Morpho Security Analysis

by Pessimistic

This report is public

October 06, 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 Different algorithms for the same values issue of medium severity and a few several low severity issues. The project's code quality meets high standards.

After the initial audit the codebase was updated. Most of the issues have been fixed. One medium severity issue was commented and one low severity issue was addressed.

General recommendations

We recommend fixing the remaining issue.

Project overview

Project description

For the audit, we were provided with Morpho project on a private GitHub repository, commit e3a9755a93c798c1e4a9039f93617d4ea85eab9.

The scope of the audit included only the following folders in the repository:

- **aave-v2/** folder;
- **aave-v3/** folder;
- **compound/** folder.

The documentation for the project included **Yellow_Paper.pdf** file, sha1sum is `a407d6f65ed3d2c5bff4fc35ed1aae640991124e`.

The project includes tests:

- 230 tests for **aave-v2/** folder;
- 169 tests for **aave-v3/** folder;
- 261 tests for **compound/** folder.

All tests pass successfully. The code coverage is not measured.

The total LOC of audited sources is 1300.

Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 81afc946e08deced6437f2345f6e6abea547748a. This update includes only fixes for most of the issues mentioned in the initial audit and an increase in the number of tests.

The project includes tests:

- 246 tests for **aave-v2/** folder;
- 183 tests for **aave-v3/** folder;
- 277 tests for **compound/** folder.

All tests passed successfully. The code coverage is not measured.

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 tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.
- 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.

Inter alia, we verify that:

- Morpho protocol properly implements documented functionality, particularly for supply, borrow, withdraw, liquidate, and repay actions.
- Contracts interact properly, and the value exchange process goes as expected when users enter or exit the protocol.
- Morpho protocol correctly integrates with Compound, AAVE v2, and AAVE v3 protocols.
- The project is resistant to re-entrancy and to flash loans and front-running attacks.
- Contracts' upgradeability is secure.

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. Different algorithms for the same values (commented)

This part of the code is being developed now. **InterestRatesManager** and **Lens** contracts calculate the same values using different algorithms in the **contracts/aave-v3/** folder. I.e., **InterestRatesManager** contract calculates `p2pSupplyGrowthFactor` and `p2pBorrowGrowthFactor` variables at lines 120–135. Their values depend on `poolSupplyGrowthFactor <= poolBorrowGrowthFactor` condition. However, **Lens** contract calculates `p2pSupplyGrowthFactor_` and `p2pBorrowGrowthFactor_` variables at lines 585–590 without any additional conditions. Consider using a unified method to calculate these variables.

> *Comment from the developers:* *We did not implemented M01 as the lens is still being written.*

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. Same code for different conditions (fixed)

In **contracts/aave-v3/RewardsManager.sol** file, the `if-else` block on lines 264–272 contains the duplicated code lines 271 and 272 that evaluate despite the condition. Consider moving this line of the code outside the `if-else` block to avoid code duplication.

The issue has been fixed and is not present in the latest version of the code.

L02. Unnecessary check (fixed)

In `borrowLogic` function of **EntryPositionsManager** contract (in **contracts/aave-v2/** and **contracts/aave-v3/** folders), the check at line 209 is unnecessary. Consider calling `_setBorrowing` function without this check since the check consumes extra ~20000 gas at the first call and does not optimize gas consumption for further calls.

The issue has been fixed and is not present in the latest version of the code.

L03. Unused return value (addressed)

In **PositionsManagerUtils** contract, the `_withdrawFromPool` function makes a `pool.withdraw` call at line 70 in **contracts/aave-v3/** folder and at line 69 in **contracts/aave-v2/** folder. However, the function does not process the returned value from this call, nor it returns its value. We recommend returning the value from `pool.withdraw` call in `_withdrawFromPool` function.

> *Comment from the developers: The value is not useful to our logic so we don't have need to handle it.*

This analysis was performed by Pessimistic:

Vladimir Pomogalov, Security Engineer
Vladimir Tarasov, Security Engineer
Evgeny Marchenko, Senior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

October 06, 2022