SOLIDIFIED

Audit Report for Morpho Protocol - November 4, 2021

Summary

Audit Report prepared by Solidified covering the Morpho Protocol smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on November 4, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/morpho-labs/morpho-contracts

Commit number: `7fd761c88c2aead93c967a5c16f2316201cdaedf`

UPDATE: Fixes received on November 17, 2021
Final commit number: `f6ab54e055dd729ca1a5795a686c4232a56fe6a8`

Intended Behavior

Morpho is a protocol that improves the capital efficiency of AAVE or Compound whilst preserving the same liquidity and risk guarantees.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the Morpho Protocol contracts contain no critical issues, 2 major issues, 0 minor issues, 5 informational notes and 1 warning.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	PositionsManagerForCompound.sol: Tree traversal operations could potentially bring the protocol to a halt	Major	Resolved
2	PositionsManagerForCompound.sol: Loop operations could be optimized to consume less gas	Note	-
3	MarketsManagerForCompound.sol: Function updateThreshold() does not validate _marketAddress	Note	Resolved
4	PositionsManagerForCompound.sol: Function _repay() does not validate _amount	Note	Resolved
5	Miscellaneous Notes	Note	Resolved
6	Protocol security relies heavily on external protocols	Warning	-

Critical Issues

No critical issues have been found.

Major Issues

1. PositionsManagerForCompound.sol: Tree traversal operations could potentially bring the protocol to a halt

A lot of `PositionsManagerForCompound` tree operations require traversal of their respective trees using `while()` loops. When these trees grow large enough for these operations to exceed the current Ethereum block gas limit, the protocol will come to a halt due to these functions always reverting.

Recommendation

Consider limiting the size of `PositionsManagerForCompound` trees to a value that never exceeds the block gas limit. One way to achieve this is by running simulations until these limits are reached.

Note

The block gas limit could also potentially be exceeded if an account has too many markets in `enteredMarkets`. Although this is much less likely to happen, it's still important to be aware of the possibility.

Status

Resolved

Minor Issues

No minor issues have been found.

Informational Notes

2. PositionsManagerForCompound.sol: Loop operations could be optimized to consume less gas

Loop operations in function `_getUserHypotheticalBalanceStates()` keep reading their respective array length on each loop using `enteredMarkets[_account].length`. This can add up in gas cost when traversing relatively large arrays since it's reading from storage on each iteration.

Recommendation

Consider storing the array's length in a memory variable and using this variable instead inside the loop.

3. MarketsManagerForCompound.sol: Function updateThreshold() does not validate _marketAddress

Function `updateThreshold()` doesn't include any validation which ensures that the market has actually been created for the address being passed as an argument.

Recommendation

Consider using the `isMarketCreated()` modifier to ensure that the market exists.

Status

Resolved

4. PositionsManagerForCompound.sol: Function _repay() does not validate _amount

---

Recommendation

Consider checking that `_amount > 0`.

Status

Resolved

5. Miscellaneous Notes

- Consider fixing some spelling mistakes: notde ⇒ node, Exhange ⇒ Exchange.
- Consider providing a license for the contracts.

Status

Resolved

Warnings

6. Protocol security relies heavily on external protocols

The Morpho protocol relies heavily on external protocols and its security will always be limited by its least secure dependency.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Morpho Labs or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.