# yAcademy Morpho review

**Review Resources:**
A protocol yellowpaper and documentation were provided

**Residents:**

- Peep (xiangan.eth)

# Table of Contents


**yAcademy Morpho review**
- Table of Contents
- Review Summary
- Scope
- Code Evaluation Matrix
- Findings Explanation
- Critical Findings
  - 1. Critical - P2P market can be drained if P2P gets toggled off after matching.
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- High Findings
  - 1. High - Delta manipulations in certain scenarios (notably flashloaning)
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- Medium Findings
  - 1. Medium - Governance contract needs receive()
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- Low Findings
  - 1. Low - MarketsManager.sol setReserveFactor does not require market as created
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
  - 2. Low - View function may be called on nonexistent pool address
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- Gas Savings Findings
  - 1. Gas - Double Linked List implementation
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- Informational Findings
  - 1. Informational - Aave contracts may require resizing, may require redeployment because it exceeds max memory size.
    - Impact
    - Recommendation
    - Developer Response
  - 2. Informational - Change "weth" to wEth: noncompiling gas report.
    - Developer Response
  - 3. Informational - Typo in inline documentation
    - Developer Response
  - 4. Informational - MarketsManager.sol L283 & L314 control flow decision made on block.timestamp.
    - Proof of concept
    - Impact
    - Recommendation
    - Developer Response
- Final remarks
- Appendix and FAQ


# Review Summary

**Morpho**

The Morpho protocol is a Pareto-improving interest-rate mechanism built on top of existing pool-based lending protocols.

The dev branch of the Morpho Repo was reviewed over 8 days. 1 day was used to create a report of the findings. Only the compound contracts were covered. The contracts were reviewed from April 28 to May 9. The code was reviewed by 1 resident for a total of 42 man hours. The repository was under active development during the review, but the review was limited to one specific commit.

## Scope

Code Repo
Commit

The commit reviewed was 9123b34db1bfe39f91f619de61a69d34a9f339e5. The review covered the entire repository at this specific commit but focused on the contracts directory.

After the findings were presented to the Morpho team, fixes were made and included up to commit 17f2d91a8b4c43d5287ad85f9fc47d9449d61c13. Fixes were reviewed to confirm whether they remedied the original finding, but a new security review was not performed on the revised code (e.g. to determine whether new vulnerabilities were introduced by the fixes).

The review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAcademy and the residents make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAcademy and the residents do not represent nor imply to third party users that the code has been audited nor that the code is free from defects. By deploying or using the code, Morpho and and users agree to use the code at their own risk.

## Code Evaluation Matrix

| Category | Mark | Description |
| --- | --- | --- |
| Access Control | Good | Access controls are applied where needed. Ownable is appropriately used to limit function calls to respective permissions, and view functions are made public for liquidation purposes. |
| Mathematics | Good | Solidity 0.8.13 is used, which provides overflow and underflow protect. Unchecked code was checked in this audit to yield no anomalies. No low-level bitwise operations are performed. There was no unusually complex math, except perhaps some of the calculations with current interest rates based on market growth factor. |
| Complexity | Medium | The delegatecall structure and sometimes duplicate function names can make the code hard to follow at times. The core complexity also lies in the security implications of Compound itself and what happens in the event of market anomalies and manipulations in Compound. |
| Libraries | Good | Only basic Open Zeppelin contracts such as SafeERC20, Ownable, ReentrancyGuard, and Math are imported, no other external libraries are used. Fewer and simpler external dependencies is always a plus for security. |
| Decentralization | Poor | The governance contract is successfully implemented but needs work integrating with Zodiac. Additionally, further due diligence has to be done for that integration, along with the fact that little coverage on test for governance contracts exists at the moment. |
| Code stability | Average | Changes were reviewed at a specific commit and the scope was not expanded after the review was started. However, development was ongoing when the review was performed so the code was not fully frozen, which means deployed code may vary from what was reviewed. |
| Documentation | Good | Comments existed in many places, and mostly clarified what the code did. Some documentation had to be added to distinguish between functions with duplicate names, as well as fixing some typos. |
| Monitoring | Good | Events were added to all important functions that modified state variables. |
| Testing and verification | Good | All tests were passing after a small fix to a typo, and test coverage was very expansive. |

## Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements,
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

# Critical Findings

## 1. Critical - P2P market can be drained if P2P gets toggled off after matching.

### Proof of concept

The following code, which subtracts the supplyBalance of a user upon withdrawal, does not trigger if P2P gets turned off after a user has already supplied tokens to the pools.

```
if (vars.remainingToWithdraw > 0 && !marketsManager.noP2P(_poolTokenAddress)) {
            supplyBalanceInOf[_poolTokenAddress][_supplier].inP2P -= CompoundMath.min(
                supplyBalanceInOf[_poolTokenAddress][_supplier].inP2P,
                vars.remainingToWithdraw.div(supplyP2PExchangeRate)
            ); // In p2pUnit
            updateSuppliers(_poolTokenAddress, _supplier);
```

### Impact

This causes the entire pool to be drainable, since the attacker would then be able to continue calling withdraw on the maximum balance they had before their withdrawal.

### Recommendation

It's possible to split the condition into two parts, such that the only check here is

```
if (vars.remainingToWithdraw > 0) {
            supplyBalanceInOf[_poolTokenAddress][_supplier].inP2P -= CompoundMath.min(
                supplyBalanceInOf[_poolTokenAddress][_supplier].inP2P,
                vars.remainingToWithdraw.div(supplyP2PExchangeRate)
            );
```

and then implementing the other part of the logic for when governance is turned off some other way.

### Developer Response

Fixed.

# High Findings

## 1. High - Delta manipulations in certain scenarios (notably flashloaning)

### Proof of concept

In a scenario where some borrowers are matched P2P (earning a low borrow rate), many are still on the pool and therefore in the pool queue (earning a worse borrow rate from Comp/Aave). An attacker supplies a huge amount, creating a P2P credit line for every borrower. (They can repeat this step several times if the max iterations limit is reached.)
The attacker immediately withdraws the supplied amount again. The protocol now attempts to demote the borrowers and reconnect them to the pool. But the algorithm performs a "hard withdraw" as the last step if it reaches the max iteration limit, creating a borrower delta. These are funds borrowed from the pool (at a higher borrow rate) that are still wrongly recorded to be in a P2P position for some borrowers. This increase in borrow rate is socialized equally among all P2P borrowers. (reflected in an updated p2pBorrowRate as the shareOfDelta increased.)
The initial P2P borrowers earn a worse rate than before. If the borrower delta is large, it's close to the on-pool rate.
If an attacker-controlled borrower account was newly matched P2P and not properly reconnected to the pool (in the "demote borrowers" step of the algorithm), they will earn the better P2P rate than the on-pool rate they earned before.

### Impact

High, this can cause P2P borrowers to earn at a worse rate than before the attack.

### Recommendation

In withdraw() of logic.sol, putting in an update for P2P delta immediately when hard-withdraw is triggered.

```
delta.borrowP2PAmount -= unmatched.div(
                marketsManager.borrowP2PExchangeRate(_poolTokenAddress)
            );
```

### Developer Response

Acknowledged & working solutions.

# Medium Findings

## 1. Medium - Governance contract needs receive()

Currently, the PositionManagerGovernance for compound does not have a receive() function to receive protocol fees, in addition to claimToTreasury sending the whole balance of PositionManagerGovernance.sol to the treasuryVault.

### Proof of concept

No function in PositionManagerGovernance.sol is payable, nor is there a receive function to receive ETH.

### Impact

Medium, transactions revert (also given no fallback function) when sending funds to governance contracts.

### Recommendation

Implement `receive() external payable` in contract

### Developer Response

Fixed since the contracts have been combined into Morpho.sol

# Low Findings

## 1. Low - MarketsManager.sol setReserveFactor does not require market as created

Consider putting a modifier that requires the target pool token to already have a market that is created. Or perhaps even causing updateP2PExchangeRates to revert if the target market does not exist.

### Proof of concept

See tests folder.

### Impact

Varies, depending on additional due diligence with integration with Zodiac

### Recommendation

Adding additional tests and performing edge-case testing after an integration with Zodiac.

### Developer Response

Developers agreed and will perform additional tests. Fixed in a later commit.

## 2. Low - View function may be called on nonexistent pool address

Consider putting a modifier that requires the target pool address to already have a market that is created.

### Proof of concept

```
function getUpdatedBorrowP2PExchangeRate(address _poolTokenAddress)
        external
        view
        returns (uint256)
```

May be called on `_poolTokenAddress` that does not exist as a pool yet.

### Impact

Extra gas expenditures on view functions.

### Recommendation

Put a modifier that checks for whether the target pool address exists on one or multiple of the functions referenced here.

### Developer Response

Developers agreed and later removed the targeted function

# Gas Savings Findings

## 1. Gas - Double Linked List implementation

### Proof of concept

With regards to gas and computational optimization, potentially using https://github.com/Dev43/heap-solidity instead of the current DoubleLinkedList implementation will drastically reduce average gas costs with larger amounts of matched suppliers and borrowers. This also helps prevent DOS attacks related to making the matching engine inefficient via. multiple small accounts. This can be implemented in the long run.

### Impact

Gas savings

### Recommendation

In the long term, switch to a different implementation of organizing users and pools instead of the DoubleLinkedList.

### Developer Response

Unsure whether or not fixing this is possible due to needing the ability to remove someone within a bloc. Implementation needs to be tested first to avoid potential issues worse than just inefficiency (e.g. contract halting, etc.)

## Informational Findings

## 1. Informational - Aave contracts may require resizing, may require redeployment because it exceeds max memory size.

### Impact

May cause extra gas to have to be consumed in redeploying the contract.

### Recommendation

Reducing contract size on certain Aave contracts

### Developer Response

Acknowledged & working.

## 2. Informational - Change "weth" to wEth: noncompiling gas report.

A small typo on the tests are stopping gas reports from running in TestMarketsManager.t.sol:29:37.

### Developer Response

Fixed in later version.

## 3. Informational - Typo in inline documentation

L85 of Logic.sol has a typo: : "MatchingEngineForCompound", not Aave.

### Developer Response

Fixed.

## 4. Informational - MarketsManager.sol L283 & L314 control flow decision made on block.timestamp.

block.timestamp != block.number. Block.number is guaranteed by the protocol to increase by one compared to the previous block, while block.timestamp can be minimally gamed by a miner (if this matters).

### Proof of concept

Several functions on MarketsManager.sol (e.g. getUpdatedSupplyP2PExchangeRates) rely on `block.timestamp == lastUpdateBlockNumber` to return updated exchange rates for a certain poolTokenAddress. Switching over to using block.number may yield more stable and secure exchange rate updating results.

### Impact

Informational.

### Recommendation

Choose to use block.number if miner gaming matters.

### Developer Response

Recognized and fixed in later version.

## Final remarks

With a focus on how contracts interact with each other and vulnerabilities with any delegate-call mechanics to the main Logic.sol contract, I found nothing particularly worthy of note that was a critical exploit. The protocol has immense coverage on tests with potential fluctuations and edge cases that can occur with Compound markets. Some additional tests on governance (by extension, the integration with Zodiac) can be added along with limited potential for heavy gas optimizations (viz. DoubleLinkedList setup).

## Appendix and FAQ