Audit Report

Phase

v1.0

April 13, 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 the Phase Finance Smart Contracts implementing a Dollar-Cost Averaging (DCA) strategy.

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 GitHub repository:

Repository	https://github.com/phase-fi/phase-contracts
Commit	`7303dc1f8e18403859788460b15c2b61f4e93c5a`
Scope	The scope of this audit was limited to the contract in `contracts/pf-dca`

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 smart contracts within the scope of this audit implement Phase Finance's version of a DCA strategy.

The scope of this audit is restricted to the `pf-dca` contract, which implements the DCA strategy to be managed by end-users.

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**	-
Code readability and clarity	**Medium**	Overall lack of in-line comments describing functions. Todo comments and debug code suggest that the code is still under development.
Level of documentation	**Medium**	A brief overview of functionality and requirements was provided, but no comments on individual functions.
Test coverage	**Low-Medium**	The reported coverage for the DCA contract is `41.5%`.

Summary of Findings

No	Description	Severity	Status
1	Permissionless trade execution allows for price manipulation and arbitrage opportunities	Major	Resolved
2	Lack of validation of configuration parameters upon instantiation	Major	Resolved
3	DCA end not enforced	Minor	Resolved
4	DCA execution might run out of gas if too many destinations are configured	Minor	Resolved
5	Rounding issues on swap message creation	Minor	Acknowledged
6	Users' assets can get stuck in the DCA contract until cancelation	Minor	Resolved
7	Coarse-grained authorization	Minor	Resolved
8	Commented and debug code	Informational	Resolved
9	Ineffective resume and pause operations allowed	Informational	Resolved
10	Overflow checks not enabled for release profile	Informational	Resolved
11	Unnecessary panics lead to bad UX	Informational	Resolved

Detailed Findings

1. Permissionless trade execution allows for price manipulation and arbitrage opportunities

Severity: Major

In `contracts/pf-dca/src/execute.rs:66-81`, the `try_perform_dca` function is permissionless, introducing a number of risks. While the execution of a single transaction might pose a small risk, i.e. a loss of funds for users due to arbitrage opportunities for another users, a coordinated attack that uses a larger amount of scheduled transactions might have a severe impact by extending (or even creating) such arbitrage opportunities to a degree that they could become market manipulations and enable the attacker to buy/sell funds at large discounts/premia.

First, an attacker could use a set of scheduled transactions to manipulate oracle spot prices or TWAPs by triggering them simultaneously in a low liquidity environment.

Second, an attacker can always arbitrage the slippage of the scheduled transactions by placing an order in front and behind the target transaction – a so-called sandwich attack. As the attacker can send the `PerformDca` message themself, this gives them the ability to write scripts that optimize for such an attack, for example reverting in the case of an unforeseen event.

Third, an attacker can combine the first two attacks for an inverted sandwich attack: The attacker could use one set of transactions to manipulate some oracle price to sell their own asset at a higher price and use a second set of transactions to provide enough liquidity to sell at the manipulated price, irrespective of the slippage limit.

In addition to arbitrage, the owner might be forced to execute additional DCAs that are ready for execution when planning to cash out by having their `CancelDca` transaction front-run. However, this scenario will cause a minor inconvenience to the user compared to the above.

The potential impact of this issue could be considered critical. We classify it as major though since the likelihood of exploitation is low as it requires a subset of the following market conditions: High slippage tolerances or large amounts of scheduled transactions within a short period of time and/or an oracle to be manipulated.

Recommendation

We recommend implementing access controls to limit the execution of the `PerformDca` message to the owner and/or the protocol.

Status: Resolved

2. Lack of validation of configuration parameters upon instantiation

Severity: Major

The `pf-dca` contract does not perform any validation on the configuration parameters set upon instantiation in `contracts/pf-dca/src/contract.rs:53-64`, potentially causing issues that could render the contract unusable.

- `destination_wallet` address is not validated. An invalid address could cause any swap to fail as the message in line `138` will have an invalid recipient. In addition, the current implementation does not follow the documentation that states the destination wallet should default to the sender's address.
- `source_denom` is not validated. A wrong letter case of a denom or a denom that does not exist will cause the related swap message defined in `contracts/pf-dca/src/execute.rs:104-108` to fail, potentially leaving the amount to be swapped sitting idle in the contract.
- The `destinations` vector can be empty or the denoms not sanitized. This will lead to either no swaps happening at all or swaps failing.
- `max_slippage` can be any Decimal number, even outside of the 0-100 range leading to potential inconsistencies. In addition, a high slippage is not recommended as it will allow for arbitrage opportunities as the ones described at Permissionless trade execution allows for price manipulation and arbitrage opportunities.
- `router_contract` address is not validated, which would cause all swaps to fail if an invalid address is provided.
- `swap_interval` can be set to zero, effectively allowing users to trigger all the swaps at once.

We classify this issue as major given that none of the above parameters can be updated once the contract is deployed, which means that any of these errors require a new deployment.

Recommendation

We recommend implementing thorough validation of the affected parameters, in particular:

- Validation of addresses using the `address_validate` API.
- Adherence to the documentation by defaulting the `destination_wallet` to the owner's address.
- Ensuring that the submitted denoms exist.
- Enforcing the slippage to be at least within the 0-100 range. In addition, consider enforcing a standard maximum slippage of 0.5% to limit any arbitrage opportunity. If users are expected to engage in highly illiquid asset trading, a higher cap on slippage of around 15% is suggested. Although this should be advised with care to protect unsuspecting users from arbitrage.

Status: Resolved

3. DCA end not enforced

Severity: Minor

The pf-dca contract does not enforce the configuration's num_trades limit on the number of trades performed in contracts/pf-dca/src/execute.rs:66-121. This allows users to keep using the PerformDca message for as long as the contract has enough funds to perform additional swaps.

Although not implying a security risk per se, the actual implementation differs from the implied behavior by the existence of DcaConfig.num_trades and STATE.num_trades_executed, which are not used otherwise.

Recommendation

We recommend adding a check in the try_perform_dca function so no further trades are allowed once STATE.num_trades_executed reach DcaConfig.num_trades.

If this is not intended, we recommend removing the code related to STATE.num_trades_executed.

Status: Resolved

4. DCA execution might run out of gas if too many destinations are configured

Severity: Minor

In contracts/pf-dca/src/execute.rs:83-86, the iterations performed over destinations might run out of gas if too many destinations are configured. As a consequence, trade execution might fail permanently. In that case, the only way to recover the assets in the contract is the cancellation of the whole DCA.

Recommendation

We recommend enforcing a maximum number of destinations.

Status: Resolved

5. Rounding issues on swap message creation

Severity: Minor

The pf-dca contract allows swapping into multiple destination denoms, distributing the total amount using their assigned weight. The calculation of the final amount to be traded to each denom is done in contracts/pf-dca/src/execute.rs:92-100. The division by total_weight is an integer division, leaving any remainder to stay idle in the contract until the user decides to cancel the DCA. This corresponds to rounding down/using the floor function.

A proof of concept unit test can be found in the Appendix: Rounding issues on swap message creation.

Recommendation

We recommend either

- checking for the remainder of the division and adding the amount to the next swap or
- performing further validation of the weights and `amount_per_trade` upon instantiation to ensure that the results do not have a remainder, returning an error otherwise.

Status: Acknowledged

The client acknowledges this issue, stating that funds are recoverable by users and the fix would require more involved changes.

6. Users' assets can get stuck in the DCA contract until cancelation

Severity: Minor

The `pf-dca` contract checks only if enough assets are sent along in `contracts/pf-dca/src/contract.rs:45` to satisfy that all trades N_t can be funded with the correct amount A_t per trade. The documentation states that the total amount A_D should satisfy $A_D = N_t * A_t$ such that it is required that the user submits the correct amount of funds. However, the code allows initializing contract with $A_D \geq N_t * A_t$, creating a possibility for users to lock up excessive amounts of funds accidentally.

The only way for the user to receive those funds back is to cancel the whole strategy, which would then require them to re-instantiate to resume the strategy with the correct amount of assets.

In addition, when the swaps for a given asset fail, the remaining not swapped assets are left in the contract with no way for the user to either claim them or retry the swap. This may lead to frozen funds accumulating in the contract, unusable by the user until cancellation of the strategy.

Recommendation

We recommend sending additional assets back to the user after instantiation or returning an error in case additional funds are sent along.

Status: Resolved

7. Coarse-grained authorization

Severity: Minor

The `pf-dca` contract's `verify_sender` function in `contracts/pf-dca/src/helpers.rs:7` does not follow best practices for authorization since it combines two separate entities into one layer of privileges by giving access to privileged functionality to both `owner` and `destination_wallet`.

For example, if the owner tries to pause the contract in order to stop the money flow into the `destination_wallet`, the `destination_wallet` can revert this action by unpausing the contract. The only way for the owner to prevent this is to cancel the DCA.

In the future, when further functionality is added, this may lead to an overly privileged party with unintended consequences.

Recommendation

We recommend implementing role-based authorization with distinct permissions such that permissioned functionality is only accessible to clearly specified parties.

Status: Resolved

8. Commented code

Severity: Informational

The contracts within the scope of this audit include commented code. Although not a security issue, commented code indicates that the codebase may not yet be ready for release and may negatively affect readability and maintainability.

The following instances were found:

- `contracts/pf-dca/src/state.rs:5-29`
- `packages/phase-finance/src/msg.rs:45-50`

In addition, language potentially harmful to the client's reputation was found in `contracts/pf-dca/src/tests.rs:135` and `packages/phase-finance/src/constant.rs:1`.

Recommendation

We recommend removing the instances mentioned above from the production version of the codebase.

Status: Resolved

9. Ineffective resume and pause operations allowed

Severity: Informational

The `pf-dca` contract's `resume_dca` and `pause_dca` functions in `contracts/pf-dca/src/execute.rs:34` and `46` do not check if the contract is actually paused or not.

Although not a security issue, pausing or unpausing a contract that is not already in that state will just spend gas without actually changing the contract's state.

Recommendation

We recommend checking if the contract is paused at the beginning of the affected functions.

Status: Resolved

10. Overflow checks not enabled for release profile

Severity: Informational

The following contract and package do not enable `overflow-checks` for the release profile:

- `contracts/pf-dca/Cargo.toml`
- `packages/phase-finance/Cargo.toml`

While enabled implicitly through the workspace manifest, a 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

11. Unnecessary panics negatively affect usability

Severity: Informational

In `contracts/pf-dca/src/helpers.rs`, the conversion from `string` to `Uint128` is assuming valid data due to the filtering of anything different than numbers.

There are two cases that are unhandled though:

- Empty strings (when `token_string` is not starting with number)
- Numbers that are too big to parse

Those cases cause the function to panic, negatively affecting the user experience compared to error messages.

Recommendation

We recommend handling these cases and providing users with actionable error messages instead of panicking.

Status: Resolved

Appendix: Test Cases

Rounding issues on swap message creation

```rust
#[test]
fn rounding_error_weights() {
    let mut deps = do_instantiate2();
    let info = mock_info("creator", &[]);

    for n in 1..2 {
        let mut env = mock_env();
        env.block = BlockInfo {
            height: 10000000,
            chain_id: "osmos".to_string(),
            time: Timestamp::from_seconds(9_000_000_000 + n) ,
        };

        let res = execute(
            deps.as_mut(),
            env,
            info.clone(),
            ExecuteMsg::PerformDca {},
        )
        .unwrap();
        for n in 0..res.messages.len() {
            println!("{:?}", res.messages[n].msg);
        }
    }

}

fn do_instantiate2() -> OwnedDeps<MockStorage, MockApi, MockQuerier> {
    let mut deps = mock_dependencies();
    let info = mock_info(ADMIN_ADDR, &coins(1230, "uosmo"));
    let env = mock_env();

    let instantiate_msg = InstantiateMsg {
        destination_wallet: "osmo123".to_string(),
        strategy_type: StrategyType::Linear,
        destinations: vec![
            CoinWeight {
                denom: "uion".to_string(),
                weight: Uint128::from(10u128),
            },
            CoinWeight {
                denom: "ujuno".to_string(),
                weight: Uint128::from(190u128),
            },
            CoinWeight {
                denom: "uxxx".to_string(),
                weight: Uint128::from(100u128),
            },
            CoinWeight {
                denom: "uyyyy".to_string(),
                weight: Uint128::from(100u128),
            },
        ],
        max_slippage: Decimal::from_ratio(1u128, 100u128),
        amount_per_trade: Uint128::from(123u128),
        num_trades: Uint128::from(10u128),
        swap_interval: Duration::Time(1),
        source_denom: "uosmo".to_string(),
        router_contract: "osmoabc".to_string(),
    };

    instantiate(deps.as_mut(), env, info, instantiate_msg).unwrap();

    deps
}
```