Audit Report

Calculated Finance

v1.1

May 23, 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 Calculated Finance.

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/CALC-FINANCE/calc-rs-osmosis
Commit	`f42f6dc619020db7aa18af48f8d68ebcdc769a7b`
Scope	All contracts were in scope.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The Calculated Finance protocol allows users to utilize dollar cost averaging (DCA) strategies by creating vaults that automatically swap tokens at a specified frequency on the Osmosis DEX.

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	**Low-Medium**	-
Code readability and clarity	**High**	-
Level of documentation	**High**	Detailed instructions on expected validation and behavior of each entry point was supplied. However, just a brief high-level overview of the protocol or functionalities was provided.
Test coverage	**High**	`cargo tarpaulin` reports a test coverage for the contracts in scope of 90.40% (1327/1468 lines covered).

Summary of Findings

No	Description	Severity	Status
1	Unauthorized swap adjustments	Critical	Resolved
2	Destination address set to the DCA contract address allows disbursement of escrowed funds at any time	Major	Resolved
3	`PerformanceAssessmentStrategy::CompareToStandardDca` is incorrectly used for a vaults' `performance_assessment_strategy` when choosing `SwapAdjustmentStrategyParams::WeightedScale`	Major	Resolved
4	Spot price usage may be exploited through oracle manipulation	Major	Resolved
5	`due_date` of the disburse escrow task is not enforced	Minor	Resolved
6	Default `slippage_tollerance` is set to the maximum value	Minor	Resolved
7	Vault deposits can trigger recursive trigger executions	Minor	Resolved
8	Lack of configuration parameter validation	Minor	Resolved
9	Inconsistent pair identification in storage	Minor	Resolved
10	`minimum_received_amount` not applied to swaps might give a false sense of security	Minor	Resolved
11	Lack of validation upon swap adjustment	Minor	Resolved
12	Lack of validation in liquidity provision cache	Minor	Resolved
13	Lack of validation of custom swap fee denom	Minor	Resolved
14	Misleading lack of feedback upon fee removal	Informational	Resolved
15	New vaults may be inactivated right after creation	Informational	Acknowledged
16	Lack of role-based access controls for the pausing mechanism	Informational	Acknowledged
17	Swap pair route can have duplicate entries	Informational	Resolved

18	Vault label length is not validated to be less than 100 characters	Informational	Resolved
19	Inability to update important vault configuration parameters	Informational	Partially Resolved
20	Fees can be set to 100% of the swap amount	Informational	Resolved
21	Centralization on the cancel vaults feature	Informational	Acknowledged
22	Inconsistent validation function naming and functionality	Informational	Resolved
23	Optional label parameter in update vault function should be required	Informational	Resolved
24	Late ownership validation is inefficient	Informational	Resolved
25	Panics prevent composability and may negatively affect user experience	Informational	Resolved
26	Responses lacking meaningful attributes	Informational	Resolved
27	Lack of pagination functionality in query	Informational	Acknowledged
28	Overflow checks not enabled for release profile	Informational	Resolved
29	“Migrate only if newer” pattern is not followed	Informational	Resolved
30	Use of magic numbers decreases maintainability	Informational	Resolved
31	Ownership transfer not implemented	Informational	Acknowledged

Detailed Findings

1. Unauthorized swap adjustments

Severity: Critical

The `update_swap_adjustment_handler` function in `contracts/dca/src/handlers/update_swap_adjustment_handler.rs` allows updating the swap adjustments for a specific strategy, effectively increasing or decreasing the number of tokens swapped. However, this function, which is supposed to be callable only by the contract admin, lacks authorization checks. An attacker can save arbitrary swap adjustments for any strategy, potentially griefing the protocol – for example, by setting it to a value of `0`.

We would like to point out that the described vulnerability was also detected by the Calculated Finance team during the audit. A fix was prepared, which was then reviewed by the Oak Security team.

Recommendation

We recommend adding authorization checks to the `update_swap_adjustment_handler` function to guarantee that only the contract admin can alter swap adjustments.

Status: Resolved

2. Destination address set to the DCA contract address allows disbursement of escrowed funds at any time

Severity: Major

In the DCA contract's `get_disbursement_messages` function in `contracts/dca/src/helpers/disbursement.rs`, swapped and escrowed funds are disbursed to the provided destination addresses of a vault. Destination addresses can be used as arbitrary callbacks by specifying a custom message.

However, the destination address is not checked to ensure it is not the DCA contract itself. This allows for the execution of messages in the context of the DCA contract as the caller of the message. For example, providing the `DisburseEscrow` message would lead to the disbursement of escrowed funds whenever funds are distributed, contradicting the intended behavior of disbursing escrowed funds only in specific situations, e.g., upon vault cancellation.

Recommendation

We recommend validating destination addresses to ensure none of them is the DCA contract itself. Alternatively, if the DCA contract should be a valid destination address, we recommend ensuring that the provided message is safe to be executed.

Status: Resolved

3. `PerformanceAssessmentStrategy::CompareToStandardDca` is incorrectly used for a vaults' `performance_assessment_strategy` when choosing `SwapAdjustmentStrategyParams::WeightedScale`

Severity: Major

Creating a new vault with the `create_vault_handler` function in `contracts/dca/src/handlers/create_vault.rs` allows specifying a performance assessment and swap adjustment strategy. The swap adjustment strategy can be either set to `SwapAdjustmentStrategyParams::RiskWeightedAverage` or `SwapAdjustmentStrategyParams::WeightedScale`. The performance assessment strategy determines if a vault should continue to swap tokens based on the performance of the vault and is supposed to be only used for `SwapAdjustmentStrategyParams::RiskWeightedAverage`.

However, the `performance_assessment_strategy` is set to `PerformanceAssessmentStrategy::CompareToStandardDca` in line `120` for both swap adjustment strategies, regardless of the chosen strategy.

As a result, the vault's `escrow_level`, supposed to be set to `0`, is set to the escrow level config value (`risk_weighted_average_escrow_level`) in line `127`. Additionally, incorrect swap fees are charged. Instead of imposing swap and automation fees in the `disburse_funds_handler` function in `contracts/dca/src/handlers/disburse_funds.rs:43` and `48`, performance fees are applied. As the performance fees are calculated based on the profitability delta between the standard DCA and the DCA+ strategy and the vault not using the DCA+ strategy, performance fees will be inaccurately calculated.

Recommendation

We recommend using the function parameter `performance_assessment_strategy_params` to initialize the `performance_assessment_strategy` in line `120`.

Status: Resolved

4. Spot price usage may be exploited through oracle manipulation

Severity: Major

During trigger execution, the belief price is determined using the `query_belief_price` function in `contracts/dca/src/helpers/price.rs:8-48`. In line `38`, the pools are queried for their spot price, which could be manipulated on low liquidity pools.

This issue is exacerbated due to the lax limits on slippage described below in the issue Default slippage_tollerance is set to the maximum value and because `ExecuteTrigger` is permissionless. An attacker could monitor existing vaults to compile a list of targets and exploit them by executing triggers at the correct time after manipulating the oracle's spot price.

Recommendation

We recommend using time-weighted average prices (TWAP) during asset price queries.

Status: Resolved

5. `due_date` of the disburse escrow task is not enforced

Severity: Minor

Canceling a vault through the `cancel_vault_handler` function stores a task in storage to disburse escrowed funds. The `due_date` is designed to limit the execution of the task to a specific time frame and is calculated based on the current block time in `contracts/dca/src/handlers/cancel_vault.rs:35`.

However, when disbursing escrowed funds via the `disburse_escrow_handler` function in `contracts/dca/src/handlers/disburse_escrow.rs`, the previously stored task is not loaded, and its `due_date` is not checked. This means that escrowed funds can be disbursed at any time by the contract admin, even when the `due_date` has been surpassed.

Recommendation

We recommend attempting to load the task from storage and, if available, enforcing its `due_date` before disbursing escrowed funds.

Status: Resolved

6. Default `slippage_tollerance` is set to the maximum value

Severity: Minor

Slippage tolerance determines the maximum allowable deviation between the anticipated number of tokens a user expects to receive, such as in a token swap, and the actual number of tokens received. A higher value implies a smaller amount of tokens being received, increasing the loss of value for the user.

Within the audited contracts, slippage tolerance is used in two functionalities:

- Calling the trigger through the `ExecuteTrigger` message, which swaps tokens accordingly to the selected strategy in `contracts/dca/src/handlers/execute_trigger.rs:207`.
- Providing liquidity using the `ZProvideLiquidity` message in `contracts/dca/src/handlers/z_provide_liquidity.rs:37`.

In both instances, the user can optionally define a custom slippage value. If none is provided, the default value will be used.

However, the default value for `slippage_tolerance` (for `ExecuteTrigger`) and `slippage_tollerance` (for `ZProvideLiquidity`, please note the spelling mistake in the variable name) is `Uint128::one()`. Consequently, users may experience significant losses when using these functionalities without defining a custom slippage tolerance value. In the case of pairs with low liquidity, such token swaps may be detected by MEV bots, which could sandwich the messages and thereby extract maximum value from the user.

In addition, note that in both of the above-mentioned cases, the functions do not validate the optional user-provided slippage parameters. Consequently, if their value is greater than 1, there will be an underflow in `contracts/dca/src/helpers/swap.rs:23` and `contracts/dca/src/handlers/z_provide_liquidity.rs:49`, which will result in a panic.

Recommendation

We recommend using a default slippage value between 5-10% to protect user funds. Additionally, we recommend validating user-supplied slippage parameters.

Status: Resolved

7. Vault deposits can trigger recursive trigger executions

Severity: Minor

Anyone can deposit in a vault via the `deposit_handler` function in `contracts/dca/src/handlers/deposit.rs`. If the vault was inactive prior to the deposit, a new trigger is saved in line `96` with the target time set to the current block time, prompting an immediate execution. If the vault has destination addresses configured to deposit distributed funds directly into other vaults, this can possibly lead to recursive trigger executions.

We would like to point out that the described vulnerability was also detected by the Calculated Finance team during the audit. A fix was prepared, which was then reviewed by the Oak Security team.

Recommendation

We recommend determining the next execution date and time of a vault when depositing by using the `get_next_target_time` function, calculating the time of the next trigger based on the interval set in the vault configuration.

Status: Resolved

8. Lack of configuration parameter validation

Severity: Minor

Most of the configuration parameters are properly validated before saving them to storage. However, the following parameters are missing validation steps both upon instantiation in `contracts/dca/src/handlers/instantiate.rs:23-33` and upon update in `contracts/dca/src/handlers/update_config.rs:24-34`:

- A maximum size is not enforced for the `fee_collectors` vector. This could lead to out-of-gas errors during iteration if a big enough list of collectors is supplied.
- Empty allocations in the `fee_collectors` vector are not rejected.
- A `page_limit` of zero is not rejected, even though it would render some of the contracts' queries ineffective, as no elements would be returned. In addition, the `assert_page_limit_is_valid` function implements a default value of 30, therefore if the limit is below that number, any query that makes use of the default value will return an error.

Recommendation

We recommend enforcing a suitable maximum size for the `fee_collectors` vector and asserting that no allocation is zero. In addition, we recommend enforcing a minimum equal to the default value (30 in this case) for the `page_limit`.

Status: Resolved

9. Inconsistent pair identification in storage

Severity: Minor

When saving new pairs in `contracts/dca/src/state/pairs.rs:7-9`, the storage key is determined as `format!("{}-{}", denoms[0], denoms[1])`. Therefore, both pairs `A-B` and `B-A` are saved as different pairs with potentially different routes. The `find_pair` function in lines 15-23 checks the existence of both options, however, it will always return the first that matches the submitted ordering.

Returning different routes depending on the order of the assets in a pair will cause unexpected inconsistencies in gas costs and swap fees.

Recommendation

We recommend sorting the denoms alphabetically before storing them. Alternatively, the presence of both A-B and B-A keys could be verified prior to saving, and if an existing pair is found, it should be overwritten.

Status: Resolved

10. `minimum_received_amount` not applied to swaps might give a false sense of security

Severity: Minor

When executing a trigger through the `execute_trigger_handler` function in `contracts/dca/src/handlers/execute_trigger.rs`, `vault.minimum_received_amount` is checked against `belief_price`. However, it is not provided to the actual swap message in lines 207-215, instead, `vault.slippage_tolerance` is supplied.

Since no verification exists to guarantee that `vault.slippage_tolerance` aligns with the same risk level as `minimum_received_amount`, usage of the `minimum_received_amount` creates a false sense of security for users, as it is not effectively used in the final swapping process.

Recommendation

We recommend either removing `minimum_received_amount` or checking that it matches the level of risk in `vault.slippage_tolerance`.

Status: Resolved

11. Lack of validation upon swap adjustment

Severity: Minor

The `update_swap_adjustment_handle` function allows for arbitrary values to be submitted as the swap adjustment, affecting the number of tokens to be swapped by this factor. If a large value is assigned, the swap will fail, as the resulting amount designated for swapping will exceed the available funds.

Recommendation

We recommend at least enforcing that the submitted value is between 0 and 100. In addition, defining more restrictive bounds will be beneficial as users will be confident that detrimental extreme adjustments are not possible.

Status: Resolved

12. Lack of validation in liquidity provision cache

Severity: Minor

The `z_provide_liquidity_handler` function in `contracts/dca/src/handlers/z_provide_liquidity.rs:28-35` does not validate the user-submitted `provider_address` before saving it to the `PROVIDE_LIQUIDITY_CACHE` storage. Providing an incorrect address will cause the subsequent reply handlers to fail, wasting computational resources, or rendering the locking useless if a duration of zero is provided.

Recommendation

We recommend validating that `provider_address` is a valid address.

Status: Resolved

13. Lack of validation of custom swap fee denom

Severity: Minor

The `create_custom_swap_fee_handler` function in `contracts/dca/src/handlers/create_custom_swap_fee.rs:9-23` does not validate that the supplied denom actually exists. In case of a typo, the admin might not be aware of the error, and since the expected denom will not be found, no fee will be applied to the desired operations.

Recommendation

We recommend ensuring that the supplied denom is valid before saving it to the `CUSTOM_FEES` storage.

Status: Resolved

14. Misleading lack of feedback upon fee removal

Severity: Informational

The `remove_custom_swap_fee_handler` function in `contracts/dca/src/handlers/remove_custom_swap_fee.rs:7-19` does not validate that the supplied denom is part of the current custom fees. Since no feedback is returned when removing a non-existing element from the storage, the admin would not be aware of the error, for example, in case of a typo. Therefore undesired fees will still be charged afterward.

Recommendation

We recommend checking that the supplied denom exists in the `CUSTOM_FEES` storage, returning an error if not found.

Status: Resolved

15. New vaults may be inactivated right after creation

Severity: Informational

The `create_vault_handler` function in `contracts/dca/src/handlers/create_vault.rs` does not assert that the initial token deposit `info.funds[0].amount` is sufficient to guarantee a successful swap of the desired number of tokens, i.e. `swap_amount`. As a consequence, such a vault would be inactive and unable to perform token swaps due to insufficient funds.

Recommendation

We recommend asserting that `swap_amount` is less than `info.funds[0].amount`.

Status: Acknowledged

16. Lack of role-based access controls for the pausing mechanism

Severity: Informational

The codebase implements a pausing mechanism, which is in line with best practices. However, all of the administrative functions of the contract are centralized in the `admin` role, which goes against the principle of least privilege.

Segregating the pauser role has the additional benefit of swifter reactions in case of need when assigned to an EOA compared to the admin that might be managed by a multisig or a governance contract.

Recommendation

We recommend implementing a separate pauser role that can turn on and off the pausing mechanism.

Status: Acknowledged

17. Swap pair route can have duplicate entries

Severity: Informational

The `create_pair_handler` function in `contracts/dca/src/handlers/create_pair.rs` allows the contract admin to create and update swap pairs. The `pair.route` field is used to specify the route (i.e., pool ids) that should be used for swapping tokens. While there are various checks in place to ensure that the route is not empty and the route matches the `base_denom` and `quote_denom`, there is no check to ensure that the route does not contain duplicate entries.

A swap pair with duplicate route entries will increase the total swap fees and gas costs when swapping tokens.

Recommendation

We recommend removing duplicate entries from `pair.route` when creating or updating a pair via the `create_pair_handler` function.

Status: Resolved

18. Vault label length is not validated to be less than 100 characters

Severity: Informational

Users can specify a label when creating a new vault via the `create_vault_handler` function in `contracts/dca/src/handlers/create_vault.rs`. The label is stored in the `vault.label` field and is solely used for off-chain display purposes. While updating a vaults label via the `update_vault_label_handler` function in `contracts/dca/src/handlers/update_vault.rs` is restricted to 100 characters, there is no such restriction when creating a new vault.

In addition, empty labels are allowed which do not serve the intended asset identification purpose.

Recommendation

We recommend asserting the provided vault label length via the `assert_label_is_no_longer_than_100_characters` function when creating a new vault. In addition, a minimum label length should be enforced.

Status: Resolved

19. Inability to update important vault configuration parameters

Severity: Informational

A vault owner can currently update the vault label via the `update_vault` function in `contracts/dca/src/handlers/update_vault.rs`. However, there are other important vault configuration parameters that cannot be updated once the vault is created. This includes the swap slippage tolerance (`slippage_tolerance`), the minimum receive amount (`minimum_receive_amount`), and the destination addresses (`destinations`) for fund disbursement.

The inability to update these parameters can be problematic in certain situations. For instance, if the vault owner wants to increase the slippage tolerance to ensure that the vault is able to swap tokens even when the market is volatile or liquidity is low, or if the initially provided slippage value was too conservative. In such cases, they would have to cancel the vault and create a new one.

Recommendation

We recommend allowing the vault owner to update certain vault configuration parameters such as slippage (`slippage_tolerance` and `minimum_receive_amount`) and the destination addresses (`destinations`) for fund disbursement.

Status: Partially Resolved

The client added the ability to update vault destinations to address scenarios where the destination has become invalid. To modify other vault parameters, the vault should be canceled, and a new vault with the desired parameters should be created.

20. Fees can be set to 100% of the swap amount

Severity: Informational

Various fees are employed throughout the contract, such as `Config.swap_fee_percent` and `Config.delegation_fee_percent`, which calculate the swap and automation fees, respectively. Custom fees for a denom are specified using the `CUSTOM_FEES` field. Values for these fees are validated to be less than or equal to 100% in the `update_config` function in `contracts/dca/src/state/config.rs:45` and in the `create_custom_fee` function in line `56`.

If the contract admin sets the fees to 100%, all swapped funds will be charged as fees, and the vault owner will not receive any funds.

Recommendation

We recommend enforcing reasonable upper bound checks for `CUSTOM_FEES` as well as `Config.swap_fee_percent` and `Config.delegation_fee_percent`.

Status: Resolved

21. Centralization on the cancel vaults feature

Severity: Informational

The `cancel_vault_handler` function allows both the owner of a vault and the admin to set a vault as canceled in `contracts/dca/src/handlers/cancel_vault.rs:23`.

Although these added administrative privileges over user assets aim to counter potential exploits, their effectiveness might be constrained by the team's response time. Furthermore, they could inadvertently amplify the potential harm that may arise from a malicious insider or compromised admin keys.

Recommendation

Consider increasing the coverage of the pause mechanism and removing the administrative powers for canceling user vaults.

Status: Acknowledged

The client decided to maintain this functionality as a safety measure in production, allowing for the return of users' funds in the event of a bug or exploit.

22. Inconsistent validation function naming and functionality

Severity: Informational

Calculated Finance uses validation functions with names following the `assert_{tested condition}` pattern. In the following cases, these names are not consistent with what is actually validated in the function:

- The `assert_risk_weighted_average_escrow_level_is_less_than_100_percent` function checks if `weighted_average_escrow_level` is less or equal to 100%. An error will be returned for values greater than this number.

- The `assert_target_start_time_is_in_future` function checks if `target_start_time` is not less than the current time, which is misleading because no error will be returned for calls in the same block.

Recommendation

We suggest adjusting the names of these functions or changing the conditions being checked to ensure the naming is consistent with the functionality.

Status: Resolved

23. Optional label parameter in update vault function should be required

Severity: Informational

The `update_vault_handler` function called from the `ExecuteMsg::UpdateVault` message serves the sole purpose of modifying the `label` parameter in user-created vaults.

However, this `label` parameter is optional rather than mandatory. If no label is provided, the transaction will consume gas unnecessarily.

Recommendation

We suggest changing the `label` parameter type from `Option<String>` to `String`.

Status: Resolved

24. Late ownership validation is inefficient

Severity: Informational

In `contracts/dca/src/handlers/update_vault.rs:26`, the `update_vault_handler` function performs a validation check to ensure that the transaction's sender is the owner of the updated vault. However, while updating the vault, the validation function is called after the `update_vault` is performed in lines `19` to `24`.

If the validation was performed at the beginning of the `update_vault_handler` function, the transaction would be able to consume less gas before being reverted in case of an unauthorized execution attempt.

Recommendation

We recommend performing the ownership validation at the beginning of the `UpdateVault` call.

Status: Resolved

25. Panics prevent composability and may negatively affect user experience

Severity: Informational

In several places in the code, the use of the `expect` method and the direct use of the `panic!` macro has been noticed. In both cases, in the event of reaching an error in the code, the execution will panic and unrecoverably abort the transaction.

It is good practice to gracefully return errors that can be handled by the calling context. This allows for better composability of protocols and is more user-friendly due to human-readable error messages.

The following instances have been observed:

- `contracts/dca/src/helpers/price.rs:66`
- `contracts/dca/src/helpers/routes.rs:41`
- `contracts/dca/src/helpers/vault.rs:80,117,131`
- `contracts/dca/src/helpers/disbursement.rs:16`
- `contracts/dca/src/helpers/fees.rs:25`
- `contracts/dca/src/helpers/time.rs:38`
- `contracts/dca/src/state/vaults.rs:77,98,161`
- `contracts/dca/src/state/events.rs:27,40`
- `contracts/dca/src/handlers/execute_trigger.rs:59`
- `contracts/dca/src/handlers/get_events_by_resource_id.rs:30`
- `contracts/dca/src/handlers/get_events.rs:26`
- `contracts/dca/src/handlers/z_provide_liquidity.rs:45,48`
- `contracts/dca/src/types/vault.rs:77`

Recommendation

We suggest replacing these calls with the return of a well-described `ContractError` for each potential error scenario.

Status: Resolved

26. Responses lacking meaningful attributes

Severity: Informational

It is best practice to include information on both the action performed and its result in response attributes, since that provides valuable information to users and off-chain components.

The following responses will benefit from additional attributes:

- In `contracts/dca/src/handlers/update_swap_adjustment_handler.rs:14`, no details on the performed action are added.
- In `contracts/dca/src/handlers/update_config.rs:44`, no details on all updated parameters are provided.
- In `contracts/dca/src/handlers/disburse_escrow.rs:64`, shortcircuit does not add a reason.
- In `contracts/dca/src/handlers/execute_trigger.rs:144` and `148`, shortcircuit does not add a reason.

Recommendation

Consider adding meaningful attributes to these responses.

Status: Resolved

27. Lack of pagination functionality in query

Severity: Informational

The `get_pairs` function in `contracts/dca/src/state/pairs.rs:25-30` does not implement a pagination functionality. Instead, it iterates over all pairs stored in `PAIRS`.

It is best practice to offer pagination for unbounded data queries to avoid potential out-of-gas exceptions.

Recommendation

We recommend implementing a pagination mechanism as successfully done in the other query functions.

Status: Acknowledged

28. Overflow checks not enabled for release profile

Severity: Informational

The `contracts/dca` contract does not enable `overflow-checks` for the release profile.

While enabled implicitly through the workspace manifest, future refactoring might break this assumption.

Recommendation

We recommend enabling overflow checks in all packages, including those that do not currently perform calculations, to prevent unintended consequences if changes are added in future releases or during refactoring. Note that enabling overflow checks in packages other than the workspace manifest will lead to compiler warnings.

Status: Resolved

29. “Migrate only if newer” pattern is not followed

Severity: Informational

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

Recommendation

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

Status: Resolved

30. Use of magic numbers decreases maintainability

Severity: Informational

In `contracts/dca/src/helpers/fees.rs:93`, a hard-coded number literal without context or a description is 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.

Recommendation

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

Status: Resolved

31. Ownership transfer not implemented

Severity: Informational

The contract within the scope of this audit does not allow transferring of the privileged `admin` to a new address. This creates a risk of unauthorized access to the contract’s privileged functionalities in case the private key to the admin account gets compromised.

In addition, operational needs may require transferring the role to a different address at any given moment.

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

Status: Acknowledged