# Morpho Security Analysis

## by Pessimistic

This report is public

January 11, 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. One 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, security audit is not an 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 initial audit showed a critical Bug and two issues of medium severity: DoS and Dust positions. Also, many low-severity issues were found.

After the initial audit, the codebase was updated. The new version fixes both critical and medium severity issues. Most low severity issues have as well been fixed. The quality of the code has improved. No new features have been added, and no new security issues have been found.

# General recommendations

We recommend addressing the remaining issues, creating more tests for edge cases, and expanding protocol documentation.

# Project overview

## Project description

For the audit, we were provided with Morpho project on a private GitHub repository, commit 99cc6c7b4cbda9d4abc91dec57c2d516b61ef01d.

The scope of the audit only included **aave/** folder.

The project has a whitepaper, also additional private documentation was provided for the audit.

All 59 tests pass, the code coverage is 88.13%.

The total LOC of audited sources is 1571.

## Codebase update

After the initial audit, the codebase was updated. For the recheck, we were provided with commit c24232c2ea1353400bf8a784aa70200d766ac15a. In this version all the critical and medium issues were fixed.

# 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 project's codebase with automated tool Slither.
    - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
    - We manually analyze codebase for security vulnerabilities.
    - We assess 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.

### Bug (fixed)

Morpho protocol is designed to restrict users from holding undercollateralized positions. However, the implementation allows users to have such positions due to a bug.

When users attempt to withdraw their funds, `_withdraw()` function of **PositionManagerForAave** contract checks whether the user's position is undercollateralized via `_checkAccountLiquidity()` call at line 541. This function, in turn, calls `_getUserHypotheticalBalanceStates()` function at line 1001 that returns three values. The third returned value (`vars.collateralValue`) represents the available amount of tokens for the user to withdraw. However, this value is not considered in the check in `_checkAccountLiquidity()` function. Thus, users with good liquidity can withdraw their tokens from any market. As a result, the positions of these users can become undercollateralized.

Note that possible under-collateralization of positions should also be checked for other DeFi integrations.

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

## Medium severity issues

Medium issues can influence project operation in 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.

### DoS (fixed)

The `_repay` function calls `SafeERC20.safeApprove` with the requested amount of tokens and then calls `ILendingPool.repay` to perform a repayment. In the current implementation of the AAVE protocol, the actual amount of payback can become less than requested by the user. Thus, some tokens can remain approved. In this case, further calls to `SafeERC20.safeApprove` will revert, since "safeness" requires approves to be set only from- or to- zero. As a result, components of the protocol that repay to the AAVE lending pool (e.g., `supply` and `repay`) will revert.

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

### Dust positions (fixed)

In the **PositionManagerForAave** contract, the user's funds increase every second. Thus, when users attempt to withdraw funds, their balance will most likely increase by the moment of withdrawal, and the remnant will be positive. This will result in the accumulation of dust positions. Consider implementing a user method for complete position withdrawal.

*Support of complete position withdrawal added in the latest version of the code.*

## Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

### Underflow (fixed)

In **PositionManagerForAave** contract, when a user requests a withdrawal of large amount of tokens, `_withdraw()` function can revert with underflow error at lines 596 and 612. Consider capping withdraw amount with user's p2p reserve, i.e., the same way as it is handled for draining pool.

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

### Code composition (fixed)

Current implementation lacks modularity (e.g., it handles pool positions and p2p positions within a single function). Consider improving code composition by splitting complex functionality into separate functions.

*Code modularity has increased in the latest version of the code.*

### Insufficient coverage with tests

The project has tests. However, important scenarios are not covered. Testing is crucial for code security and audit does not replace tests in any way. We highly recommend covering the codebase with tests and making sure that all tests pass and coverage is sufficient.

### Gas consumption

- In **PositionManagerForAave** contract, functions that update a user's position can emit two similar events for the user's position update within a single call. E.g., in `_withdraw()` function, when the pool is not the only source of liquidity. We recommend calculating the resulting position of the user and emitting only a single event afterwards to optimize gas consumption and improve code readability.
- In **PositionManagerForAave** contract, the check at line 595 in `_withdraw()` function consumes a lot of gas and can revert in some cases, e.g. when `NMAX` positions of dust is exceeded.
- In **MarketsManagerForAave** contract, `addressesProvider` variable is set during deployment at line 36 and is never changed later. Consider declaring it as `immutable` to reduce gas consumption.

## Code quality

- **PositionManagerForAave** contract uses only variable debts mechanics of AAVE protocol. Consider declaring this mechanics parameter as a `constant` instead of magic number at lines 664, 683, and 921.

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

- In **PositionManagerForAave** contract, functions `_withdraw()` and `_repay()` contain duplicating code for external calls. Consider precalculating the parameters of an external call and performing the call in a single point of the function afterwards to minimize code duplication and improve its readability.

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

- In **MarketsManagerForAave** contract, the name of `setLendingPool()` function is misleading since it is not a setter, but it updates the lending pool address to the value from AAVE.

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

- The `claimRewards()` function of **PositionManagerForAave** contract contains a hardcoded address at line 279.

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

- Performing division operations before multiplication can result in the loss of precision. Consider performing division in the last step when calculating a siezed collateral in `liquidate()` function at line 503 of **PositionManagerForAave** contract.

- The system imports `IERC20` interface from two different sources: AAVE and OZ. Consider using only a single source to eliminate any ambiguity about which declaration is being used over the project.

This analysis was performed by Pessimistic:

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

January 11, 2022