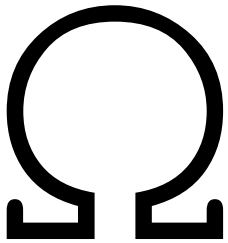

Team Omega

https://teamomega.eth.limo

Altitude

Audit of “March Version”

Final Report

May 22, 2025

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 **2** issues as “medium” - these are issues we believe you should definitely address, even if they do not lead to loss of funds. In addition, **2** issues were classified as “low”, and **6** 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	2	2
Low	2	2
Info	6	5

Team Omega
Team Omega (https://teamomega.eth.limo/) specializes in Smart Contract security audits on the Ethereum ecosystem.

Altitude
Altitude (https://www.altitude.fi/) is developing a system for managing and optimizing loans in the Ethereum ecosystem.

Method

The audit report is compiled on the basis of the findings from different auditors (Jelle and Ben). The auditors work independently. Each of the auditors has many years of experience in developing smart contracts on Ethereum, and has a a pluri-annual experience with developing contracts for DAOs

Scope of the Audit
Team Omega has previously audited the development versions of the code-base prior to the code being made open source. The scope of this audit looks at any changes made since the last audit as well as verifying the open source code aligns with the previously audited code.

The last commit we audited was `b7710bf81b57c487d093c0bc48a8fafe8b1997c3` , which resulted in the following report:
https://github.com/OmegaAudits/audits/blob/main/202501-Altitude-parallel-farming.pdf

The first PR applies linter rules to the code base, which results in commit hash `e815b737b21bf0fd3c0a8fa70f7cb37657e4742a`. We reproduced the same result and are confident that this PR only introduces syntactic changes - but no semantic ones.

The second PR - and commit hash `ab78369aff24a46d4c1aa60712639fb55cdbc9d2` - is the main focus of this audit. There are a number of new contracts here (a total of around 450 nsloc), and a number of changes in existing contracts (233 lines added and 2523 lines removed). We describe our findings below.

Liability
This audit report is presented on a best-effort basis. In particular, the audit does not represent any guarantee that the software is free of bugs or other issues, nor is it an endorsement by Team Omega of any of the functionality implemented by the software.

Audit History

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
- We audited an extension of the system in January 2025, which resulted in the following report:
  https://github.com/OmegaAudits/audits/blob/main/202501-Altitude-parallel-farming.pd
- The preliminary version of this report was created on April 11, 2025

These audits represent a continuous history - i.e. we have reviewed the entire code base and all subsequent changes.

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.

Resolution

After we delivered a preliminary report, almost all issues were subsequently addressed by the Altitude Team in commit hash `321b5a1d24743ce6342762422d1ac31dbd594af0 available at` https://github.com/altitude-fi/altitude-v2/commit/321b5a1d24743ce6342762422d1ac31dbd594af0

We reviewed the changes and updated the issues below.

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

Ingress

I1. An attacker can block withdrawals and borrows at any time [high] [resolved]

The functions `validateWithdraw` and `validateBorrow` and `validateClaimRewards` are `public` and not access protected. These functions call `withinRateLimit`, which will update the rate limit.
An attacker can call these functions with arbitrary values (the crucial line here is `_rateLimit.available -= amount`) and block any withdraw, borrow or claim rewards actions.
Recommendation: Make a clear distinction between "view" functions and state-changing functions. In other words, have a public view that checks if withdrawing (etc) is possible and allowed, and make the state-changing functions access-controlled.
Severity: High
Resolution: The issue was resolved as recommended. The functions mentioned now implement access control.

BorrowVerifier

BV1. An attacker can DoS borrowOnBehalfOf [low] [resolved]

`BorrowVerifier.verifyAndBurnNonce` is a public function that can be called by anyone. It checks a signature and augments the corresponding nonce of the signer of the message.
A call to `borrowOnBehalfOf`, which uses this function, can be frontrun and the nonce burnt by calling `verifyAndBurnNonce` with the signature passed to `borrowOnBehalfOf`. This allows an attacker to do a griefing attack and DoS any calls to `borrowOnBehalfOf`.
Recommendation: Either add access control to the `verifyAndburnNonce`, so it can only be called from the vault, or (probably easier) define the function `internal` and include it in `VaultCore`.
Severity: Low
Resolution: The issue was resolved as recommended. The `verifyAndBurnNonce` function mentioned now implements access control and can only be called by the vault.

StrategyPendleBase

SPB1. Slippage variable can be marked immutable [info] [resolved]

The `slippage` variable can be marked immutable, as there is no setter to change it.
Recommendation: Either mark the variable immutable, or introduce a setter.
Severity: Info

Resolution: The issue was resolved as recommended. A setter function was added for the slippage.

SPB2. Twap duration is shorter than what Pendle recommends [info] [resolved]

The default value for the `twapDuration` variable is set to 300 (seconds). This is shorter than what Pendle recommends - the documentation says that "The recommended duration is 15 mins (900 secs) or 30 mins (1800 secs), but it can vary depending on the market."
Recommendation: Set the default value to the recommended value.
Severity: Info
Resolution: The issue was resolved as recommended. The `twapDuration` value was updated to the recommended value of 1800.

SBP3. Contract contains duplicated functions from other strategies [info] [resolved]

The `_swap`, `_emergencySwap`, and `setRewardAssets` are the same as in the other strategies, and the code could be shared in a base contract instead of duplicated across strategies.
Recommendation: Move common code to a base contract inherited by the strategies.
Severity: Info
Resolution: The issue was resolved as recommended. The functions with common implementation were moved to the `FarmStrategy` base contract.

StrategyPendlePT

SPP1. Consider using getPtToAssetRate instead of getPtToSyRate for TWAP rate [medium] [resolved]

At the moment, when converting farmAsset to PT token in the `_deposit` function, or vice versa in withdraw, the code calls `getPtToSyRate` on the oracle to get the TWAP price which is used for slippage protection. In the case where SY tokens are 1-1 wrappers of the farm token, this is fine, as the PT price in SY is the same as the PT price in farmToken.
However, as per the documentation in https://docs.pendle.finance/Developers/Contracts/StandardizedYield, some SY tokens do not have such a 1-1 correspondence. This specifically holds for SY tokens that wrap stable coins, which presumably are the main target use case for the current system. For those cases, it is recommended to use `getPtToAssetRate`.
The same remark holds for the use of `getLpToSyRate` in `StrategyPendleLP`.
Recommendation: It is probably safer to always use `getPtToAssetRate` instead of `getPtToSyRate`
Severity: Medium

Resolution: The team now consistently prices all assets in SY tokens.

SPP2. Unused slippage parameter when converting tokens [info] [not resolved]

In `_deposit`, tokens are swapped for PT tokens using the Pendle DEX router - specifically, a call to `router.swapExactTokenForPt`. This function takes a parameter called `minPtOut`, which provides some slippage protection. This parameter is not used - instead, the swap is executed, and on the resulting output, a check is added comparing the output to the oracle price, using `_validateRate`.

This pattern works, but some gas can be saved by using the built-in slippage protection, which will make the transaction abort earlier.

A similar pattern is used in other places, like `_withdraw` and `_exitExpiredMarket`, and in `StrategyPendleLP._withdrawYT` and `StrategyPendleLP._withdrawLP`
Recommendation: Use built-in slippage protection when using the Pendle Router.
Severity: Info
Resolution: The issue was not resolved. The team opted to leave the code as is for consistency with other contracts.

StrategyPendleLP

SPL1. Withdrawals favor LP tokens over YT tokens, skewing the risks [low] [resolved]

`StrategyPendleLP` invests tokens in two assets: LP tokens that represent a share in the liquidity pool (i.e. a varying proportion of SY and PT tokens) and an amount of YT (yield tokens) of the same number as the amount of PT tokens obtained at the time of the original deposit. This basket of tokens can be expected, at least approximately, to keep a stable value relative to the underlying token (and is guaranteed to do so at the expiry date).
However, in the `_withdraw` function, first all LP tokens are withdrawn, and only if there are no such tokens anymore, will YT be sold. This means that after each withdrawal, the basket will be further skewed towards YT tokens. Because the deposit logic does not contain a similar asymmetry, during the lifetime of the vault, users will be exposed more and more to the YT leg of the investment, which is generally much more volatile and holds no guarantee for any future value. This is a risk profile that is considerably different then the other investment strategies that Altitude deploys, and not what a user may expect.
Recommendation: On `withdrawal`, withdraw a proportionate amount of LP and YT tokens. Or, alternatively, simplify the strategy and invest in LP tokens only.

Resolution: The team acknowledged the issue. They observe that there are other strategies available for investing in PT tokens only that are less risky, and explained that the implementation is a conscious decision.

RebalanceIncentivesController

RIC1. The constructor does not emit an UpdateRebalanceThresholds event [info] [resolved]

The constructor is updating the thresholds, but does not emit a corresponding event.
Recommendation: emit a `UpdateRebalanceThresholds` event, or just use the `setThresholds` (with helper function).
Severity: Info
Resolution: The issue was resolved as recommended. The constructor now uses a helper function and emits an `UpdateRebalanceDeviation` event.

RIC3. setThreshold and UpdateRebalanceThresholds names should be updated [info] [resolved]

Since the threshold variables were renamed to deviations, their setter function and event should match their new name to avoid confusion.
Recommendation: Rename `setThreshold` and `UpdateRebalanceThresholds` to use deviation instead of threshold.
Severity: Info
Resolution: The issue was resolved as recommended. The function and event were renamed..

Fixes from older reports

Altitude has also implemented a number of fixes for issues that we mentioned in previous reports. These issues are listed below.

The relevant reports are:

OM-1: https://github.com/OmegaAudits/audits/blob/main/202207-Altitude-v1.0.pdf
OM-2: https://github.com/OmegaAudits/audits/blob/main/202310-Altitude-v1.1.pdf

OM-3: https://github.com/OmegaAudits/audits/blob/main/202408-Altitude-morpho-integration.pdf
OM-4: https://github.com/OmegaAudits/audits/blob/main/202501-Altitude-parallel-farming.pdf

Harvested rewards are not reserved for users - Source: OM-1 as HV1b, OM-2 as HM3

HV1b. Harvested tokens claimable by as rewards are not reserved for users [medium] [resolved]

When claiming rewards using `claimRewards`, the user's rewards are withdrawn from the farming strategy to be sent to the user. Operations such as `rebalance` (especially if the `targetThreshold` is low or equal to 0) and `_repayVaultDebt` move funds out of the farming strategy without reserving funds to pay out user rewards or the vault reserve. These functions will pay off the vault's debt, and will leave any remaining `borrowUnderlying` tokens in the vault's balance. This means that when closing the farm's position, users will not be able to claim their rewards anymore - `claimRewards` will revert.
Recommendation: Respect the values of `userClaimableEarnings` and `vaultReserve` when rebalancing or repaying the vault debt. Or, alternatively (but this would require a more extensive revision), rewrite the `withdrawReserve` and `claimEarnings` functions so they will not be affected by rebalancing.
Resolution: The issue was resolved - much of this code referenced in this issue (which is from our report from 2022) was rewritten and the issue does not apply anymore.

HM3. claimRewards may revert and will not pay off existing debt [low] [resolved]

The function `claimRewards()` takes the user's claimable earnings and uses that to pay off the debt of the user. This is an internal adjustment of balances, and no tokens are actually transferred.

If there are additional rewards that remain, these will be withdrawn from the farming strategy and sent to this user. If this fails (for example because the farming strategy does not have enough tokens to cover the debt) the call to `claimRewards` will revert.

This is not desired behavior. The user's debt should be paid off with the rewards, even if the farming strategy does not have enough tokens available to send the remaining rewards

Recommendation: Do not revert in case the farming strategy cannot cover the rewards. Also, use the buffer here as you do in other places where tokens are moved from and into the farming strategy.
Severity: Low
Resolution: The issue was resolved - if the farming strategy does not contain enough funds to pay for the rewards, the entire balance is withdrawn, but the function does not revert.

Remove unused state variables [low] - Source: OM-1 as HM5

HM5. Remove unused state variables [low] [resolved]

The `UserHarvestData` that is saved for each user contains multiple variables that are written to, but never read from. These include the `supplyIndex`, `borrowIndex`, `harvestEarnings`, and `harvestCosts`. These variables could just be emitted in events instead of saved to storage, or even entirely removed to save gas.
Recommendation: Remove the `UserHarvestData´s` `supplyIndex`, `borrowIndex`, `harvestEarnings`, and `harvestCosts` variables, and emit their values in events if needed.
Severity: Low
Resolution: The issue was resolved as recommended.

Lock() modifier not applied consistently - Source: OM-1 as VV1

VV1. lock() modifier not applied consistently [low] [resolved]

The `lock` modifier defined in `VaultETH` and `VaultERC20` is a simple reentrancy guard: a first function call sets a flag so that any other calls of functions modified by the lock modifier will fail. The modifier is applied to a number of state-changing public functions such as `deposit`, `withdraw` and `borrow`.
The lock modifier provides some protection against reentrancy, but it is not applied consistently. For example, functions such as `claimRewards`, `transfer` or `liquidateUsers` do not have these locks set, and so the current implementation of the lock mechanism does not avoid re-entrancy when using these functions.
Recommendation: Use the lock more extensively and apply it to all public state-changing functions. Consider as well to use OpenZeppelin's `ReentrancyGuard` contract instead of defining your own lock function.
Severity: Low
Resolution: The issue was resolved as recommended. OpenZeppelin's `ReentrancyGuard` contract is now used.

Use claimRewardsToSelf to claim rewards - Source: OM-1 as SA10

SA10. Use claimRewardsToSelf to claim rewards [info] [resolved]

In line 255, the rewards are claimed by the contract to its own address by calling the `incentivesController.claimRewards`, but since the contract claims the rewards to its own address, it is more appropriate to use the `incentivesController.claimRewardsToSelf` function which claims rewards to the caller, instead of asking for an extra address parameter to send the rewards to.
Recommendation: In line 255, change the call from `incentivesController.claimRewards` to `incentivesController.claimRewardsToSelf`, and remove passing the `address(this)` as parameter in line 258.
Severity: Info
Resolution: The issue is not relevant anymore, as `claimRewardsToSelf` was removed

_farmWithdraw might withdraw more than necessary - Source: OM-3 as MV2

MV2. _farmWithdraw might withdraw from the vault more than necessary [low] [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 is not relevant anymore, as rewards are not recognized anymore as part of the withdraw process.

Mark functions as external where possible - Source: OM-1 as SA12

SA12. Mark functions as external where possible [info] [resolved]

The `deposit`, `withdraw`, `withdrawAll`, `borrow`, and `repay` functions are declared `public` but are not used from within the contract, and so could be marked as `external`.

Recommendation: Mark `public` functions that are not used within the contract as `external`.
Severity: Info
Resolution: The issue was resolved as recommended

Skim functionality more permissive than necessary - Source: OM-3 as G1

G1. Skim functionality is more permissive than necessary [info] [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 function is now more restricted in what it can withdraw.

Anyone can cause the vault calculations to break by paying the vault debt directly to the lender - Source OM-4 as G1

G1. Anyone can cause the vault calculations to break by paying the vault debt directly to the lender [info] [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 resolved in https://github.com/altitude-fi/altitude-v2-private/pull/47 - in case the actual amount borrowed is less than expected, the extra room to borrow can be used to send borrowed funds to the vault reserve, from which they can be distributed to the users.