Audit Report

Timewave Computer Valence Services

v1.0

April 10, 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL LICENSE.

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security has been engaged by the Neutron Audit Sponsorship Program to perform a security audit of the Timewave Computer Valence Services smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/timewave-computer/valence-services/
Commit	`ae8c3eb4d5360f19dbda4084a9a251aaecb85eec`
Scope	All contracts were in scope.
Fixes verified at commit	`c6aa41eab2972bc33e1b2893e60deaeb66ad0085` Note that changes to the codebase beyond fixes after the initial audit have not been in scope of our fixes review.

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

Timewave Computer Valence Services empowers users to create and oversee smart contract accounts, registering them to various services.
The audit scope encompasses the rebalance service, tasked with trading funds from registered user accounts through auctions to uphold and track user-defined asset portfolios.

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**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	The client provided a Figma whiteboard and a video walkthrough.
Test coverage	**High**	`cargo tarpaulin` reports a 90.77% code coverage.

Summary of Findings

No	Description	Severity	Status
1	Attackers can cause DoS of the rebalancer contract by sending arbitrary coins to the contract	Major	Resolved
2	Rebalancing mechanism scalability issues allow attackers to perform DoS attacks	Major	Partially Resolved
3	`UpdateDenomWhitelist` and `UpdateBaseDenomWhitelist` execution may cause conflicts with denoms in use and rebalancing halt	Major	Acknowledged
4	Incorrect leftover calculation if no tokens were sold during the auction	Major	Resolved
5	Prices equal to zero cause rebalancing to panic	Major	Resolved
6	Code IDs unregistered before whitelisting could allow attackers to instantiate whitelisted malicious contracts	Minor	Acknowledged
7	Division by zero if the auction start block equals the end block	Minor	Resolved
8	`SignedDecimal` type has two zero values with different signs	Minor	Resolved
9	Lack of contract ownership transfer functionality	Minor	Resolved
10	Input vectors with duplicated elements could be stored in the contract	Minor	Resolved
11	Missing address validation	Minor	Resolved
12	Centralization and risk of unintended manual price updates	Minor	Resolved
13	`max_limit` parameter is not validated	Minor	Resolved
14	The BPS value could overflow	Minor	Resolved
15	Missing validation for `auction_strategy` parameter	Minor	Resolved
16	Inefficiencies in auction finalization	Minor	Resolved
17	The duration of unsold auctions is implicitly	Minor	Resolved

	prolonged		
18	Usage of `expect` and `unwrap` functions	Informational	Acknowledged
19	Excessive gas consumption during account registration	Informational	Resolved
20	Excessive gas consumption during whitelist updates	Informational	Partially Resolved
21	Missing query pagination implementation	Informational	Resolved
22	Inefficient conditional pattern in `execute_system_rebalance`	Informational	Resolved
23	The tests calling the `SystemRebalance` method run with the `admin` account	Informational	Resolved
24	Code ID updates could silently fail	Informational	Resolved
25	Magic numbers reduce code clarity and maintainability	Informational	Resolved
26	Limit of zero is accepted as the parameter for rebalancing	Informational	Resolved
27	Remove unimplemented `migrate` and `reply` entrypoints	Informational	Resolved
28	Inconsistent hardcoded price refreshing time	Informational	Resolved
29	Print statements in production code	Informational	Partially Resolved
30	Funds are unnecessarily sent with the `FinishAuction` message	Informational	Resolved
31	Miscellaneous comments	Informational	Partially Resolved

Detailed Findings

1. Attackers can cause DoS of the rebalancer contract by sending arbitrary coins to the contract

Severity: Major

The `get_inputs` function, defined in `contracts/services/rebalancer/src/rebalance.rs:314`, queries all the coin balances of an account and then iterates over them to create a `targets_helpers` vector for a target denom within the `do_rebalance` function.

This iteration over bank balances is problematic because coins held in an account can be manipulated externally.

For instance, an attacker can enumerate all valence accounts creating several worthless tokenfactory coins, and multi-send them to all of the accounts.

This unbounded iteration will increase the gas consumption, up to the point of out-of-gas errors and leading to a DoS of the rebalancer contract.

This issue can be resolved through a contract migration, but it would have a critical impact if the contract is instantiated without defining an `admin` and hence could not be upgraded.

Recommendation

We recommend removing the `all_balances` query and handling only whitelisted denoms to eliminate the risk of a DoS attack.

Status: Resolved

2. Rebalancing mechanism scalability issues allow attackers to perform DoS attacks

Severity: Major

The rebalancing mechanism, specifically the `execute_system_rebalance` function in `contracts/services/rebalancer/src/rebalance.rs:28`, executes the rebalance of the first `limit` accounts.

However, out-of-gas errors coupled with a potential incentive misalignment can restrict the number of accounts that can be rebalanced at one time and increase the total rebalance cost. For instance, an out-of-gas error can occur in the `get_prices` function defined in `contracts/services/rebalancer/src/rebalance.rs:255-281` due to nested iterations through the denomination whitelists. The issue is the exhaustive pairing of all denominations from both whitelists, instead of limiting to those in use by the portfolios being rebalanced. Additionally, each inner iteration includes a smart query to the `AuctionsManager` contract.

Furthermore, there is a potential incentive misalignment. The `execute_system_rebalance` function is permissionless so it is expected that any user can call the function to kick off a rebalance, but there is no guarantee that a user who is paying the gas and initiating the rebalance will even have their own account rebalanced.

This implies that rebalancing may not be executed frequently enough and will not process all the accounts before the end of the cycle.

Consequently, attackers can leverage the mentioned weakness to perform various attacks aiming to cause a DoS of the rebalancer. For instance, an attacker could spam a very large number of accounts registered to the rebalancer to make the rebalance operations more costly, discouraging executors that have to spend gas on rebalancing these accounts and consequently render the service unusable.

Recommendation

We recommend optimizing the rebalancing logic to consume less computational resources.

For instance, regarding the `get_prices` function, we recommend implementing progressive price updates by splitting the whole workload into batches. Another approach could be pushing new prices into the next rebalancing cycle similarly to how funds are auctioned into the next auction. It is also worthwhile to track what pairs are used in the current set of portfolios and update only their prices.

Additionally, users should be allowed to execute the rebalancing of their own account directly without waiting for them to be selected by the system and implementing a mechanism to unregister stale or fake accounts.

Status: Partially Resolved

The client introduced a financial deterrent to mitigate attacks by instituting a registration fee for accessing the service.

Despite this measure, the possibility of an attack persists if the malicious actor is willing to incur the cost associated with fees.

3. `UpdateDenomWhitelist` and `UpdateBaseDenomWhitelist` execution may cause conflicts with denoms in use and rebalancing halt

Severity: Major

The `UpdateDenomWhitelist` message handler, in `contracts/services/rebalancer/src/contract.rs:353`, directly updates the `DENOM_WHITELIST`.

Likewise, the `UpdateBaseDenomWhitelist` message handler, in `contracts/services/rebalancer/src/contract.rs:373`, directly updates the `BASE_DENOM_WHITELIST`.

However, this may cause potential issues in the future for rebalancing and price calculation since the `RebalancerConfig` may contain targets with a denom that has been removed from the whitelist.

Consequently, the `get_inputs` function could panic in `contracts/services/rebalancer/src/rebalance.rs:308`, halting the entire rebalancing cycle.

Recommendation

We recommend not taking into account denoms not included in the whitelist during rebalancing. Additionally, when a denom is removed from the whitelist, the targets containing the specific denom can be removed.

Status: Acknowledged

4. Incorrect leftover calculation if no tokens were sold during the auction

Severity: Major

In `contracts/auction/auction/src/execute.rs:246-248`, during the execution of the `finish_auction` function, in case no tokens were sold during the auction, `total_sent_sold_token` is not updated to track the amount of funds transferred.

Consequently, the auction's `available_funds` will be added as leftovers to the next auction, leading to incorrect calculations and the impossibility of finishing the dutch auction.

Recommendation

We recommend tracking `total_sent_sold_token` also in the case no tokens were sold during the auction.

Status: Resolved

5. Prices equal to zero cause rebalancing to panic

Severity: Major

In `contracts/services/rebalancer/src/rebalance.rs:313-316`, during the execution of a rebalance, the `get_inputs` function performs a division by `price`.

However, prices can be zero since when storing the price in `contracts/auction/price_oracle/src/contract.rs:54`, there is no validation to ensure that the price is not zero.

Consequently, in case the price is zero a division by zero will occur, blocking system rebalances.

Recommendation

We recommend validating that the input price is not equal to zero.

Status: Resolved

6. Code IDs unregistered before whitelisting could allow attackers to instantiate whitelisted malicious contracts

Severity: Minor

In `contracts/services_manager/src/contract.rs:31`, the `instantiate` function does not perform any validation on the provided `whitelisted_code_ids` before they are saved to `WHITELISTED_CODE_IDS`.

This is potentially problematic because if a code ID is not currently stored on the chain, an attacker can attempt to upload a malicious whitelisted service with the aforementioned code ID.

This issue is also present in the `UpdateCodeIdWhitelist` message in `contracts/services_manager/src/contract.rs:164`, and in `contracts/auction/auctions_manager/src/contract.rs:31`.

We classify this issue as minor because this misconfiguration is only possible from the admin during the instantiation.

Recommendation

We recommend performing a query for each of the `whitelisted_code_ids` specified to ensure they are valid. This can be performed using `deps.querier.query_wasm_code_info` (Note: This query is dependent on the cosmwasm_1_2 feature).

Status: Acknowledged

7. Division by zero if the auction start block equals the end block

Severity: Minor

In `contracts/auction/auction/src/helpers.rs:7` during the execution of the `calc_price` function, the `block_diff` is calculated as a subtraction between `terms.end_block` and `terms.start_block`.

However, if both the values are equal, panic will be triggered due to a division by zero error.

Recommendation

We recommend ensuring that the start and end blocks are not equal.

Status: Resolved

8. `SignedDecimal` type has two zero values with different signs

Severity: Minor

In `packages/valence-package/src/signed_decimal.rs`, the implementation of the `SignedDecimal` type allows two zeros with different signs: specifically `(0, true)` and `(0, false)`, respectively corresponding to `+0` and `-0`.

That is possible because the implementation does not check that the values of the `Decimal` element can be equal on subtraction (e.g., `packages/valence-package/src/signed_decimal.rs:93`, `101`) or addition (e.g., `packages/valence-package/src/signed_decimal.rs:69`, `79`).

Consequently, the operations could result in two different values representing zero.

Recommendation

We recommend checking for edge cases when two `Decimal` values are equal in addition and subtraction operations and using one zero element corresponding to `(0, true)`.

Status: Resolved

9. Lack of contract ownership transfer functionality

Severity: Minor

In all the audited contracts, there is no way to update the `Admin` of the contract.

Consequently, in case of an issue with key management, or if it becomes necessary to transfer the ownership of this contract, it would be impossible.

Recommendation

We recommend implementing functionality that allows for the transfer of the `Admin` of the contract.

Status: Resolved

10. Input vectors with duplicated elements could be stored in the contract

Severity: Minor

In `contracts/services_manager/src/contract.rs:31`, the `instantiate` function does not perform any deduplication on the provided `whitelisted_code_ids` vector before saving it to `WHITELISTED_CODE_IDS`.

Likewise, in `contracts/services/rebalancer/src/contract.rs:63-64`, the `denom_whitelist` and `base_denom_whitelist` vectors are stored respectively in `DENOM_WHITELIST` and `BASE_DENOM_WHITELIST` without deduplicating them.

Consequently, duplicated elements could be stored in the contract state and handled by its logic, which would cause unexpected behaviors and inefficiencies.

Recommendation

We recommend deduplicating input vectors before storing them in the contract.

Status: Resolved

11. Missing address validation

Severity: Minor

In `contracts/services/rebalancer/src/contract.rs:152-158`, during the `update` of the `rebalance` contract, the `trustee` address is not verified.

As a result, the contracts could store invalid addresses leading to the inability to send messages correctly.

Recommendation

We recommend implementing address validation.

Status: Resolved

12. Centralization and risk of malicious manual price updates

Severity: Minor

The documentation in `contracts/auction/price_oracle/README.md:4-20` states that manual price update should be used only to set the initial price or if the price is not “fresh enough”.

However, the admin can always update the price in `contracts/auction/price_oracle/src/contract.rs:46-85` leading to centralization concerns and risks of malicious manual price updates.

Recommendation

We recommend implementing additional logic to check whether the admin sets the price during the bootstrap phase or when there is no fresh enough price for several blocks.

Status: Resolved

The client has modified the contract logic to allow the `admin` to set manual prices only in case less than four auctions have been processed or if more than two days have passed since the last auction.

13. `max_limit` parameter is not validated

Severity: Minor

The `max_limit` parameter is not validated in `contracts/services/rebalancer/src/contract.rs:207`, but the value of the similar parameter `max_limit_bps` is validated in `packages/valence-package/src/services/rebalancer.rs:70-74`.

Consequently, any `max_limit` parameter range would be considered valid leading to possible misconfigurations.

We classify this issue as minor since only the `admin` can set the aforementioned value.

Recommendation

We recommend implementing validation that `max_limit` is in range a range of `[1, 10000]`.

Status: Resolved

14. The BPS value could overflow

Severity: Minor

In `contracts/services/rebalancer/src/contract.rs:135` and `193`, a check ensures the sum of all `total_bps` does not equal `10000`.

However, this check does not consider a potential overflow in `contracts/services/rebalancer/src/contract.rs:121` and `180`, making the entire check ineffective.

Recommendation

We recommend implementing a check to prevent overflows.

Status: Resolved

15. Missing validation for `auction_strategy` parameter

Severity: Minor

In `contracts/auction/auction/src/contract.rs:63`, `msg.auction` is stored in `AUCTION_STRATEGY`.

However, the `start_price_perc` and `end_price_perc` fields of `action_strategy` are not validated before being stored in the contract leading to possible misconfigurations.

We classify this issue as minor since only the `admin` can set the aforementioned value.

Recommendation

We recommend validating `auction_strategy`.

Status: Resolved

16. Inefficiencies in auction finalization

Severity: Minor

In `contracts/auction/auction/src/execute.rs:310-318`, during the execution of the `finish_auction` function, a new element containing the computed `avg_price` is added to the `TWAP_PRICES` `VecDeque`.

However, adding a new element to a `VecDeque` could require a reallocation with an `O(n)` asymptotic cost due to resizing operations.

For instance, if the `TWAP_PRICE_LIMIT` constant is set to a large value, such as `1000`, then adding a new price could result in copying `512` items due to `VecDeque`'s internal buffer resizing. Initially, an empty buffer has a capacity of `0`, and it expands by doubling its size each time a resize is necessary starting from `4`. The last resize before increasing the capacity to `1024` provides an attack opportunity window.

Consequently, this results in excessive gas spending for users potentially leading to an out-of-gas exception.

Recommendation

We recommend initializing the `VecDeque` in `contracts/auction/auction/src/contract.rs:72` using the `with_capacity` method such that the prices buffer will always be the same size. As a result, prices will be updated with an asymptotic cost of `O(1)`.

Additionally, we recommend altering the sequence of mutating operations to `pop_back` followed by `push_front`. This change ensures the vector does not resize when reaching the configured capacity limit.

Status: Resolved

17. The duration of unsold auctions is implicitly prolonged

Severity: Minor

In `contracts/auction/auction/src/state.rs:29`, the state variable `end_block` is defined and used to interpolate the current price to terminate an active auction.

However, it only affects the duration of an auction in `contracts/auction/auction/src/execute.rs:206` in the `finish_auction` function.

In fact, when the auction is terminated in the `do_bid` function defined in `contracts/auction/auction/src/execute.rs:106`, the variable `available_amount` is checked to be strictly positive but the `end_block` variable is not checked, so the duration of the auction is implicitly extended if it still has unsold tokens.

Recommendation

We recommend checking the `end_block` variable in the `do_bid` function.

Status: Resolved

18. Usage of `expect` and `unwrap` functions

Severity: Informational

The `unwrap` and `expect` functions are used in the codebase to handle `Options`, for example in `contracts/services/rebalancer/src/contract.rs:92`, `contracts/services/rebalancer/src/rebalance.rs:136`, `308`, `316`, `411`, and `596`.

However, the usage of those functions is generally discouraged because errors result in panics without meaningful error messages. They also cause the wasm execution to abort, which does not allow handling of the error from calling functions.

Recommendation

We recommend returning errors gracefully instead of panicking.

Status: Acknowledged

19. Excessive gas consumption during account registration

Severity: Informational

In `contracts/services/rebalancer/src/helpers.rs:29-36`, the `has_dup` function causes high gas costs when registering an account with the rebalancer service.

This function, designed to identify duplicate targets, operates with *$O(N^2)$* complexity due to its nested iteration, despite the minor optimization of the inner loop.

Consequently, it will cause the transaction executor to consume an excessive amount of gas.

Recommendation

We recommend offloading the task of deduplication to the client side by accepting only a `HashSet` data structure as the input parameter.

Status: Resolved

20. Excessive gas consumption during whitelist updates

Severity: Informational

In `contracts/services_manager/src/contract.rs:171-175` an iteration occurs through code IDs designated for removal. Each loop iteration locates the position of an ID in the whitelist by performing a full linear scan of the list. Once found, another linear pass is performed to remove the ID, as it requires shifting the tail of the vector.

Consequently, the process for updating whitelists in the system operates with an *O(N * M)* complexity, where *N* is the size of the existing list and *M* represents the number of denominations to be removed.

This process is notably inefficient due to the two linear scans per ID removal, escalating the time complexity with larger lists and more IDs to remove.

The same issue impacts the updating of `BASE_DENOM_WHITELIST` and `DENOM_WHITELIST` in `contracts/services/rebalancer/src/state.rs`.

Recommendation

We recommend utilizing a `HashSet` data structure and declaring whitelists as `Item<HashSet<u64>>`, reducing the complexity of updates to *O(M)*.

Status: Partially Resolved

21. Missing query pagination implementation

Severity: Informational

In `contracts/services_manager/src/contract.rs:195-202`, the `ServicesManagerQueryMsg::GetAllServices` query iterates and retrieves all elements in the `SERVICES_TO_ADDR` without implementing any pagination.

As a result, the query may lead to potential performance issues, especially as the number of elements grows.

Recommendation

We recommend enhancing the efficiency of the `ServicesManagerQueryMsg::GetAllServices` query by implementing pagination for retrieving elements from `SERVICES_TO_ADDR`.

Status: Resolved

22. Inefficient conditional pattern in `execute_system_rebalance`

Severity: Informational

In `contracts/services/rebalancer/src/rebalance.rs:129`, the `execute_system_rebalance` function checks if `total_accounts` is less than `limit`. If this condition is met, the `SystemRebalanceStatus` is set to `Finished`.

This pattern is inefficient because if the `execute_system_rebalance` is called with a `limit` that is equal to the number of accounts left to rebalance, then the `SystemRebalanceStatus` will be `Processing` and the function will have to be called again which will rebalance zero accounts but then it will finally transition the status to `Finished`.

Recommendation

We recommend modifying the function to handle the case where a caller specifies a limit of the exact number of accounts remaining to rebalance. One method for determining if this condition is met is to check if the loaded `configs` in `contracts/services/rebalancer/src/rebalance.rs:83` returns any additional elements with a take value of `limit + 1`.

Status: Resolved

23. The tests calling the `SystemRebalance` method run with the `admin` account

Severity: Informational

In `tests/rust-tests/src/suite/suite.rs:177`, the `SystemRebalance` method is called by a user with admin privileges.

However, according to the `contracts/services/rebalancer/README.md:125`, anyone can call this method.

Consequently, the test cases could be ineffective for testing access control.

Recommendation

We recommend running this test with non-privileged users.

Status: Resolved

24. Code ID updates could silently fail

Severity: Informational

In `contracts/services_manager/src/contract.rs:159-177`, the `UpdateCodeIdWhitelist` message, enables the contract admin to update the whitelisted code IDs.

However, since code IDs contained in the `to_remove` vector are not verified to exist in the stored `WHITELISTED_CODE_IDS` vector the removal of them could silently fail.

Recommendation

We recommend returning an error in case of removal of a non-existing code ID.

Status: Resolved

25. Magic numbers reduce code clarity and maintainability

Severity: Informational

The codebase contains several values written as inline constants:

- The number `9999` is used in `contracts/services/rebalancer/src/rebalance.rs:469` and `contracts/auction/auction/src/execute.rs:258`
- The number `10_u128` is used in `contracts/services/rebalancer/src/rebalance.rs:179`

Inlining numerical constants reduces their visibility, consequently potential readers might not be aware of their signifcance. Another concern is that the values copied in different places of the codebase are difficult to modify atomically.

Recommendation

We recommend declaring these values as named constants, complete with inline documentation for better understanding and to reduce mistakes during updates.

Status: Resolved

26. Limit of zero is accepted as the parameter for rebalancing

Severity: Informational

In `contracts/services/rebalancer/src/rebalance.rs:28-32`, the function `execute_system_rebalance` takes a `limit` parameter, which determines the number of accounts to be processed in each call.

While the function is designed for permissionless by allowing incremental execution by users it accepts a `limit` of `0`.

However, this results in no advance of the rebalancing process.

Recommendation

We recommend rejecting rebalancing requests where `limit` is set to `0`.

Status: Resolved

27. Remove unimplemented `migrate` and `reply` entrypoints

Severity: Informational

Some of the contracts in the scope of this audit add the reply and migrate entrypoints but then utilize the `unimplemented!` macro for them. This will result in a panic if they are called.

The `migrate` and `reply` entrypoints are not required, so they can simply be removed rather than leaving them unimplemented.

Recommendation

We recommend removing the migrate and reply entrypoints as they are not intended to be used.

Status: Resolved

28. Inconsistent hardcoded price refreshing time

Severity: Informational

In `contracts/auction/price_oracle/src/contract.rs:71`, the hardcoded price refreshing period equals 3 days and 6 hours.

However, there are no comments on why this time was chosen. Additionally, `contracts/auction/auction/README.md:87` states, “If the price is older than 4 days, we consider it stale, and we don't start the auction.”, which does not match the hardcoded value.

Recommendation

We recommend documenting the chosen time and using a consistent value across the codebase and documentation.

Status: Resolved

29. Print statements in production code

Severity: Informational

The contracts in the scope of this audit contain several print statements. It is best practice to remove all print statements and other debugging code from the codebase before the code is released.

Recommendation

We recommend removing all print and debugging statements from the codebase.

Status: Partially Resolved

30. Funds are unnecessarily sent with the `FinishAuction` message

Severity: Informational

In `contracts/auction/auctions_manager/src/contract.rs:83`, the `info.funds` value is passed to the `FinishAuction` message.

However, since the `FinishAuction` function is not intended to receive funds, it is best practice to return an error message instead.

Recommendation

We recommend returning an error in `contracts/auction/auctions_manager/src/contract.rs:76` if funds are received.

Status: Resolved

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:

- Check the number of input arguments in the scripts in the `scripts` directory (e.g., `scripts/add_service_to_manager.sh`)
- Remove unused variables from the scripts (e.g., `scripts/new_auction.sh:8`, `15`, `24`)

- Use double quotes in scripts to prevent globbing and word splitting (e.g., `scripts/add_service_to_manager.sh:41`)
- Fix typos (the following are examples, non-exhaustive list):
    - `contracts/auction/auction/src/msg.rs:32`
    - `contracts/account/src/contract.rs:99`
    - `contracts/account/src/contract.rs:121`
    - `contracts/services_manager/src/helpers.rs:25`
    - `contracts/services_manager/src/helpers.rs:25`
    - `contracts/account/src/contract.rs:99`
    - `contracts/account/src/contract.rs:121`
    - `contracts/services_manager/src/helpers.rs:25`
- Remove duplicated lines of code in `contracts/account/src/contract.rs:108-117` and `127-136`
- Rename `TWAP_PRICE_LIMIT` variable in `contracts/auction/auction/src/execute.rs:315` to a name reflecting the semantics of a list of prices, e.g., `TWAP_PRICE_MAX_LEN`
- Remove unused errors from the contracts (examples, non-exhaustive list):
    - `contracts/auction/price_oracle/src/error.rs:8`
    - `contracts/auction/price_oracle/src/error.rs:17`
    - `contracts/auction/price_oracle/src/error.rs:11`
    - `contracts/account/src/error.rs:11`
    - `contracts/services_manager/src/error.rs:11`

Status: Partially Resolved