MetaMorpho
Security Review

Cantina Managed review by:
Tnch, Security Researcher
Om parikh, Security Researcher
0xDeadbeef, Associate Security Researcher

September 24, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	**Description**
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Morpho Porotocol is a decentralized, noncustodial lending protocol implemented for the Ethereum Virtual Machine. The protocol had two main steps in its evolution with two independent versions: Morpho Optimizers and Morpho Blue.

From Aug 27th to Sep 4th the Cantina team conducted a review of metamorpho-private on commit hash bf008d62. The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 2
- Low Risk: 0
- Gas Optimizations: 0
- Informational: 5

3 Findings

3.1 Medium Risk

3.1.1 Unfair pricing of metamorpho share when it is used as collateral in morpho market

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: `MetaMorpho` uses MorphoChainlinkOracleV2.sol for fair pricing of the vault's share.

However, in the case of vault which supports `lostAssets` (i.e no share price decrease feature), if `lostAssets > 0` and bad debt is not repaid externally, the oracle would give the inflated price. As per discussion with morpho team in this comment:

> For a vault with 50% of bad debt, it is highly more likely that it will experience bad debt, sooner or later. When it will happen, users will not be able to get anything out of their shares. Is it sound to price shares at $0.5, not sure. But in this case, lenders lending against these shares already lost, so it's not even very useful to care about the price.

It is still important that shares are priced closest to the actual redeemable price since that would allow borrowers to be liquidated and lenders having fair chance to participate in bank-run for remaining 50%.

> Morpho Labs beleives that it's not a good idea to lend against a vault that has significant lost assets, so pricing the vault in such situation should not be cared of.

Though this true that lending will pause against such vault when offchain consensus is reached that debt hole is large enough, but the majority of impact is from the capital already at stake and not then newly anticipated incoming assets.

Recommendation: Current oracle should be modified for vaults supporting `lostAssets` to:

- Assess the risk offchain for bad-debt and adjust the share price by some tolerance threshold. Since the price is only used for computing borrowable and liquidation, once bad has start accruing, effective price can be aggressively reduced and after a certain point it can be a step function switching it to 0.
- Have circuit breakers in place to revert in the `price()` function temporarily until fair price is assessed offchain and put on-chain. This will also revert liquidations and borrows temporarily.

Morpho Labs: Acknowledged. We are aware of this issue, as mentioned, and we don't think that it requires any change in the code.

Cantina Managed: Acknowledged.

3.1.2 Liquidators can make suppliers inadvertently deposit assets to unhealthy `MetaMorpho` vaults

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The reviewed version of the `MetaMorpho` contract introduces the `lostAssets` state variable, which according to its documentation "*stores the missing assets due to realized bad debt or forced market removal*".

This is due to the fact that in this kind of vault, by design `totalAssets` won't be decreased to account for missing assets. So in principle, when there's bad debt realized in a Morpho market, the `MetaMorpho` vault must increase the `lostAssets` amount accordingly, to account for any missing assets. For this vault, anyone that is assessing its health and risk exposure is expected to not only pay attention to the total assets managed by the vault (as returned by `totalAssets`), but also to the actual hole in the vault, given by `lostAssets`.

However, it must be noted that `lostAssets` only starts accounting for missing assets when bad debt is realized in Morpho. Which happens in circumstances when a liquidation leaves a position with strictly zero collateral and positive debt. But it's possible for liquidators to leave dust collateral after the liquidation, and avoid realizing bad debt (see Morpho.sol#L392-L403). This behavior in Morpho was noted and discussed in a previous Cantina review (see issue 3.5.5 - There might not be enough incentives for liquidators to realize bad debt).

In such circumstances, the `lostAssets` value doesn't increase in the `MetaMorpho` vault. It stays at zero, even if practically there's no collateral left after the liquidation (only dust). This can be seen in the following test case:

```
function testLostAssetsAfterAlmostTotalLiquidation(uint256 borrowed, uint256 collateral, uint256 deposit)
↪   public {
    borrowed = bound(borrowed, MIN_TEST_ASSETS, MAX_TEST_ASSETS);
    collateral = bound(collateral, borrowed.mulDivUp(1e18, allMarkets[0].lltv), type(uint128).max);
    deposit = bound(deposit, borrowed, MAX_TEST_ASSETS);

    collateralToken.setBalance(BORROWER, collateral);
    loanToken.setBalance(SUPPLIER, deposit);

    // Supplier deposits loanToken using MetaMorpho vault
    vm.prank(SUPPLIER);
    vault.deposit(deposit, SUPPLIER);

    // Borrower deposits some collateral and borrows loanTokens
    vm.startPrank(BORROWER);
    morpho.supplyCollateral(allMarkets[0], collateral, BORROWER, hex"");
    morpho.borrow(allMarkets[0], borrowed, 0, BORROWER, BORROWER);
    vm.stopPrank();

    uint256 initialTotalAssets = vault.totalAssets();

    // Price drops
    oracle.setPrice(0);

    // Liquidator takes almost all collateral, leaving 1 unit
    vm.prank(LIQUIDATOR);
    morpho.liquidate(allMarkets[0], BORROWER, collateral - 1, 0, hex"");

    // One unit of collateral remaining => no bad debt accounted for yet in Morpho
    assertEq(collateralToken.balanceOf(address(morpho)), 1);

    vault.deposit(0, SUPPLIER); // trigger update lostAssets on vault just in case

    // As expected total assets hasn't changed
    assertEq(vault.totalAssets(), initialTotalAssets);

    // But the MetaMorpho vault hasn't registered any lost assets either
    assertEq(vault.lostAssets(), 0);
}
```

While this could be considered the expected behavior of `lostAssets`, it must be highlighted that it introduces a potentially risky scenario for suppliers of `MetaMorpho` vaults. Because if `totalAssets` cannot be fully relied on to know the actual assets managed by the vault, and `lostAssets` cannot be fully relied on to know the actual assets lost, then it becomes harder to asses risk of this kind of `MetaMorpho` vaults.

Liquidators now have even more incentives to not realize bad debt in Morpho markets, and just leave dust units of collateral. Because they can use this to attack vault's suppliers.

Let's imagine a scenario where there's a Morpho market that is on the brink of realizing bad debt if a position is liquidated, and a `MetaMorpho` vault that integrates this market. Suppliers know that they should inspect the `lostAssets` state variable to better assess risks, so the more cautious could supply with sensible protections, using an on-chain contract that checks `lostAssets` before actually joining the vault:

```
contract Supplier {
    function supply(/*...*/) {
        if(vault.lostAssets() == 0) {
            // Supply logic
        } else {
            revert(/*...*/);
        }
    }
}
```

But this protection can be easily bypassed by malicious liquidators. Because as explained, it's possible to have the vault lose assets without actually increasing `lostAssets`. A liquidator can wait for one or many suppliers to submit transactions that supply assets to the vault (which, before being liquidated, might look healthy enough), and create a bundle of transactions where the liquidator:

1. Acquires shares of the `MetaMorpho` vault.

2. Liquidates the position in the `Morpho` contract, without realizing bad debt (that is, making the vault lose assets but leaving some dust collateral).
3. Inserts users' transactions that supply assets to the vault (where any call to `vault.lostAssets()` will return 0).
4. Redeems the shares acquired in (1) to withdraw the assets supplied by users in (3).

This can leave suppliers with shares in a vault that won't allow them to withdraw, because the liquidator has taken their assets in (4). Even if the suppliers implemented protections to not supply to unhealthy markets.

Recommendation: If the described behavior of `lostAssets` is expected, the Morpho team should consider improving the related documentation and test cases to better specify it. Moreover, they should provide clear guidelines for vault suppliers on how to evaluate risks of this new type of vaults, including any guidelines and possible protections to help mitigate the described risks of malicious liquidators bundling supply transactions to make suppliers lose assets.

Morpho Labs: Interesting issue.

First it's important to note that this situation is a pure griefing attack, meaning you don't get anything out of it, except that you harmed the new depositor.

Second, the issue is a bit more tricky than what's written. Indeed, when there is a bad debt that's about to be realised on a market, that the vault has V assets, supplying x just before the bad debt realisation won't ever make you lose more than V. Which means that the scenario described is impossible, the depositor could always instantly withdraw. Although other depositors might start a bank run, and the new depositor could get stuck as described.

Also, there is a similar issue in the "normal" MetaMorpho, where you could enter in a vault thinking everything was fine but there was pending bad debt and it gets realised just after you enter in the vault. This has not been documented, because we considered it too edge case (depositors being liquidators etc).

In the end we think that there is no need to put a comment for this edge-case.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Spec mismatch for ERC20Permit `string name`

Severity: Informational

Context: MetaMorpho.sol#L136

Description: `name` in used in ERC4626, ERC20 and ERC20Permit, out of which it function override solves for ERC4626 and ERC20, but in `EIP712` contract, `name` is immutable.

From EIP712 spec:

> `string name` the user readable name of signing domain, i.e. the name of the DApp or the protocol.

It should be human-readable since that's the only human-readable field to assist the user.

Recommendation: Consider changing the `name` to a human-readable string in the `EIP712` contract.

Morpho Labs: Fixed in PR 42.

Cantina Managed: Fixed.

3.2.2 Confusing use of `uint184` and `uint196` types for pending values

Severity: Informational

Context: PendingLib.sol#L34

Description: The `MetaMorpho` contract uses the `PendingLib` library to update pending values of guardians, supply caps and timelocks. The supply caps and pending timelock values, respectively stored in the `pendingCap` and `pendingTimelock` state variables, are updated using the `PendingLib::update(PendingUint192 storage pending, uint184 newValue, uint256 timelock)` function. These two state variables use the `PendingUint192` struct, which internally stores the pending value to set as a `uint192` number.

However, the second parameter of the mentioned `update` function is a `uint184`, which is implicitly cast to a `uint192` when written to the `value` field of the `PendingUint192` struct.

While supply caps of markets are expected to be of `uint184` type, their pending values are stored as `uint192` values, but capped to `uint184` when calling the `update` function. Pending timelock values are expected to be `uint192`, but they're always cast to `uint184` when calling the `update` function, only to then be stored as `uint192` as well.

While this does not currently pose a security risk, the inconsistent use of these types seems error-prone and could lead to unexpected casting errors in the future.

Recommendation: Consider either updating the code to consistently cast and store pending values in the expected types, or better documenting the reasoning behind the current implementation to avoid future mistakes.

Morpho Labs: Acknowledged. Indeed, the pending values should be `uint184`, although we decided not to change this to keep the code diff with the normal MetaMorpho small.

Cantina Managed: Acknowledged.

3.2.3 Incomplete documentation of `totalAssets` behavior

Severity: Informational

Context: MetaMorpho.sol#L611

Description: The reviewed version of the `MetaMorpho` contract introduces a notable change in the way the ERC4626's function `totalAssets` behaves. While the EIP specifies that `totalAssets` represents the "*total amount of the underlying asset that is managed by the vault*", that might not always be the case in the new `MetaMorpho` vault.

The `totalAssets` function has been re-designed to maintain a non-decreasing value, even in scenarios where the vault experiences asset losses due to bad debt in Morpho markets or forced market removals. Consequently, the value returned by `totalAssets` may not always accurately reflect the current amount of assets under management by the vault. Instead, it may overestimate the amount.

Recommendation: Consider improving the documentation of `totalAssets` in the function's docstrings to clearly specify this behavior and avoid errors in contracts and off-chain clients querying it.

Morpho Labs: Fixed in PR 44.

Cantina Managed: Fixed.

3.2.4 Inconsistent use of `msg.sender`

Severity: Informational

Context: MetaMorpho.sol#L781

Description: The `MetaMorpho` contract uses the inherited `_msgSender()` function whenever it needs to read the `msg.sender` value. The only place this doesn't happen is in the `_setCap` function, when emitting the `SetWithdrawQueue` event.

Recommendation: For consistency across the contract and to avoid confusions, consider changing this occurrence to `_msgSender()`.

Morpho Labs: Acknowledged. We decided to not fix this issue, to minimise the code diff with the normal MetaMorpho.

Cantina Managed: Acknowledged.

3.2.5 Missing new owner powers in documentation

Severity: Informational

Context: README.md#L41

Description: The project's README lists all powers of vault owners. However, it's missing the new powers to set the vault's name and symbol granted by the `setName` and `setSymbol` functions.

Recommendation: Consider including them to better specify the expected powers of vault owners.

Morpho Labs: Fixed in PR 43.

Cantrina Managed: Fixed.