Morpho Protocol V1

Retest Report

June 3, 2022

Prepared for:
Paul Frambot and Merlin Egalite
Morpho Labs

Prepared by: **Michael Colburn**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

When undertaking a retesting project, Trail of Bits reviews the fixes implemented for issues identified in the original report. Retesting involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

Table of Contents

Executive Summary

Engagement Overview

Morpho Labs engaged Trail of Bits to review the security of its Morpho-Compound smart contracts. From May 9 to May 20, 2022, a team of three consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's scope, timeline, test targets, and coverage are provided in the original audit report.

Morpho Labs contracted Trail of Bits to review the fixes implemented for issues identified in the original report. On May 27, 2022, one consultant conducted a review of the client-provided source code, with one half person-day of effort.

Summary of Findings

The original audit uncovered significant flaws that could impact system integrity or availability. A summary of the original findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	2
Low	1
Informational	3
Undetermined	2

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	2
Auditing and Logging	1
Data Validation	4
Timing	1

Overview of Retest Results

Morpho Labs has sufficiently addressed most of the issues described in the original audit report. The issues identified by the Morpho Labs team, listed in appendix D of the original report, have all been adequately addressed. No new issues were identified during this retest.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineers were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Felipe Manzano, Consultant
felipe.manzano@trailofbits.com

Bo Henderson, Consultant
bo.henderson@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
May 5, 2022	Pre-project kickoff call
May 15, 2022	Status update meeting #1
May 20, 2022	Delivery of report draft
May 24, 2022	Report readout meeting
June 4, 2022	Final report delivery
June 4, 2022	Delivery of retest report

Project Methodology

Our work in the retesting project included the following:

- A review of the findings in the original audit report, including the additional issues noted by the Morpho Labs team
- A manual review of the client-provided source code and configuration material

Project Targets

The engagement involved retesting of the following target.

`morpho-contracts`

Repository	https://github.com/morpho-labs/morpho-contracts
Version	`26274a9c7e69b1bf785e1c3cfe36756df3d38fb6`
Type	Solidity
Platform	EVM

Summary of Retest Results

The table below summarizes each of the original findings and indicates whether the issue has been sufficiently resolved.

ID	Title	Status
1	Lack of two-step process for contract ownership changes	Unresolved
2	Incomplete information provided in Withdrawn and Repaid events	Resolved
3	Missing access control check in withdrawLogic	Resolved
4	Lack of zero address checks in setter functions	Unresolved
5	Risky use of toggle functions	Resolved
6	Anyone can destroy Morpho's implementation	Resolved
7	Lack of return value checks during token transfers	Resolved
8	Risk of loss of precision in division operations	Resolved

Detailed Retest Results

1. Lack of two-step process for contract ownership changes	
Status: **Unresolved**	
Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MORPHO-1
Target: Throughout the codebase	

Description

The owner of the `IncentivesVault` contract and other `Ownable` Morpho contracts can be changed by calling the `transferOwnership` function. This function internally calls the `_transferOwnership` function, which immediately sets the contract's new owner. Making such a critical change in a single step is error-prone and can lead to irrevocable mistakes.

```
contract IncentivesVault is IIncentivesVault, Ownable {
```

Figure 1.1: Inheritance of `contracts/compound/IncentivesVault.sol`

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}
```

Figure 1.2: The `transferOwnership` function in `@openzeppelin/contracts/access/Ownable.sol`

Fix Analysis

The Morpho Labs team chose not to address this issue. Morpho Labs does not anticipate the need to update contract owners and expects any issues to be detected during the governance process.

2. Incomplete information provided in Withdrawn and Repaid events	
Status: **Resolved**	
Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-MORPHO-2
Target: `contracts/compound/PositionsManager.sol`	

Description

The core operations in the `PositionsManager` contract emit events with parameters that provide information about the operations' actions. However, two events, `Withdrawn` and `Repaid`, do not provide complete information.

For example, the `withdrawLogic` function, which performs withdrawals, takes a `_supplier` address (the user supplying the tokens) and `_receiver` address (the user receiving the tokens):

```
    /// @param _supplier The address of the supplier.
    /// @param _receiver The address of the user who will receive the tokens.
    /// @param _maxGasForMatching The maximum amount of gas to consume within a
matching engine loop.
    function withdrawLogic(
        address _poolTokenAddress,
        uint256 _amount,
        address _supplier,
        address _receiver,
        uint256 _maxGasForMatching
    ) external
```

Figure 2.1: The function signature of `PositionsManager`'s `withdrawLogic` function

However, the corresponding event in `_safeWithdrawLogic` records only the `msg.sender` of the transaction, so the `_supplier` and `_receiver` involved in the transaction are unclear. Moreover, if a withdrawal is performed as part of a liquidation operation, three separate addresses may be involved—the `_supplier`, the `_receiver`, and the `_user` who triggered the liquidation—and those monitoring events will have to cross-reference multiple events to understand whose tokens moved where.

```
    /// @notice Emitted when a withdrawal happens.
    /// @param _user The address of the withdrawer.
    /// @param _poolTokenAddress The address of the market from where assets are
withdrawn.
```

```
    /// @param _amount The amount of assets withdrawn (in underlying).
    /// @param _balanceOnPool The supply balance on pool after update.
    /// @param _balanceInP2P The supply balance in peer-to-peer after update.
    event Withdrawn(
        address indexed _user,
        …
```

Figure 2.2: The declaration of the `Withdrawn` event in `PositionsManager`

```
        emit Withdrawn(
            msg.sender,
            _poolTokenAddress,
            _amount,
            supplyBalanceInOf[_poolTokenAddress][msg.sender].onPool,
            supplyBalanceInOf[_poolTokenAddress][msg.sender].inP2P
        );
```

Figure 2.3: The emission of the `Withdrawn` event in the `_safeWithdrawLogic` function

A similar issue is present in the `_safeRepayLogic` function's `Repaid` event.

Fix Analysis

Variable names and event parameters have been updated to more accurately reflect the state changes made by the affected operations.

3. Missing access control check in withdrawLogic

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-MORPHO-3
Target: `contracts/compound/PositionsManager.sol`	

Description

The `PositionsManager` contract's `withdrawLogic` function does not perform any access control checks. In practice, this issue is not exploitable, as all interactions with this contract will be through `delegatecalls` with a hard-coded `msg.sender` sent from the main `Morpho` contract. However, if this code is ever reused or if the architecture of the system is ever modified, this guarantee may no longer hold, and users without the proper access may be able to withdraw funds.

```
    /// @dev Implements withdraw logic with security checks.
    /// @param _poolTokenAddress The address of the market the user wants to
interact with.
    /// @param _amount The amount of token (in underlying).
    /// @param _supplier The address of the supplier.
    /// @param _receiver The address of the user who will receive the tokens.
    /// @param _maxGasForMatching The maximum amount of gas to consume within a
matching engine loop.
    function withdrawLogic(
        address _poolTokenAddress,
        uint256 _amount,
        address _supplier,
        address _receiver,
        uint256 _maxGasForMatching
    ) external {
```

Figure 3.1: The `withdrawLogic` function, which takes a supplier and whose comments note that it performs security checks

Fix Analysis

The Morpho Labs team did not address this issue.

4. Lack of zero address checks in setter functions

Status: **Unresolved**	
Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-MORPHO-4
Target: `contracts/compound/MorphoGovernance.sol`	

Description

Certain setter functions fail to validate incoming arguments, so callers can accidentally set important state variables to the zero address. A mistake like this could initially go unnoticed because a `delegatecall` to an address without code will return success.

```
/// @notice Sets the `positionsManager`.
/// @param _positionsManager The new `positionsManager`.
function setPositionsManager(IPositionsManager _positionsManager) external onlyOwner
{
    positionsManager = _positionsManager;
    emit PositionsManagerSet(address(_positionsManager));
}
```

Figure 4.1: An important address setter in `MorphoGovernance`

Fix Analysis

The Morpho Labs team did not address this issue.

5. Risky use of toggle functions	
Status: **Resolved**	
Severity: **Low**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-MORPHO-5
Target: `contracts/compound/MorphoGovernance.sol`	

Description

The codebase uses a toggle function, `togglePauseStatus`, to pause and unpause a market. This function is error-prone because setting a pause status on a market depends on the market's current state. Multiple uncoordinated pauses could result in a failure to pause a market in the event of an incident.

```
/// @notice Toggles the pause status on a specific market in case of emergency.
/// @param _poolTokenAddress The address of the market to pause/unpause.
function togglePauseStatus(address _poolTokenAddress)
    external
    onlyOwner
    isMarketCreated(_poolTokenAddress)
{
    Types.MarketStatus storage marketStatus_ = marketStatus[_poolTokenAddress];
    bool newPauseStatus = !marketStatus_.isPaused;
    marketStatus_.isPaused = newPauseStatus;
    emit PauseStatusChanged(_poolTokenAddress, newPauseStatus);
}
```

Figure 5.1: The `togglePauseStatus` method in `MorphoGovernance`

This issue also applies to `togglePartialPauseStatus`, `toggleP2P`, and `toggleCompRewardsActivation` in `MorphoGovernance` and to `togglePauseStatus` in `IncentivesVault`.

Fix Analysis

The relevant setter functions have been updated to set a specific state rather than to toggle between states.

6. Anyone can destroy Morpho's implementation	
Status: **Resolved**	
Severity: **High**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-MORPHO-6
Target: `contracts/compound/Morpho*.sol`	

Description

An incorrect access control on the `initialize` function for Morpho's implementation contract allows anyone to destroy the contract.

Morpho uses the `delegatecall` proxy pattern for upgradeability:

```
abstract contract MorphoStorage is OwnableUpgradeable, ReentrancyGuardUpgradeable {
```

Figure 6.1: `contracts/compound/MorphoStorage.sol#L16`

With this pattern, a proxy contract is deployed and executes a `delegatecall` to the implementation contract for certain operations. Users are expected to interact with the system through this proxy. However, anyone can also directly call Morpho's implementation contract.

Despite the use of the proxy pattern, the implementation contract itself also has `delegatecall` capacities. For example, when called in the `updateP2PIndexes` function, `setReserveFactor` executes a `delegatecall` on user-provided addresses:

```
function setReserveFactor(address _poolTokenAddress, uint16 _newReserveFactor)
    external
    onlyOwner
    isMarketCreated(_poolTokenAddress)
{
    if (_newReserveFactor > MAX_BASIS_POINTS) revert ExceedsMaxBasisPoints();
    updateP2PIndexes(_poolTokenAddress);
```

Figure 6.2: `contracts/compound/MorphoGovernance.sol#L203-L209`

```
function updateP2PIndexes(address _poolTokenAddress) public {
    address(interestRatesManager).functionDelegateCall(
        abi.encodeWithSelector(
            interestRatesManager.updateP2PIndexes.selector,
            _poolTokenAddress
        )
    );
}
```

Figure 6.3: `contracts/compound/MorphoUtils.sol#L119-L126`

These functions are protected by the `onlyOwner` modifier; however, the system's owner is set by the `initialize` function, which is callable by anyone:

```
function initialize(
    IPositionsManager _positionsManager,
    IInterestRatesManager _interestRatesManager,
    IComptroller _comptroller,
    Types.MaxGasForMatching memory _defaultMaxGasForMatching,
    uint256 _dustThreshold,
    uint256 _maxSortedUsers,
    address _cEth,
    address _wEth
) external initializer {
    __ReentrancyGuard_init();
    __Ownable_init();
```

Figure 6.4: `contracts/compound/MorphoGovernance.sol#L114-L125`

As a result, anyone can call `Morpho.initialize` to become the owner of the implementation and execute any `delegatecall` from the implementation, including to a contract containing a `selfdestruct`.

Doing so will cause the proxy to point to a contract that has been destroyed.

This issue is also present in `PositionsManagerForAave`.

Fix Analysis

The `MorphoStorage` contract, which is inherited by all of the contracts that receive `delegatecalls`, now has a constructor with the `initializer` modifier to prevent the implementation from being initialized directly.

7. Lack of return value checks during token transfers

Status: **Resolved**	
Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-MORPHO-7
Target: `contracts/compound/IncentivesVault.sol`	

Description

In certain parts of the codebase, contracts that execute transfers of the Morpho token do not check the values returned from those transfers.

The development of the Morpho token was not yet complete at the time of the audit, so we were unable to review the code specific to the Morpho token. Some tokens that are not ERC20 compliant return `false` instead of reverting, so failure to check such return values could result in undefined behavior, including the loss of funds. If the Morpho token adheres to ERC20 standards, then this issue may not pose a risk; however, due to the lack of return value checks, the possibility of undefined behavior cannot be eliminated.

```
function transferMorphoTokensToDao(uint256 _amount) external onlyOwner {
    morphoToken.transfer(morphoDao, _amount);
    emit MorphoTokensTransferred(_amount);
}
```

Figure 7.1: The `transerMorphoTokensToDao` method in `IncentivesVault`

Fix Analysis

While the Morpho token is still under development, `transferMorphoTokensToDao` now uses `safeTransfer` to transfer tokens.

8. Risk of loss of precision in division operations

Status: **Resolved**	
Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-MORPHO-8
Target: `contracts/compound/PositionsManager.sol`	

Description

A common pattern in the codebase is to divide a user's debt by the total supply of a token; a loss of precision in these division operations could occur, which means that the supply delta would not account for the entire matched delta amount. The impact of this potential loss of precision requires further investigation.

For example, the `borrowLogic` method uses this pattern:

```
    toWithdraw += matchedDelta;
    remainingToBorrow -= matchedDelta;
    delta.p2pSupplyDelta -= matchedDelta.div(poolSupplyIndex);
    emit P2PSupplyDeltaUpdated(_poolTokenAddress, delta.p2pSupplyDelta);
```

Figure 8.1: Part of the `borrowLogic()` method

Here, if `matchedDelta` is not a multiple of `poolSupplyIndex`, the remainder would not be taken into account. In an extreme case, if `matchedDelta` is smaller than `poolSupplyIndex`, the result of the division operation would be zero.

An attacker could exploit this loss of precision to extract small amounts of underlying tokens sitting in the `Morpho` contract.

Fix Analysis

The Morpho Labs team restructured the arithmetic to further mitigate potential rounding errors.

A. Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Retest Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

B. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories

Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.