Morpho Security Analysis

by Pessimistic

This report is private

March 8, 2022

Abstract

In this report, we consider the security of smart contracts of Morpho project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Morpho smart contracts. We performed our audit according to the procedure described below.

The audit showed two issues of medium severity: Position fragmentation and a Bug. Also, two low-severity issues were found.

The overall code quality is good. However, there are several design flaws.

General recommendations

We recommend addressing the mentioned issues.

Project overview

Project description

For the audit, we were provided with Morpho project on a private GitHub repository, commit 42f7acccc6b4ca79baf63ef1ef88ff4a44e7a2e2.

The project includes sufficient protocol description. The code has thorough NatSpec comments. Additional documentation was provided to cover new Delta Withdraw and Modular iterations features.

All 88 tests pass, the code coverage is not measured.

The scope of the audit included the following files:

- **aave/MatchingEngineForAave.sol**
- **aave/PositionsManagerForAave.sol**
- **aave/MarketsManagerForAave**
- **aave/RewardsManager.sol**
- **common/SwapManager.sol**
- their dependencies

The total LOC of audited sources is 1703.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's codebase with the automated tools: Slither.
  - We manually verify (reject or confirm) all the issues found by the tools.
- Manual audit
  - We manually analyze the codebase for security vulnerabilities.
  - We assess the overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Position fragmentation

Each user operation (borrow, withdraw, etc.) calls (via delegate call) one of the matchingEngine contract methods to iterate over double linked list data structures to match positions in p2p.

The protocol provides ordering for such lists (via the insertSorted method) to prevent non-optimal matching - the positions are sorted in descending order, and thus large positions are taken into account first.

Since the maximum number of iterations for insertSorted is fixed at a constant of 20, the unsorted part might be accumulated after the 20th position. If this part is big enough, it can exclude from sorting all positions that come after it. Thus the matching algorithm will lose its efficiency.

Besides, managing gas directly is against current best practices.

Consider adding a function for manual sorting by the order computed on the backend for such a case.

M02. Bug

"Hard repay" logic may be incorrectly triggered during `_repay` execution at line 1028 of **PositionsManagerForAave** contract. We assume the code should compare `remainingToRepay` and `matchedBorrow` values.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Code quality

In the **MarketsManagerForAave** contract the value of `normalizedVariableDebt` is calculated in both branches of the if-expression at line 302. Consider moving this calculation outside the conditional operator.

L02. Misleading paramater name

The `_maxGasToConsume` paramater is used twice in some functions of **PositionsManagerForAave** contract, so it is possible that the specified amount of gas will be consumed twice, which may be misleading for the user of the protocol.

L03. Hardcoded address

SwapManager contract includes hardcoded addresses at lines 27-29. Consider utilzing immutable variables and initializing them via constructor.

Notes

N01. Unbalanced P2P positions

"Delta" feature gives users P2P rates when in fact their positions are pooled in **Aave**. The difference is payed by the protocol.

N02. Design drawback

"Delta" feature increases gas efficiency and relyability of the protocol. However, its implementation is coupled with the rest of the code, which additionally complicates the contracts.

We recommend selecting approaches that are free of this flaw. E.g., users could provide "hints" on which positions to match with. Alternatively, consider switching to a more efficient data structure.

This analysis was performed by Pessimistic:

Vladimir Tarasov, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Nikita Kirillov, Junior Security Engineer
Irina Vikhareva, Project Manager

March 8, 2022