Audit Report

Apollo DAO Smart Contracts

v1.0

March 17, 2022

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 Apollo DAO to perform a security audit of the Apollo DAO 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 behaviour.
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:

https://github.com/apollodao/apollo-contracts

Commit hash: `cf186a0d31c40330b3f0b39d5da0920c5578dfbd`

The following directories were included in the audit:

```
contracts/*
packages/apollo-protocol/*
```

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 Apollo DAO offers yield management strategies on the Terra blockchain. The audited smart contracts implement different strategies including auto-compounding vaults for Anchor, Apollo itself and Mirror.

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**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve 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 a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Users can steal all LP tokens from the Apollo static strategy	Critical	Resolved
2	Registering fees in the factory contract does not update the strategy's extension reward index	Major	Acknowledged
3	Users are unable to collect rewards after withdrawing/zapping out of strategy	Major	Resolved
4	Strategies that start with paused rewards can never receive any rewards	Major	Resolved
5	Current block height not used in reward calculation	Minor	Resolved
6	APR calculation might panic if staked liquidity value is zero	Minor	Resolved
7	Price queries do not ensure that price base is not outdated	Minor	Resolved
8	Queried reward and LP token prices in Apollo autocompound strategy are not checked to not be outdated	Minor	Resolved
9	Anchor reward rate calculation does not account for overlaps in distribution schedule	Minor	Resolved
10	Updates of strategy config fields could lead to incorrect state	Minor	Resolved
11	Oracle contract's query responses for TVL and APR will always return zero	Minor	Resolved
12	Apollo factory config missing validation parameters	Minor	Resolved
13	Existing vesting accounts will be overwritten on updates	Minor	Resolved
14	Strategy contract instantiation lacks validation of swap commission	Minor	Resolved
15	Apollo reward info query always returns zero pending rewards	Minor	Resolved
16	Oracle returns wrong LP price for pairs that do not	Minor	Resolved

	contain UST		
17	Pausing/unpausing rewards of a strategy has no effect without explicitly calling update weights	Minor	Resolved
18	Factory contract owner can emergency withdraw all funds	Minor	Acknowledged
19	Updating strategy addresses in the factory contract may leave funds inaccessible	Minor	Resolved
20	Oracle contract is relying on one feeder per asset	Informational	Resolved
21	Vault sell asset attribute contains wrong amount	Informational	Resolved
22	Number of seconds in month/year does not account for leap year	Informational	Acknowledged
23	Vesting account queries will panic if queried with a time prior to genesis time	Informational	Resolved
24	Claimed amount values greater than claimable amount will cause panics	Informational	Resolved
25	Canonical address transformations are inefficient	Informational	Acknowledged
26	Finding a strategy by address is inefficient	Informational	Resolved
27	Use of unbounded loops over strategies is inefficient and could exhaust gas	Informational	Acknowledged
28	Unnecessary calculations outside of match branch	Informational	Resolved
29	Unused code	Informational	Resolved
30	Usage of hard-coded values decreases maintainability	Informational	Resolved
31	Overflow checks not set for release profile in most packages	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	**Medium-High**	-
Code readability and clarity	**Medium**	-
Level of Documentation	**Medium**	-
Test Coverage	**Medium-High**	-

Detailed Findings

1. Users can steal all LP tokens from the Apollo static strategy

Severity: Critical

Since the `apollo-static` strategy simply holds LP tokens, the query in `contracts/strategies/apollo-static/src/strategy.rs:38` will query the LP token balance of the contract including all previous deposits by other users. All of these deposits will be assigned to the caller, and the user can then extract all that value by immediately withdrawing/zapping out.

Recommendation

Rather than querying the balance with `query_token_balance` in line `38`, we recommend sending the message with the correct amount.

Status: Resolved

2. Registering fees in the factory contract does not update the strategy's extension reward index

Severity: Major

In the `handle_register_fee` function in `contracts/apollo-factory/src/contract.rs:586`, the `EXTENSION_TOTAL_COLLECTED_FEES` are updated, but the strategy's `extension_reward_index` field is not adjusted. That leads to an inconsistent state. Additionally, `update_strategy` is called, even though there were no changes to the strategy.

Recommendation

We recommend either updating the strategy's `extension_reward_index` or removing the `update_strategy` function call.

Status: Acknowledged

The Apollo team states that this code was updated after the Community Farming Event was started and will not be used anymore.

3. Users are unable to collect rewards after withdrawing/zapping out of strategy

Severity: Major

In `contracts/apollo-factory/src/contract.rs:816`, strategies are skipped if a user has no active deposits in a strategy. Users might have withdrawn/zapped out of a strategy, and hence have zero shares, but they might not have collected rewards yet. Such users are unable to collect the rewards from the strategies because of the condition described above.

Recommendation

We recommend skipping reward calculation, but adding `reward_info.lm_pending_reward` to `claim_amount` and then setting it to zero.

Status: Resolved

4. Strategies that start with paused rewards can never receive any rewards

Severity: Major

The `rewards_paused` field cannot be updated in the `handle_update_strategy` function in `contracts/apollo-factory/src/contract.rs:516`. That means that a strategy that started without rewards can never receive any rewards.

Recommendation

We recommend adding the ability to update the `rewards_paused` field.

Status: Resolved

5. Current block height not used in reward calculation

Severity: Minor

In `packages/apollo-protocol/src/vault/vaults/anchor.rs:119`, the reward info from Anchor deposits is queried. That query uses a `block_height` of `None`, which will cause Anchor to return the latest stored values, rather than recalculating them at the current block height.

Recommendation

We recommend passing the current block height to the `StakingQueryMsg`. With the current block height, the current reward value will be correctly calculated.

Status: Resolved

6. APR calculation might panic if staked liquidity value is zero

Severity: Minor

When calculating the APR in `contracts/strategies/apollo-static/src/strategy.rs:493`, the contract might panic when `staked_liquidity_value` is 0.

Recommendation

We recommend adding a condition to check if `staked_liquidity_value` is `0` before calculating the APR, as is done in `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs:490`.

Status: Resolved

7. Price queries do not ensure that price base is not outdated

Severity: Minor

In multiple price queries from the oracle contract, the time since `last_updated_quote` is asserted to be greater than the strategy's `price_age_limit`. However, the time since `last_updated_base` is not asserted – which might lead to usage of outdated prices. Instances of that issue are found in:

- `contracts/strategies/apollo-static/src/strategy.rs:433`
- `contracts/strategies/apollo-static/src/strategy.rs:467`
- `contracts/strategies/apollo-static/src/strategy.rs:481`
- `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs:409`
- `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs:465`
- `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs:479`
- `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs:423`

Recommendation

As done on `last_updated_quote`, we recommend implementing a check to ensure that `last_updated_base` is not outdated.

Status: Resolved

8. Queried reward and LP token prices in Apollo autocompound strategy are not checked to not be outdated

Severity: Minor

When querying the `reward_token_price` and `lp_token_price` in `packages/apollo-protocol/src/strategy/strategies/autocompound/strategy.rs:309-325` and in `369-385`, a check should be added to ensure these prices are not outdated.

Recommendation

We recommend introducing a check to make sure the prices are not outdated when they are returned from querying the oracle.

Status: Resolved

9. Anchor reward rate calculation does not account for overlaps in distribution schedule

Severity: Minor

The reward rate calculation for Anchor rewards in `packages/apollo-protocol/src/vault/vaults/anchor.rs:147` does not account for overlaps in the distribution schedule coming from the anchor config. The Anchor factory does allow overlaps. In the event of an overlap, the reward rate would be off.

Recommendation

We recommend adjusting the reward rate calculation logic to consider potential overlaps

Status: Resolved

10.Updates of strategy config fields could lead to incorrect state

Severity: Minor

In multiple places in the codebase, `update_config` functions allow updates to values that could lead to incorrect contract states. The following are references to contracts and fields that should not be allowed to be updated:

- `contracts/strategies/apollo-static/src/strategy.rs`
  - `apollo_factory`
  - `base_token`
  - `base_denom`
  - `asset_token`
  - `asset_token_pair`
  - `apollo_strategy_id`
- `contracts/strategies/autocompound/apollo/src/strategy.rs`
  - `apollo_factory`
  - `staking_contract`
  - `base_token`
  - `reward_token`
  - `base_denom`
  - `oracle_contract`
  - `apollo_strategy_id`
  - `farm_factory_contract`
- `packages/apollo-protocol/src/strategy/strategies/autocompound/reuse.rs`
  - `apollo_factory`
  - `base_token`
  - `staking_contract`
  - `reward_token`
  - `base_denom`
  - `reward_token_pair`
  - `oracle_contract`
  - `farm_factory_contract`

Recommendation

We recommend removing the fields referenced above from their respective contract's `update_config` functions.

Status: Resolved

11. Oracle contract’s query responses for TVL and APR will always return zero

Severity: Minor

The query responses for `query_tvl` and `query_apr` in `contracts/oracle/src/contract.rs:164` and `174` from the Oracle contract return values that are never set. Consequently, the query responses will always be zero.

Recommendation

We recommend removing the TVL and APR queries from the oracle contract or implementing the functionality.

Status: Resolved

12. Apollo factory config missing validation parameters

Severity: Minor

The Apollo factory contract does not currently validate the distribution schedule in the `instantiate` function in `contracts/apollo-factory/src/contract.rs:58` and in the `handle_update_config` function in line `58`. It allows end dates that are smaller than the start dates and does not check for overlaps and gaps.

Recommendation

We recommend adding validation for date ranges, overlaps and gaps.

Status: Resolved

13. Existing vesting accounts will be overwritten on updates

Severity: Minor

When registering vesting accounts in `contracts/vesting/presale-vesting/src/contract.rs:106`, the list of vesting accounts is iterated over and the vesting information for each address is stored. If the vesting info for an address exists, it will be overwritten during this loop.

Recommendation

We recommend handling the condition of an existing vesting address by either merging the new vesting information with the existing one, or by returning an error if the vesting information already exists for an address.

Status: Resolved

14. Strategy contract instantiation lacks validation of swap commission

Severity: Minor

The swap commission is not currently validated, which could result in panics:

- `swap_commission` in the `instantiate` function in `contracts/strategies/apollo-static/src/contract.rs`
- `swap_commission` in the `instantiate` function in `contracts/strategies/autocompound/apollo/src/contract.rs`
- `swap_commission` in the `instantiate` function in `packages/apollo-protocol/src/strategy/strategies/autocompound/contract.rs`

Recommendation

We recommend adding validation to those values.

Status: Resolved

15. Apollo reward info query always returns zero pending rewards

Severity: Minor

The `query_reward_info` function in `packages/apollo-protocol/src/vault/vaults/apollo.rs:118` always returns zero pending rewards.

Recommendation

We recommend returning the actually pending rewards from the factory contract.

Status: Resolved

16. Oracle returns wrong LP price for pairs that do not contain UST

Severity: Minor

The LP price calculation in `packages/apollo-protocol/src/oracle.rs:146-148` only uses the `price_res.rate` of one asset, which does not account for the price of the other asset.

This is not an issue in the current Apollo contracts since all pairs include UST, which has a price of 1 and hence would cancel out. It will become problematic though if Apollo ever uses pairs that do not contain UST.

Recommendation

We recommend adding support for pairs that do not contain UST.

Status: Resolved

17. Pausing/unpausing rewards of a strategy has no effect without explicitly calling update weights

Severity: Minor

If rewards are paused through the `handle_update_strategy` function in `contracts/apollo-factory/src/contract.rs:557`, the reward weight of the strategy is not updated. This implies that the strategy will still receive rewards, until update weights is called.

The same issue exists when unpausing rewards.

Recommendation

When dealing with strategies where `is_apollo` is `false`, we recommend setting the reward weight of the strategy to zero when it is paused, and setting it back to the reward weight when it is unpaused, as is currently done in the `handle_update_reward_weights` function in line `647`.

Status: Resolved

18. Factory contract owner can emergency withdraw all funds

Severity: Minor

The current implementation allows the owner of the factory contract to emergency withdraw all funds from all strategies. This can happen directly through the `EmergencyWithdraw` message, or indirectly through the `PassMessage` message in `contracts/apollo-factory/src/contract.rs`.

If the owner key is ever compromised, the total value locked in the Apollo protocol can be stolen. This kind of centralization creates a risk for all users. It also makes the owner key signatories potential targets, or in the case of a governance contract being the owner it introduces an incentive for an attacker to economically exploit the governance mechanism by buying enough governance tokens to then steal the total value locked.

We classify this issue as minor since the likelihood of an attack is low.

Recommendation

We recommend changing the emergency withdrawal mechanism to either work on a per-user basis, effectively removing centralization, or to withdraw to a predetermined custody contract that lets users reclaim their tokens. We also recommend removing the `PassMessage` message. At the very least, we recommend using a multi-sig with enough parties and a high threshold and adoption of operation security best practices.

Status: Acknowledged

The Apollo team states that the current owner is a multi-sig with 11 signatories. That multi-sig will be replaced by governance contracts in the future.

19. Updating strategy addresses in the factory contract may leave funds inaccessible

Severity: Minor

The `handle_update_strategy` in `contracts/apollo-factory/src/contract.rs` allows updates to a strategy's address. Such updates without state migration will result in inaccessible funds.

We classify this issue as minor since it can only be caused by the contract owner.

Recommendation

We recommend removing the ability to update strategy addresses.

Status: Resolved

20. Oracle contract is relying on one feeder per asset

Severity: Informational

The current oracle contract only supports a single feeder per asset, see `contracts/oracle/src/contract.rs:96`. That leads to centralization and makes the oracle more subject to manipulation.

Recommendation

We recommend using multiple data sources, either by having a multi-party computation to aggregate prices off-chain or by having on-chain voting on prices, where the median of the prices could be taken.

Status: Resolved

21. Vault sell asset attribute contains wrong amount

Severity: Informational

While the vault's `sell_asset` function correctly calculates the appropriate amount of the asset to sell, the response attribute in `packages/apollo-protocol/src/vault/vault_trait.rs:263` is incorrectly calculated as it does not consider the previous balance. We classify this issue as informational since it has no impact on the functionality of contracts, but leads to emission of a wrong event.

Recommendation

We recommend updating the attribute's value to `asset_balance - asset_token_balance_before`.

Status: Resolved

22. Number of seconds in month/year does not account for leap year

Severity: Informational

In the vesting contracts, the amount of seconds in a month and year should be updated to reflect leap years. Currently `ONE_YEAR_IN_SECONDS` is set to equal `31536000` which is the correct amount of seconds in a year when not including leap years. But the current value for `MONTH_IN_SECONDS` does not reflect `ONE_YEAR_IN_SECONDS` divided by 12. The following are references to the occurrences of these values:

- `contracts/vesting/cfe-vesting/src/contract.rs:19-20`
- `contracts/vesting/cfe-vesting/src/tests.rs:14-15`
- `contracts/vesting/presale-vesting/src/contract.rs:18-19`
- `contracts/vesting/presale-vesting/src/tests.rs:11-12`

Recommendation

We recommend changing both `ONE_YEAR_IN_SECONDS` and `MONTH_IN_SECONDS` to be consistent and consider leap years:

- `MONTH_IN_SECONDS` from `2592000` to `2629744`
- `ONE_YEAR_IN_SECONDS` from `31536000` to `31556926`

Status: Acknowledged

The Apollo team states that these values were intended and would cause rounding issues if modified. They acknowledge that this would have a minor impact on the calculations involving these values.

23. Vesting account queries will panic if queried with a time prior to genesis time

Severity: Informational

In the vesting contracts, the time delta is calculated by subtracting the `genesis_time` from the `ending_time_target`. If `VestingAccount`, CfeAccount, `VestingAccounts` are queried with a time prior to `genesis_time`, the functions will panic. Below are references to the time delta calculations:

- `contracts/vesting/cfe-vesting/src/contract.rs:358`
- `contracts/vesting/cfe-vesting/src/contract.rs:392`
- `contracts/vesting/presale-vesting/src/contract.rs:187`

Recommendation

We recommend returning zero if the queries are made with a time that occurs before the genesis time.

Status: Resolved

24. Claimed amount values greater than claimable amount will cause panics

Severity: Informational

In `contracts/vesting/cfe-vesting/src/contract.rs:367`, `395`, and `contracts/vesting/presale-vesting/src/contract.rs:189` pending rewards are calculated without performing a check whether the claimed amount is greater than the claimable amount. This can occur if the genesis time has been updated after amounts have been claimed, or in the case of the presale vesting contract if amounts are reduced after claims have happened.

Recommendation

We recommend performing a checked subtraction and return `0` if the calculation underflows.

Status: Resolved

25. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend using `Addr` instead of `CanonicalAddr`.

Status: Acknowledged

26. Finding a strategy by address is inefficient

Severity: Informational

In `contracts/apollo-factory/src/state.rs:101`, an unbounded loop is used to find a strategy by address. The cost of executing that function increases with the number of strategies in Apollo, and may eventually lead to the iteration running out of gas.

Recommendation

We recommend accepting the `strategy_id` to prevent unnecessary iteration over all of the strategies. Alternatively, the factory could store a map from address to strategy id.

Status: Resolved

27. Use of unbounded loops over strategies is inefficient and could exhaust gas

Severity: Informational

There are multiple occasions throughout the contracts where unbounded loops over strategies are used. Those loops will consume more gas as the number of strategies increases. Examples are:

- `contracts/apollo-factory/src/contract.rs:622`
- `contracts/apollo-factory/src/contract.rs:664`
- `contracts/apollo-factory/src/contract.rs:795`
- `contracts/apollo-factory/src/contract.rs:996`
- `contracts/apollo-factory/src/state.rs:101`

Since strategies are likely limited, we only consider this issue to be informational.

Recommendation

To prevent any gas issues in the future, we recommend enforcing a maximum number of strategies through the factory.

Status: Acknowledged

28. Unnecessary calculations outside of match branch

Severity: Informational

In `contracts/apollo-factory/src/contract.rs:791-792` `strategies_len` and `limit` are both calculated outside of the batch branch where they are needed. For invocations that do not meet this condition, they are calculated unnecessarily.

Recommendation

We recommend that both `strategies_len` and `limit` are moved to within the `None` branch of the match condition.

Status: Resolved

29. Unused code

Severity: Informational

There are multiple occurrences of unused code in the codebase:

- The `TOTAL_COLLECTED_FEES` storage item in `contracts/apollo-factory/src/state.rs:31` is defined and read, but never written to.
- The `oracle` field of the `Config` struct in `contracts/apollo-factory/src/state.rs:43`.

Unused code adds overhead and impacts readability and maintainability.

Recommendation

We recommend removing unused code.

Status: Resolved

30. Usage of hard-coded values decreases maintainability

Severity: Informational

The use of hard-coded values (also called magic numbers) is generally discouraged, since values do not communicate their meaning. That will negatively impact maintainability of the code. An example can be found in `packages/apollo-protocol/src/vault/vaults/apollo.rs:172`.

Recommendation

We recommend replacing hard-coded values with a constant that is aptly named.

Status: Resolved

31. Overflow checks not set for release profile in most packages

Severity: Informational

The following Cargo.toml files do not enable overflow-checks for the release profile:

- `contracts/apollo-factory/Cargo.toml`
- `contracts/apollo-token/Cargo.toml`
- `contracts/collector/Cargo.toml`
- `contracts/oracle/Cargo.toml`
- `contracts/strategies/apollo-static/Cargo.toml`
- `contracts/strategies/autocompound/anchor/Cargo.toml`
- `contracts/strategies/autocompound/apollo/Cargo.toml`
- `contracts/strategies/autocompound/mirror/Cargo.toml`
- `contracts/vesting/cfe-vesting/Cargo.toml`
- `contracts/vesting/presale-vesting/Cargo.toml`
- `packages/apollo-protocol/Cargo.toml`
- `packages/bignumber/Cargo.toml`

Recommendation

Even though this check is implicitly applied to all packages from the workspace's `Cargo.toml`, we recommend also explicitly enabling overflow checks in every individual package. That helps prevent unintended consequences when the codebase is refactored in the future.

Status: Resolved