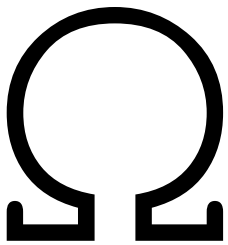
Altitude Parallel Farming

Final Audit Report

January 22, 2025



Team Omega

teamomega.eth.limo

Summary

Altitude has asked Team Omega to audit their smart contract system.

We found a number of issues, which are described in the current report. Specifically, we found **one high severity issue** - these are issues that can lead to a loss of funds, and are essential to fix. We classified **one** issue as “medium” - these are issues we believe you should definitely address, even if they do not lead to loss of funds. In addition, **3** issues were classified as “low”, and **10** issues as “info” - we believe the code would improve if these issues were addressed as well.

Severity	**Number of issues**	**Number of resolved issues**
High	1	1
Medium	1	1
Low	3	2
Info	11	10

Scope of the Audit

We audited the code from the following Github PR:

`https://github.com/refi-network/protocol-v1-audit/pull/11/`

Which concerns the diff between commit `5b3026b758aed1e61f8c1ba8e7b46caec985c0ad` and `4ce09aa7812e7e7d50582c4373d48f21b640d7a4`

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Resolution

Most issues were addressed in the following commit:

`b7710bf81b57c487d093c0bc48a8fafe8b1997c3`

We reviewed the changes and updated the report accordingly.

Audit History

In the current audit, we focused mostly on the changes between the two commits above. We have reviewed several earlier iterations of the code base in the past years:

- In 2022, we audited a first version of the code, the report is here: https://github.com/OmegaAudits/audits/blob/main/202207-Altitude-v1.0.pdf
- We audited a second iteration of the code in 2023 https://github.com/OmegaAudits/audits/blob/main/202310-Altitude-v1.1.pdf
- We audited a new integration with Morpho (in version 1.1) in the summer of 2024 https://github.com/OmegaAudits/audits/blob/main/202408-Altitude-morpho-integration.pdf

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Anyone can cause the vault calculations to break by paying the vault debt directly to the lender [info] [not resolved]

The external lending protocols do not always limit who can repay the debt of a user, which means anyone could repay the debt the vault takes in the external lending protocol on behalf of its users. Such a case, when the repayment is done in full, or when a significant portion of the vault's debt is repaid multiple times, could cause the calculations of the vault to overflow and halt its further operation. However, as there is no clear benefit to the attacker beyond disruption, and since this will essentially reward users by wiping out their debts, we believe this issue does not constitute an immediate threat to the system.
Recommendation: For now, we believe it is safe to leave the code as is, as such an attack will be both costly, and eventually beneficial to the users, as their debt gets paid for them.
Severity: Info
Resolution: This issue was not resolved. The developers are aware of the possibility of this happening.

G2. Upgrade Solidity version from 0.8.20 to at least 0.8.23 [info] [resolved]

The latest Solidity version contains numerous improvements and optimizations, and three bug fixes https://docs.soliditylang.org/en/latest/bugs.html. Although the changes do not directly affect the current contracts, it is good practice to upgrade to a newer, if not the latest version.
Recommendation: Use a more recent Solidity version to compile the files
Severity: Info
Resolution: This issue was resolved as recommended.

SwapStrategy.sol

SS1. getMaximumAmountIn returns wrong result when slippage is 0 or very low [low] [resolved]

The `getMaximumAmountIn` function has a check that returns 0 in case the slippage did not have an effect on the quote amount. This is meant for cases where the quote amount is so small the slippage has no effect on it, but it can also happen in case the `slippage` is set to 0, or in some case just to a very low number. In this case, you should just return the quote amount.
Recommendation: Remove the check and return the `quoteAmount`.
Severity: Low
Resolution: This issue was resolved. The check now returns `quoteAmount` + 1.

SS2. Avoid code duplication when getting the asset price [info] [resolved]

The `getMaximumAmountIn` function could use the `_getPrice` helper function to get the price of the asset, as currently it is just using a duplication of its code.
Recommendation: Use the `_getPrice` function in `getMaximumAmountIn` like it's used in the `getMinimumAmountOut` function.
Severity: Info
Resolution: This issue was resolved as recommended.

FarmDropStrategy.sol

FDS1. Unnecessary duplicate modifier in withdraw [info] [resolved]

The `withdraw` function has the `onlyDispatcher` modifier, but since `super.withdraw` already triggers this modifier, it could be removed from the function (like in `emergencyWithdraw`).
Recommendation: Remove the `onlyDispatcher` modifier from FarmDropStrategy's `withdraw`.
Severity: Info
Resolution: This issue was resolved as recommended.

FarmStrategy.sol

FS1. Use msg.sender instead of storage variable could save gas [info] [resolved]

The `emergencySwap` function sends the swapped asset to the farmDispatcher, but since the caller must be `farmDispatcher`, some gas can be saved by transferring to `msg.sender` instead of `farmDispatcher`.
Recommendation: Use `msg.sender` instead of `farmDispatcher`.
Severity: Info
Resolution: This issue was resolved as recommended.

TokensFactory.sol

TF1. Debt token is given wrong name on deployment [info] [resolved]

The `_deployToken` function in the `TokensFactory` is used for deploying both the supply and debt tokens, but has the name for the token deployed hard-coded as "... v1 Supply Token", which names both the supply and debt token in the same name.
Recommendation: Update the function to correctly name each token, you could also update the variables names in the function to be more generic.
Severity: Info
Resolution: This issue was resolved as recommended.

TF2. UpdateProxyAdmin event is not emitted on first setting of the proxy admin [info] [resolved]

The `proxyAdmin` is set in the contract's constructor, but a corresponding `UpdateProxyAdmin` event is not emitted, which could cause a discrepancy when using the event for a UI or indexing.
Recommendation: Emit the `UpdateProxyAdmin` event in the contract's constructor.
Severity: Info
Resolution: This issue was resolved as recommended.

HarvestableManager.sol

HM1. ClaimedRewards is potentially emitted with wrong debtRepayed amount [low] [resolved]

The `ClaimedRewards` event is emitted with a `debtRepayed` which indicates the amount that was used to repay the user's debt. However, when the total amount of user rewards are not sufficient to cover the debt, the event still emits the full debt amount as repaid, instead of only the actual amount repaid.
Recommendation: Below line 423, set the `debtBalance` variable to be equal to the `amountTotal`.
Severity: Low
Resolution: This issue was resolved as recommended.

InterestToken.sol

IT1. Should use state instead of storage variable in initialize [info] [resolved]

In the initialize function, the interestIndex is set to `MATH_UNITS`, which is a storage variable whose value is set in the `initialize` function from a parameter of the function. It could save gas to use the function parameter also to set the value of `interestIndex`.
Recommendation: In the `initialize` function, set `interestIndex` to `mathUnits` instead of `MATH_UNITS`
Severity: Info
Resolution: This issue was resolved as recommended.

FarmDispatcher.sol

FD1. Funds are not being repaid in case all strategies have reached their max deposits [medium] [resolved]

In case all strategies of the dispatcher are at their max deposit limits, the rest of the funds will simply sit idle in the dispatcher. This is very inefficient as the system continues to pay interest on those funds which are not used to generate yield. In such a case the system should stop borrowing and use those funds to repay its loan.
Recommendation: Avoid holding funds beyond the maximum total deposits the dispatcher can have, and instead pay them back to the lending strategy.
Severity: Medium
Resolution: This issue was resolved as recommended.

FD2. Funds are not being re-deposited when a strategy max amount changes [low] [not resolved]

The `setStrategyMax` allows the owner to set the max amount to deposit in a strategy. In case a strategy has more funds deposited than the new max, the funds will just remain idle in the dispatcher until the dispatch is called again. It would be more efficient to immediately re-deposit funds in other strategies.
Recommendation: In case funds are withdrawn from a strategy which reached its new max, the dispatch should be triggered to re-deposit those funds.
Severity: Low
Resolution: This issue was acknowledged by the developers, who chose not to address the issue.

FD3. Implement a function for collecting all rewards [info] [resolved]

The system now has its reward collection completely decoupled and independent, but there is no function to claim all rewards from all strategies in a single action. The system could benefit from such an option both to save gas, and in case you'd like to eventually reintroduce processing rewards as part of the harvest.
Recommendation: Add a function (in the dispatcher or other place), which calls recognise rewards for all farming and lending strategies of the vault, or at least a list of addresses of strategies to call for.
Severity: Info
Resolution: This issue was resolved as recommended.

FD4. Unnecessary duplicated modifier in addStrategies [info] [resolved]

The `addStrategies` function has the `onlyOwner` modifier, but since the `addStrategy` function which it calls already triggers this modifier it could be removed from the function.
Recommendation: Remove the `onlyOwner` modifier from `addStrategies`.
Severity: Info
Resolution: This issue was resolved as recommended.

CurveV2Dispatcher.sol

CD1. swapOutBase can be optimized [info] [resolved]

The function `swapOutBase` first transfers `amountInMaximum` tokens to the strategy, then calculates how much is actually needed for the swap, does the swap with these tokens, and finally refunds the user with the difference.
Recommendation: This code can be optimized by transferring only `amountIn` tokens to the strategy - in this way, no refund is necessary.
Severity: Info
Resolution: This issue was resolved as recommended.

StrategyMorphoV1.sol

SM1. Swaps in _recognizeRewardsInBase will fail [high] [resolved]

In `_recogniseRewardsInBase`, the amount passed to the _swap function is `type(uint256).max`

```
      _swap(rewardAssets[i], asset, type(uint256).max);
```

This value is then passed on to the swapStrategy. However, the swapStrategy expects the actual amount to be swapped, and will revert when receiving such a high value. This will make it impossible to receive the rewards.
Recommendation: Just as in `MorphoVault`, add the following lines to the swap function:

```
if (amount == type(uint256).max) {
      amount = IERC20(inputAsset).balanceOf(address(this));
}
```

Severity: High
Resolution: This issue was resolved as recommended.