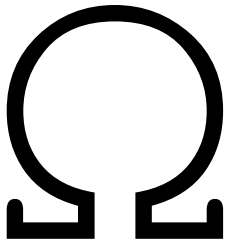
Altitude Morpho Integration

Final Audit Report

August 21, 2024



Team Omega

Teamomega.eth.limo

Summary

Altitude has asked Team Omega to audit the contracts that define the behavior of their integration with Morpho.

We found **no high severity issues** (issues that can lead to a loss of funds, and are essential to fix) and we classified **1** issue as “medium”. We did classify **2** issues as “low”, and an additional **5** issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

Severity	Number of issues	Number of resolved issues
High	0	0
Medium	1	1
Low	2	1
Info	5	3

Scope of the Audit

The scope of the audit concerns the `StrategyMorphoV1.sol` and `MorphoVault.sol` contracts. The code in scope developed here:

`https://github.com/refi-network/protocol-v1-audit/tree/morpho-lender`

We audited commit hash `295e5c4759ac7a08ad90a52528fc2dcd20b453cb`

First Report

We submitted a first report on August 7, 2024 with the issues described below.

Resolution

The issues were subsequently addressed in the following commit:

`aed7fcde094dbd902955e7ba17fed8ef8538154f`

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.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. The tools have detected a number of low severity issues, concerning mostly the variables naming and external calls, were found. We have included any relevant issues below in the appropriate parts of the report.

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. Skim functionality is more permissive than necessary [info] [not resolved]

The `skim` function on both `MorphoVault` and `StrategyMorphoV1` allow the owner to transfer any tokens out of the contract. While transferring tokens like this is currently needed for the system to function, it seems more permissive than necessary, since it allows to also take any supply, borrow, or farm asset in the contract, which gives the owner more power than what seems intended.
Recommendation: You could consider disallowing the `skim` function to withdraw the supply, borrow, and farm assets to slightly reduce the power it grants over the strategies.
Severity: Info
Resolution: The issue was acknowledged by the project, but the functionality deemed necessary, so the issue is not resolved.

MorphoVault.sol

MV1. Utilize ERC4626 maxWithdraw and maxRedeem functions [medium] [resolved]

In lines 102 -103 and 168, 173, the code makes unnecessary “double” calls to the `convertToAssets` and `balanceOf` functions of the Morpho vault which could be replaced with a single call to the `maxWithdraw` function, and in lines 111, 135, and 153, the call to `balanceOf` of Morpho vault should more appropriately call the `maxRedeem` function of the vault.
More importantly though, using `balanceOf` instead of `maxWithdraw` and `maxRedeem` functions could try to withdraw/ redeem more than is actually available, since the maximum amount might be lower than the full balance (see this line in MetaMorpho). This could cause the calls to revert, since they will try to withdraw more than the max, and so the vault functionalities depending on full redeem/ withdrawal are at risk of failure.
Recommendation: Utilize the Morpho vault’s `maxWithdraw` and `maxRedeem` functions instead of the `balanceOf` and `convertToAssets` functions.
Severity: Medium
Resolution: The issue was resolved.

MV2. _farmWithdraw might withdraw from the vault more than necessary [low] [not resolved]

If the amount locked in the Morpho vault is less than the amount requested, the `_farmWithdraw` will call `_recogniseRewardsInBase` to try and get more tokens for the withdrawal, then it will withdraw its entire balance from the Morpho Vault. However this might be too much, since it does not account for the additional tokens received in `_recogniseRewardsInBase`. So even in case the call to `_recogniseRewardsInBase` has added enough tokens so that it is no longer necessary to withdraw the farm's entire position, the farming strategy will still withdraw its entire position, which is more than the necessary amount to complete the farm withdrawal process.
Recommendation: After the call to `_recogniseRewardsInBase`, you should reduce the amount that was added from the amount to withdraw, then withdraw either just the amount necessary, or in the case the vault still does not have enough tokens to cover for the entire withdrawal, withdraw everything.
Severity: Low
Resolution: The issue was not resolved in the provided commit. The team acknowledged the issue and stated that they are working on a solution.

MV3. Mark state variables as immutable when possible [info] [resolved]

The `morphoVault` and `farmAsset` state variables in `MorphoVault` are set in the constructor and cannot be changed afterwards, and so can be marked `immutable`.
Recommendation: Mark these state variables as `immutable`.
Severity: Info
Resolution: The issue was resolved as recommended.

StrategyMorphoV1.sol

SM1. getInBase returns wrong values if the assets passed to it are not the supply and borrow assets [low] [resolved]

The `getInBase` function accepts any address as the `fromAsset` and `toAsset`, but will only work if the assets used are the supply or borrow assets.
Recommendation: The function should ensure the assets passed to it are the supply and borrow assets, and revert otherwise.
Severity: Low
Resolution: The issue was resolved as recommended.

SM2. Strategy expects the Morpho market to use supply and borrow assets without verifying it does [info] [resolved]

The strategy expects that the Morpho Blue market it is set to will use its supply and borrow assets, but the constructor does not verify this, which may lead to issues using it.
Recommendation: Verify in the constructor the Morpho market used matches the supply and borrow assets of the strategy.
Severity: Info
Resolution: The issue was resolved as recommended.

SM3. repay may approve more than necessary [info] [resolved]

The `repay` function approves the entire amount passed to it for the Morpho Blue to use for repayment, but the actual amount repaid might be lower, potentially leaving an unused approved amount.
Recommendation: Approve only the amount actually used in repayment instead of the maximum amount to repay, or set the approval to 0 at the end of the procedure.
Severity: Info
Resolution: The issue was resolved as recommended. The approval is set to 0 at the end.

SM4. supplyPrincipal is set twice on withdrawAll [info] [resolved]

The call to `_withdrawAll` sets the `supplyPrincipal = supplyBalance()`, but the `withdrawAll` function will call the `_updateState` which will do the exact same thing, setting it twice to the same value.
Recommendation: Remove the setting of `supplyPrincipal = supplyBalance()` in line 150.
Severity: Info
Resolution: The issue was resolved as recommended.