May Become Expensive	Info	**Noted**
AP-17	`denom_addresses` Configuration Not Well-structured	Minor	**Resolved**
AP-18	No Capital Controls Implemented	Info	**Noted**

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

ENTROPIC

Detailed Findings

AP-01 - Incorrect Implementation of Swap Actions

Critical

Description: The rebalancer immediately withdraws any orders placed during the Swap action, effectively simulating a partial market swap, and not actually swapping the whole amount. In addition, the following ghost deposit message assumes that the swap completed instantly with the calculated swap details, and will fail, causing a denial of service.

Recommendation: Use market orders instead of limit orders, modifying the off-chain swap algorithm to compensate. If this is not an option, then a more complex system tracking and claiming orders from the exchange must be implemented, with amounts deposited into GHOST only via withdrawal/swap callbacks to confirm the expected amounts match the actual swap amounts.

Resolved

AP-02 - Token Withdrawal Message Constructed Without Sorting Funds

Critical

Description: In `src/helpers/calc_withdraw_tokens.rs:116`, the coins fed into the `BankMsg::Send` are not sorted. The Cosmos SDK will error on unsorted coins. Thus, this may present a denial of service issue during withdrawal actions.

Recommendation: Use the `cw-utils` library's `NativeBalance` helpers to sort the balance, before constructing the `Bankmsg`.

Resolved

AP-03 - Precision Information is Hardcoded to 6 Decimal Places per Denomination

Major

Description: In `src/helpers/calc_withdraw_tokens.rs:53`, and `src/helpers/helpers_func.rs:{28, 91}`, the oracle price is normalized using a statically defined 6 decimal places. Should any denomination whitelisted on the contract be of more or less than six decimals, this will introduce inconsistencies when calculating pool sizes and withdrawal amounts, which could lead to an exploitable loss of funds.

Recommendation: Store decimal precision information alongside the whitelisted denomination information, and use this to normalize the oracle price.

Resolved

AP-04 - Possible Execution Errors Due to Lack of Zero-Balance Check

Critical

Description: In `src/handler/deposit.rs:50`, there is no zero-check on `fee_calc_result.1`, and attempting to send a coin with zero amount will cause an execution error and revert the transaction, causing a denial of service until `fee_calc_result.1` is no longer zero. Additionally, if the entire performance fee is low, the mint message in `src/helpers/take_fee.rs:36` may attempt the minting of a zero coin, which will also cause an execution error.

Recommendation: Only add the fee message if `fee_calc_result.1` is not zero, and only add the mint message if `to_mint` is not zero.

Resolved

AP-05 - Incorrect Swap Amount Calculations

Major

Description: The swap amount calculations in `src/helpers/swap_calcs.rs:82` lead to an incorrect `swap_amount`, as calculated on line `96`. In addition, the calculations must manually implement the exact swap mechanics of the exchange, as further actions in `src/handler/swap.rs` depend upon these calculations.

Recommendation: Do not depend on precomputing swap details from the exchange. Adopt a defensive model by executing actions only after a callback from the order submission.

Resolved

AP-06 - Protocol Performance Fee Collection Relies on Unmodified `State`

Minor

Description: In `src/helpers/take_fee.rs:13`, the current working `State` is being used to calculate the protocol's performance fee. This creates an unstated assumption that the performance/profit calculation relies on `total_funds_usd_new` not to include any funds that have just been deposited or withdrawn. If the `take_fee` function is called after any deposit or withdraw logic, this may cause inconsistencies such as fees to be taken on the deposit principal.

Recommendation: Extract protocol performance fee calculation and disbursal to a hook which is invoked before any handler functions.

Resolved

AP-07 - Non-KUJI Staker Performance Fees Are Locked in Contract

Major

Description: In `src/helpers/take_fee.rs:39`, the naUSD used as a performance fee is minted to the contract address. The share not sent to the Kujira address (90%) is not sent anywhere, and thus, is locked in the contract.

Recommendation: If there are more than zero naUSD minted as a performance fee, send the 90% share to an external fee address.

Resolved

AP-08 - Use Taker Fees from Orderbook Config for Cost Calculations

Major

Description: In `src/helpers/swap_calcs.rs:105`, the order book fee is calculated using a fixed fee of 0.075%. This value may diverge from the real taker fee on the order book contract.

Recommendation: Use the `fee_taker` parameter returned in the `ConfigResponse` struct from the order book config query.

Resolved

AP-09 - No Explicit Validation of `from` or `to` Denomination During Swaps

Major

Description: The check to see if the `from` denom exists in the contract state fails when the check in `src/handler/swap.rs:55` unwraps a `None` value. Likewise, the same is the case for the `to` denomination `src/handler/swap.rs:103`.

Recommendation: Insert an explicit validity check in the swap function for the `from` and `to` denominations.

Resolved

AP-10 - Replace Expensive and Complex Swap Cost Logic with Worst-case Approximation

Minor

Description: The logic in the `swap_possible` function, defined in `src/helpers/swap_calcs.rs:26` is complex and incurs very high cost, due to querying the order book state with `255` set as the limit. The logic in the `swap_possible` function is used to calculate the exact cost incurred by a limit order, to limit daily protocol costs.

Recommendation: Use an approximation deriving from the maximum fee on the entire order size to prevent expensive queries and complex logic.

Resolved

AP-11 - Unnecessary Extra Computation in `calc_pool_size`

Informational

Description: In `src/helpers/helpers_func.rs`, there are several computations which can be removed or cheapened, whilst also making the source code clearer.

Recommendation: Remove unnecessary computations in `calc_pool_size`.

Resolved

AP-12 - Arbitrary Tokens Returned During Withdrawals

Informational

Description: In `src/helpers/calc_withdraw_tokens.rs`, the logic attempts to withdraw tokens from GHOST during redemptions using the denominations that the protocol holds the most of. This results in potentially unfavorable withdrawals for the protocol.

Recommendation: Modify the withdrawal logic to use another ordering derived from current asset yield/profitability.

Noted

AP-13 - Fee Address is Statically Defined

Informational

Description: In `src/handler/deposit.rs:52` and `src/handler/withdraw.rs:44`, the address that fees are sent to is statically defined. This does not follow Kujira's best practices.

Recommendation: Add a configurable fee address parameter to the contract config.

Resolved

AP-14 - Use Ghost Amounts as Basis for Calculating USD Values

Informational

Description: In `src/handler/deposit.rs:{81, 85}`, the `mint_amount` and the `added_value_usd` use the input amount as the basis for calculating USD.

Recommendation: Calculate the ghost return amount and the associate USD value using utilities in `kujira::ghost::math`.

Resolved

AP-15 - Ghost Token Value Should Use `mul_ceil` During Withdrawal

Minor

Description: During withdrawals, the protocol should withdraw and return the minimum amount of tokens.

Recommendation: Change `token_amount.mul_floor` to `token_amount.mul_ceil` in `calc_withdraw_tokens`.

Resolved

AP-16 - `denom_addresses` Checks and Loads May Become Expensive with Many Denoms

Informational

Description: The `Config::denom_addresses` parameter is currently stored as `Vec`. This may become expensive to load, save, and scan through if many denominations are added.

Recommendation: Use a `Map` structure to store valid denominations.

Noted

AP-17- `denom_addresses` Configuration Not Well-structured

Minor

Description: The `Config::denom_addresses` parameter is structured in such a way that there is significant ambiguity as to the meaning of each element.

Recommendation: Use a well-defined `struct` containing fields for `denom`, `vault`, and `oracle`.

Resolved

AP-18 - No Capital Controls Implemented

Informational

Description: The contract does not enforce any deposit limits or capital controls to ensure that single assets do not dominate the supply.

Recommendation: Consider implementing capital controls to safeguard the protocol.

Noted

Audit (Phase 2)

SUMMARY

AP-01	State not saved after all execution paths	**Critical**	**Resolved**
AP-02	Denomination validation done using address validation	**Major**	**Resolved**
AP-03	Certain decimal normalization still uses static decimal factor	**Major**	**Resolved**
AP-04	Total USD Value not updated in State during and after Swap	**Major**	**Resolved**
AP-05	Status query results diverge from execution due to fee calculations	**Major**	**Resolved**
AP-06	Handle fee calculations in execution entry point	Minor	**Resolved**
AP-07	Fee zero checks implemented incorrectly	Minor	**Resolved**
AP-08	Various redundant computations	Minor	**Resolved**
AP-09	Total withdrawn USD value incorrectly applied to state	Minor	**Resolved**
AP-10	Configuration update does not remove old `DenomInfo` entries	Minor	**Resolved**
AP-11	Configuration update does not trigger validation	Minor	**Resolved**
AP-12	Remove extra `println!` statements	Info	**Resolved**
AP-13	Order of operations in Withdrawal	Info	**Resolved**
AP-14	Restructure invariant checks	Info	**Resolved**
AP-15	Swap function does not allow swapper to specify maximum spread	Info	**Noted**
AP-16	Swap may preemptively abort due to cost limit	Info	**Resolved**
AP-17	Collocate interface structs and definitions	Info	**Resolved**

Entropic Labs
https://entropiclabs.io
hello@entropiclabs.io

ENTROPIC

Detailed Findings

AP-01 - State not saved after all execution paths

Critical

Description: The state may not be saved after all execution paths, for example in the `DepositToGhost` callback. This can result in major inconsistencies.

Recommendation: Save the state after all calls, outside of the handler functions.

Resolved

AP-02 - Denomination validation done using address validation

Major

Description: In `src/contract.rs:{132, 137}`, the denomination check is done via address validation, and the validation check will always fail.

Recommendation: Validate using loads from the saved denomination information map.

Resolved

AP-03 - Certain decimal normalization still uses static decimal factor

Major

Description: In `src/helpers/helpers_func.rs:73`, normalization is still done via the statically defined 6 decimal places

Recommendation: Use the associated decimals variable, as implemented in Phase 1.

Resolved

AP-04 - Total USD Value not updated in State during and after Swap

Major

Description: After a swap, the total funds value may have changed due to the swap. The `from_amount` is never taken from the `total_funds_usd` in State, and the output value is never added to the `total_funds_usd` in State

Recommendation: Subtract the USD value of the swap input coin in `try_swap`, and add the USD value of the swap output coin in the callback in `src/contract.rs`

Resolved

AP-05 - Status query results diverge from execution due to fee calculations

Major

Description: The `Status` query does not account for the dilution and redemption rate adjustments that would be caused by an execution of the contract, due to `take_fee`. Any external service or contract relying upon the `Status` query for bookkeeping or calculations would receive inconsistent results between queries and execution.

Recommendation: Extract the `take_fee` calculations to a common function, which will be called both in the `take_fee` handler, and in the `query_status` handler, to keep calculations consistent.

Resolved

AP-06 - Handle fee calculations in execution entry point

Minor

Description: Passing around `fee_result` and adding the `msgs` and `events` from it in each and every handler is verbose and prone to human error.

Recommendation: Instead of doing it per-handler, add the messages and events to the response returned from the handlers. in the execute entry point. For example:

```
Unset
let response = match msg { ... }?;
if let Some(fee_result) = fee_result {
       response = response.add_messages(...).add_events(...);
}
```

Resolved

AP-07 - Fee zero checks implemented incorrectly

Minor

Description: Aborting the entire fee process, if `amount_to_kuji` or `amount_to_nami` are zero, in `src/helpers/take_fee.rs:83`, may result in state inconsistency, as the state is mutated, but the mint and send messages are not executed.

Recommendation: Instead of aborting the fee transaction, simply add the `kuji_send_msg` if and only if `amount_to_kuji` is greater than zero, and similarly for `nami_send_msg` and `amount_to_nami`, as well as the `mint_msg`.

Resolved

AP-08 - Various redundant computations

Minor

Description: Certain computations and queries should be cached, or reused for efficiency or clarity purposes.

- In `src/handler/deposit.rs:44`, the USD value is fetched, but line `49` fetches it again, incurring extra query cost.
- In `src/handler/withdraw.rs:{38, 50}` the same loop is being executed twice. Instead, replace this with a simpler and more readable `for` loop, like so:

```
Unset
let mut ghost_msgs = vec![];

let mut withdraw_coins = vec![];

for (denom_info, ghost_coin) in tokens_for_withdrawal { ... }
```

- In `src/handler/withdraw.rs:77`, the withdrawn USD value has already been calculated in `value_for_withdrawal`
- In `src/handler/swap.rs:35`, the `to_denom` is checked to be known, but the `output_denom` is checked anyway during the callback in `src/contract.rs:127`. Remove the check in the `try_swap` handler.
- In `src/handler/swap.rs:85`, the `STATE` key is saved to, even though the `src/contract.rs`'s execute entry point saves the `STATE` as the state is passed to each handler via mutable reference. Delete the extra save call in the `try_swap` handler.
- In `src/queries.rs:76`, an unnecessary zero check is being done.

Resolved

AP-09 - Total withdrawn USD value incorrectly applied to state

Minor

Description: Due to rounding, the USD value withdrawn via GHOST, calculated in `calculate_tokens_for_withdrawal`, may diverge from the `value_for_withdrawal` USD value calculated simply from the naUSD withdrawal amount.

Recommendation: Add logic to additionally return the USD value withdrawn from NAMI protocol in `calculate_tokens_for_withdrawal`, and use that for subtracting the total state USD funds in `src/handler/withdraw.rs:77`.

Resolved

AP-10 - Configuration update does not remove old `DenomInfo` entries

Minor

Description: In `src/config.rs:123`, the `denom_addresses` update only saves or overwrites each new `DenomInfo` item in state. Any `DenomInfo` that existed, but was not passed into the configuration update via the `denom_addresses` parameter *will not be removed*, as the `DENOM_ADDRESSES` map is not cleared before the update.

Recommendation: Add documentation surrounding this behavior, if intended. Otherwise, clear the `DENOM_ADDRESSES` map before saving the new `DenomInfo` to state via `DENOM_ADDRESSES.clear()`

Resolved

AP-11 - Configuration update does not trigger validation

Minor

Description: The `Config::validate` function is never called after `Config::apply_update`, potentially allowing invalid values.

Recommendation: Add a validation call after `src/contract.rs:164`

Resolved

AP-12 - Remove extra `println!` statements

Informational

Description: In `src/handler/swap.rs:{36,37,52,55,70,88,99}`, several `println!` statements seem to have been left in the codebase.

Recommendation: Remove the `println!` statements.

Resolved

AP-13 - Order of operations in Withdrawal

Informational

Description: In case any external contract interacts with the NAMI protocol and uses the supply of the naUSD token, burning the tokens as the first action will help ensure consistency at all times during message execution after withdrawal.

Recommendation: In `src/handler/withdraw.rs:95`, move the burn message to the first message, followed by the ghost withdrawal and the funds return.

Resolved

AP-14 - Restructure invariant checks

Informational

Recommendation: For consistency, and clarity / locality of code, we suggest moving the `ensure!` checks not related to sender authentication from `src/contract.rs:{104-112}` into the `try_swap` handler.

Resolved

AP-15 - Swap function does not allow swapper to specify maximum spread

Informational

Description: The `try_swap` handler uses the `belief_price` in the `SwapMsg` interface, but does not have a `max_spread` parameter input. Thus, there is no way for the swapper to set a maximum slippage / spread, potentially causing unexpected protocol losses. These losses will be limited by the daily cost, but may be incurred unnecessarily.

Recommendation: Noting this behavior in back-end logic, or allowing the swapper to input a `max_spread` alongside each `SwapMsg`

Noted

AP-16 - Swap may preemptively abort due to cost limit

Informational

Description: The `try_swap` function aborts the swap if the daily costs are reached, preemptively, at `src/handler/swap.rs:47`. This check is redundant, as the limit invariant is asserted post-swap in `src/contract.rs:138`, with transaction rollback if the limit is exceeded. Additionally, a profitable swap may also be aborted preemptively due to this redundant and early check.

Recommendation: Remove the redundant check in `src/handler/swap.rs:47`

Resolved

AP-17 - Collocate interface structs and definitions

Informational

Recommendation: It is best practice to colocate the struct definitions, currently located in `src/structs.rs` with the Execute, Instantiate, and Query messages, in `src/msg.rs`, move the struct definitions to the message file.

Resolved