Audit Report

Apollo DAO Smart Contracts

Aug 18, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Philip Stanislaus

Oak Security

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

Introduction

Purpose of this Report

Oak Security has been engaged by the Apollo team 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 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:

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

Commit hash: `b3f2e9a66e49f35d51aeea0f68d20c8286737a39`

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 simple auto-compounding vaults for mAsset pools on Mirror as well as the MIR-UST pool on 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 withdraw and zap out all LP tokens from the autocompound strategy	Critical	Resolved
2	Users might receive additional asset tokens during zap out	Major	Resolved
3	Users can still deposit, zap in and then execute, withdraw and zap out additional tokens after an emergency withdrawal	Major	Resolved
4	Zapping in high amounts of base denom tokens leads to relatively big remainder of tokens that can will be used by the next user for free	Major	Resolved
5	Half up tie breaking rule within rounding function will lead to last user being unable to withdraw full bond	Minor	Resolved
6	Hard-coded base token denom might break autocompound strategy execution	Minor	Resolved
7	Updating the staking contract, base or asset token or the asset token pair in the autocompound strategy may lead to inconsistent protocol state	Minor	Resolved
8	Users may lose bonded tokens after emergency halt if Mirror's unstaking logic changes in the future	Minor	Resolved
9	Performance fee values greater than 1 will lead to failing execution	Minor	Resolved
10	Storing config parameter as strings may lead to runtime errors	Minor	Resolved
11	Overflow checks not set for release profile in most packages	Minor	Resolved
12	Autocompound strategy execution will fail if total bond amount is zero	Informational	-
13	Coins other than UST sent to the factory are lost	Informational	-
14	Querying all strategies or user strategies of the factory might run out of gas	Informational	-

15	Warchest config value in factory contract cannot queried and updated	Informational	-
16	Factory's TVL calculation queries each strategy's TVL twice	Informational	-
17	Unnecessary storage of user info by base token within apollo-base-strategy	Informational	-
18	Unused factory contract field in collector config	Informational	-
19	Unused terraswap factory contract field in autocompound strategy config	Informational	-
20	Unused constants in factory state	Informational	-

Code Quality Criteria

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

Detailed Findings

1. Users can withdraw and zap out all LP tokens from the autocompound strategy

Severity: Critical

Both the autocompound strategy's `Withdraw` and `ZapOut` messages allow any user to withdraw all LP tokens from the strategy. Within the handlers of both messages, the `BaseStrategy`'s `withdraw` function is called with the user supplied withdrawal amount. That `withdraw` function calls the `remove_shares` function, which reduces a user's bond by the given amount. If the amount is greater than the currently bonded tokens by the user, the function gracefully reduces the bond to zero `contracts/apollo-base-strategy/src/strategy.rs:132`. Then the `withdraw` and `zap_out` handlers in `contracts/apollo-base-strategy/src/strategy.rs:455` and `832` unstake and send the full user supplied amount to the user, allowing the user to steal all LP tokens from the protocol.

Recommendation

Instead of gracefully reducing the user's bonds to zero if a user tries to withdraw more tokens than they own, we recommend either returning an error in `contracts/apollo-base-strategy/src/strategy.rs:132` or returning the actual amount the user can unbond and using that within the `withdraw` and `zap_out` handlers.

Status: Resolved

2. Users might receive additional asset tokens during zap out

Severity: Major

During provision of liquidity within the autocompound strategy, it could happen that asset tokens remain in the contract. This is caused since the amount of assets sent to TerraSwap depends on the ratio of base to asset tokens in TerraSwap, which is determined in `contracts/strategies/autocompound/src/strategy.rs:366`. If more asset tokens were sent to TerraSwap, arbitrageurs could make a risk free profit. Such remaining tokens are not kept in the contract though – they will be sold in `contracts/strategies/autocompound/src/strategy.rs:512` and then sent in line `561` to the next user sending a `ZapOut` message.

This issue only affects asset tokens. For base tokens, a query is made at the beginning of the `zap_out` function in `contracts/strategies/autocompound/src/strategy.rs:847` to determine the

base token balance, and only the difference is sent to the the user, see `contracts/strategies/autocompound/src/strategy.rs:553`.

Recommendation

As is currently implemented for base tokens, we recommend adding a query for the current asset token balance at the beginning of the `zap_out` function, and then just selling the asset token difference in `contracts/strategies/autocompound/src/strategy.rs:512`.

Status: Resolved

3. Users can still deposit, zap in and then execute, withdraw and zap out additional tokens after an emergency withdrawal

Severity: Major

The current logic implementing the emergency withdrawal in `contracts/strategies/autocompound/src/strategy.rs:902` does not prevent users from depositing and zapping in additional tokens, which also allows execution of the strategy and withdrawal and zapping out of those additional tokens. This will lead to incorrect states.

For example, strategy execution will distribute rewards to users that do not have bonded tokens anymore through the `increase_stake_amount` function in `contracts/strategies/autocompound/src/strategy.rs:92`.

Moreover, additionally added tokens cannot be withdrawn with another emergency withdrawal, since unbonding is using `total_bond_amount`, which is higher than the actual bonded amount.

Recommendation

We recommend preventing further deposits and zap ins after an emergency withdrawal – either by setting `deposits_paused` to `true` in the `handle_emergency_withdraw` function in `contracts/apollo-factory/src/contract.rs:304` or by adding a similar flag to the autocompound strategy that will be set to `false` in the `emergency_withdraw` function in `contracts/strategies/autocompound/src/strategy.rs:902`.

Status: Resolved

4. Zapping in high amounts of base denom tokens leads to relatively big remainder of tokens that can will be used by the next user for free

Severity: Major

During a zap in of the autocompound strategy, half of the provided base denom balance is spent on TerraSwap to buy asset tokens in the `buy_asset` function. Depending on the amount of tokens swapped and the liquidity of the asset token/base token pair on TerraSwap, there can be quite a sizable price impact. After buying the asset tokens, liquidity will be provided at the current token ratio of the TerraSwap pair, see `contracts/strategies/autocompound/src/strategy.rs:367`. Due to the price impact, there will be base asset tokens remaining in the contract. If the amount of tokens zapped in is big and the liquidity on TerraSwap is low, that remaining base asset balance will be sizable.

The same issue occurs during execution of the strategy if the asset token is not equal to the reward token, see `contracts/strategies/autocompound/src/strategy.rs:199`. In that case, the `buy_asset` function will run as well and will also lead to base asset tokens being left in the contract.

In the current implementation, the next user that zaps into the strategy will be automatically using these remaining tokens as part of their provided tokens.

Recommendation

We recommend adjusting the logic in the `buy_asset` function to account for the price impact, minimizing the amount of tokens being left in the contract after a swap.

Alternatively, the contract could use the amount of base denom tokens sent, rather than assuming that all available base denom tokens the contract owns were provided by the message sender. Any tokens that have not been used for liquidity provision could be refunded to the sender. This will still imply though that users that send bigger amounts of base denom tokens will experience a higher relative amount of lost tokens than users that send smaller amounts.

Status: Resolved

5. Half up tie breaking rule within rounding function will lead to last user being unable to withdraw full bond

Severity: Minor

The `decimal_rounding` function defined in `packages/apollo-protocol/src/utils.rs:47` uses the round half up tie-breaking rule. That rule has a positive bias. The function is used to determine how a user's bonds

increase as rewards accumulate in the `calculate_user_bonds_from_index` function in `packages/apollo-protocol/src/utils.rs:53`. The bias in the rounding function used will lead to the sum of all user bonds being slightly bigger than the actual total bonded amount. That will render the last user unable to withdraw all their bonds, since the balance in the contract will be slightly smaller than the bond assigned to the user.

Even though this issue involves a loss of funds, we classify it as minor since the amount affected is very small.

Recommendation

We recommend changing the rounding function to use an unbiased tie-breaking rule, such as round half to even. Alternatively, any decimals could be truncated to change the bias such that the contract will retain a tiny amount, rather than one user losing it.

Status: Resolved

6. Hard-coded base token denom might break autocompound strategy execution

Severity: Minor

In `contracts/strategies/autocompound/src/strategy.rs:303`, the base token denom is hard-coded to `"uusd"`. This is problematic, since the autocompound strategy allows a configurable base denom, see `contracts/strategies/autocompound/src/state.rs:17`. If a base denom other than `"uusd"` is configured, executing the strategy may fail.

Recommendation

We recommend using `config.base_denom` instead of a hardcoded value in `contracts/strategies/autocompound/src/strategy.rs:303`.

Status: Resolved

7. Updating the staking contract, base or asset token or the asset token pair in the autocompound strategy may lead to inconsistent protocol state

Severity: Minor

The `UpdateConfig` message of the autocompound strategy allows updates of most configuration values, but the handler in `contracts/strategies/autocompound/src/strategy.rs:571` does not include state migration to ensure a consistent protocol state. For example, an update of the `staking_contract` without migrating the staked tokens to the new contract will lead to a

failure of strategy execution and withdrawals. An update of the `base_token` or `asset_token` without migration of the balances will lead to the same issues. An update to the `asset_token_pair` contract without migrating the liquidity to the new contract will lead to a failure of withdrawals.

Recommendation

We recommend removing the possibility to update those values and instead relying on deploying a new strategy to ensure a consistent protocol state. Alternatively, state migration could be added to the `update_config` handler.

Status: Resolved

8. Users may lose bonded tokens after emergency halt if Mirror's unstaking logic changes in the future

Severity: Minor

In the `return_lp` function in `contracts/strategies/autocompound/src/strategy.rs:432`, which is called by the `Withdraw` message, the tokens to be returned to a user are set to the maximum available tokens. After an emergency withdraw, that condition could be fulfilled, since the bonded tokens by a user are still positive, while the actually available tokens are zero. In such an instance, a user could lose access to their tokens.

Example: If a user had deposited 10 LP tokens before the emergency withdraw, then deposits further 10 LP tokens after the emergency withdraw, and finally tries to withdraw 15 LP tokens, they will only receive 10 LP tokens, but their balance in the storage would be reduced by 15 LP tokens.

The same issue exists in the `withdraw_liquidity` function in `contracts/strategies/autocompound/src/strategy.rs:469`, which is called by the `ZapOut` message.

This issue is currently prevented by Mirror's staking contract since it returns an error if a user tries to unbond more LP tokens than currently bonded.

We still classify this issue as minor as it depends on an implementation detail of the Mirror contract.

Recommendation

We recommend panicing or returning an error in `contracts/strategies/autocompound/src/strategy.rs:432` and `469` to prevent a user receiving less tokens than requested.

Status: Resolved

9. Performance fee values greater than 1 will lead to failing execution

Severity: Minor

The `performance_fee` config parameter of the autocompound strategy in `contracts/strategies/autocompound/src/state.rs:24` is currently not validated. If it is set to a value greater than 1, the autocompound strategy can not be executed, since sending more than the available CW20 tokens will fail in `contracts/strategies/autocompound/src/strategy.rs:156` or in `312`.

Recommendation

We recommend validating that `performance_fee` is set to a value less than or equal to 1 in the `init` function in `contracts/strategies/autocompound/src/contract.rs:42` and in the `update_config` function in `contracts/strategies/autocompound/src/strategy.rs:642`.

Status: Resolved

10. Storing config parameter as strings may lead to runtime errors

Severity: Minor

Several config parameters are stored as strings and converted to Decimals before they are used. That leads to potential conversion failures at runtime of handlers and adds computation overhead to regular message handlers. Instances are:

- `performance_fee` in `contracts/apollo-base-strategy/src/state.rs:16`
- `max_spread` in `contracts/collector/src/state.rs:18`
- `max_spread` in `contracts/strategies/autocompound/src/state.rs:18`
- `slippage_tolerance` in `contracts/strategies/autocompound/src/state.rs:19`
- `performance_fee` in `contracts/strategies/autocompound/src/state.rs:24`

Recommendation

We recommend storing those parameters as Decimals and do the conversion in the contracts' init and config update functions.

Status: Resolved

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

Severity: Minor

While set in the workspace's `Cargo.toml` files, the following `Cargo.toml` files do not enable `overflow-checks` for the `release` profile:

- `contracts/apollo-base-strategy/Cargo.toml`
- `contracts/apollo-factory/Cargo.toml`
- `contracts/collector/Cargo.toml`
- `contracts/strategies/autocompound/Cargo.toml`
- `packages/apollo-protocol/Cargo.toml`
- `packages/bignumber/Cargo.toml`
- `packages/cw0/Cargo.toml`
- `packages/cw20/Cargo.toml`
- `packages/schema/Cargo.toml`
- `packages/std/Cargo.toml`
- `packages/storage-plus/Cargo.toml`
- `packages/storage/Cargo.toml`
- `packages/terra-cosmwasm/Cargo.toml`
- `packages/terraswap/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

12. Autocompound strategy execution will fail if total bond amount is zero

Severity: Informational

Execution of the autocompound strategy in `contracts/strategies/autocompound/src/strategy.rs:737` will fail if the total bond amount is zero. This is caused by a panic from a division by zero in `contracts/strategies/autocompound/src/strategy.rs:100`.

Recommendation

While this panic has no security implication, we recommend handling division by zero gracefully.

13. Coins other than UST sent to the factory are lost

Severity: Informational

The `handle_zap_into_strategy` function in the factory contract filters sent coins for `"uusd"` in `contracts/apollo-factory/src/contract.rs:236`, but ignores any additional coins sent to the contract. Such additional coins will be inaccessible to anyone.

This issue is classified as informational since it originates from user error.

Recommendation

We recommend checking the number of sent coins and returning an error if any coins other than `"uusd"` are received.

14. Querying all strategies or user strategies of the factory might run out of gas

Severity: Informational

The factory's `GetStrategies` and `GetUserStrategies` queries are using unbounded storage iterators in `contracts/apollo-factory/src/state.rs:91` and `132`. As more strategies are added to the factory, the gas cost of running those queries does increase.

This is not a security concern for Apollo, since those queries are not used anywhere within the current codebase. Other projects could rely on the queries though. If they do, a high amount of strategies could lead to the queries running out of gas.

Recommendation

We recommend adding pagination to the `GetStrategies` and `GetUserStrategies` queries.

15. Warchest config value in factory contract cannot queried and updated

Severity: Informational

The factory's `GetConfig` query does return the owner in `contracts/apollo-factory/src/contract.rs:510`, but not the warchest.

Likewise, the `handle_update_config` function allows an update of the owner in `contracts/apollo-factory/src/contract.rs:343`, but not of the warchest.

Recommendation

We recommend returning the warchest in the response to the `GetConfig` query and adding the ability to update the warchest.

16. Factory's TVL calculation queries each strategy's TVL twice

Severity: Informational

The `get_total_tvl` function in `contracts/apollo-factory/src/contract.rs:531` queries the TVL for each strategy in line `542`, even though the TVL of each strategy has already been queried within the `read_all_strategies` function in line `534`. That causes unnecessary queries.

Recommendation

We recommend summing up `strategy.tvl` in line `544` instead.

17. Unnecessary storage of user info by base token within apollo-base-strategy

Severity: Informational

In the apollo-base-strategy contract in `contracts/apollo-base-strategy/src/state.rs:56` and `65`, user info is stored in a multilevel bucket, where each entry is accessed by both the user's and the base token's canonical address. Since there is one strategy contract instance per base token and since every CosmWasm contract uses their own storage namespace, using the base token as an identifier is not necessary.

Recommendation

We recommend storing user info only by a user's canonical address. That simplifies the code and reduces computation.

18. Unused factory contract field in collector config

Severity: Informational

The field `factory_contract` of the collector's `Config` in `contracts/collector/src/contract.rs:23` is unused.

Recommendation

We recommend removing unused config entries to simplify the code and reduce gas cost.

19. Unused terraswap factory contract field in autocompound strategy config

Severity: Informational

The field `terraswap_factory` of the collector's `Config` in `contracts/strategies/autocompound/src/state.rs:13` is unused.

Recommendation

We recommend removing unused config entries to simplify the code and reduce gas cost.

20. Unused constants in factory state

Severity: Informational

The constants `STRATEGY_KEY` and `STRATEGY_COUNT_KEY` in `contracts/apollo-factory/src/state.rs:22` and `23` are unused.

Recommendation

We recommend removing unused constants to simplify the code and reduce gas cost.