SPEARBIT

Morpho Security Review

Auditors

Christoph Michel, Lead Security Researcher

Emanuele Ricci, Security Researcher

JayJonah8, Security Researcher

Datapunk, Junior Security Researcher

Emile Baizel, Junior Security Researcher

Report prepared by: Pablo Misirov

June 19, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Morpho is a lending pool optimizer. It improves the capital efficiency of positions on existing lending pools by seamlessly matching users peer-to-peer.

Morpho's rates stay between the supply rate and the borrow rate of the pool, reducing the interests paid by the borrowers while increasing the interests earned by the suppliers. It means that you are getting boosted peer-to-peer rates or, in the worst case scenario, the APY of the pool. Morpho also preserves the same experience, liquidity and parameters (collateral factors, oracles, ...) as the underlying pool.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of morpho-aave-v3 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 5 days in total, Morpho engaged with Spearbit to review the morpho-aave-v3 protocol. In this period of time a total of **24** issues were found.

Summary

Project Name	Morpho
Repository	morpho-aave-v3
Commit	4f1e0c...7eae
Type of Project	Lending and Borrowing, DeFi
Audit Timeline	June 1 to June 7
Two week fix period	Jan 7 - Jan 21

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	3	2	1
Low Risk	5	1	4
Gas Optimizations	2	1	1
Informational	13	8	5
Total	**24**	**13**	**11**

5 Findings

5.1 Critical Risk

5.1.1 Morpho is not using the correct user's balance when calling `MorphoInternal._updateRewards`

Severity: Critical Risk

Context: PositionsManagerInternal.sol#L419, PositionsManagerInternal.sol#L437, MorphoInternal.sol#L383

Description: Every time that the user performs an operation on Morpho that updates his pool balance, Morpho must update the user's position on the `RewardsManager`.

When the user performs a `supply/supplyCollateral/withdraw/withdrawCollateral` operation, Morpho is not calling the `MorphoInternal._updateRewards` function with the correct user balance. As a result, the user will accrue fewer rewards compared to what he/she should.

That missed reward is permanently lost because once `RewardsManager._updateUserData` is executed, the `localRewardData.usersData[user].index` is updated to the asset index and can't be accrued until the asset's index has accrued more rewards.

Recommendation: Morpho should

- Add the user's on pool supply balance to the supply collateral balance when the `_updateRewards` is called from the `PositionsManagerInternal._accountWithdrawCollateral` and `PositionsManagerInternal._accountSupplyCollateral`
- Add the user's on pool supply collateral balance to the supply balance when the `_updateRewards` is called from the `MorphoInternal._updateInDS` and the action involves a supply/withdraw operation

Morpho: Recommendation implemented in PR 860.

Spearbit: Verified.

5.2 Medium Risk

5.2.1 `SupplyVault` can be initialized with `initialDeposit` equal to zero, allowing attackers to leverage the ERC4626 inflation attacks

Severity: Medium Risk

Context: SupplyVault.sol#L76

Description: While it's correct that the `ERC4626UpgradeableSafe` lib contract does allow users to configure the vault without minting some initial shares, the `SupplyVault` contract should revert if `initialDeposit` is equal to zero.

With `initialDeposit` equal to zero, the ERC4626 is keen to known inflation attacks (see Mixbytes "Overview of the Inflation Attack" for an explanation).

Recommendation: Morpho should consider adding a lower bound value to the `initialDeposit`. The input parameter `initialDeposit` should be anyway validated with proper tests and simulations for each `underlying` before the vault deployment in order to prevent this kind of attack once the vault is deployed.

Spearbit: ThePR 854 will make the `initialize` execution revert if `initialDeposit` is equal to zero.

Morpho should anyway choose a proper value for `initialDeposit` to be sure to initialize the `SupplyVault` in a way that prevents the execution of any ERC4626 inflation attacks.

5.2.2 `SupplyVault` can use outdated `totalAssets`

Severity: Medium Risk

Context: MorphoGetters.sol#L86, MorphoInternal.sol#L470

Description: The `SuppyVault.totalAssets()` function calls `_MORPHO.supplyBalance(_underlying, address(this))` which computes the underlying balance by multiplying the scaled balances with the corresponding supply indexes. These indexes are only updated once per block. If a flashloan happens on Aave that pays a premium, Aave's liquidity index increases which should leader to a larger underyling total amount for the vault. However, the old pool supply index for the block is used, this allows minting more shares at the old cheaper share price.

Example:

- MEV searcher sees a transaction that performs a flashloan
- Frontruns the transaction by:
  - triggering an index update on Morpho
  - minting shares at the vault at the lower, un-updated share price
- In the next block, they redeem their shares for an increase

Recommendation: This is part of a larger issue of only updating indexes once per block. Not caching the indexes per block in the main protocol in `_computeIndexes` also fixes this issue for the vault.

Morpho: This is fixed in the latest upgrade of the Morpho-AaveV3 optimizer, whose content is available in PR 849, which is in the process of being merged.

Spearbit: Acknowledged.

5.2.3 `SupplyVault`'s `maxDeposit`, `maxWithdraw`, `maxMint`, `maxRedeem` functions not EIP4626-compliant

Severity: Medium Risk

Context: ERC4626Upgradeable.sol#L93-L110

Description: The `maxDeposit`, `maxWithdraw`, `maxMint`, `maxRedeem` functions use the inherited ERC4626's standard implementation. There are cases where Morpho paused supplying and withdrawing, and according to the EIP4626, these functions must return 0 in this case:

> `maxDeposit` - MUST factor in both global and user-specific limits, like if mints are entirely disabled (even temporarily) it MUST return 0. EIP4626

Recommendation: Consider overwriting these functions by checking Morpho-specific behavior to be compliant to the EIP. Think about if it makes sense to also respect Aave-v3's supply caps, however, these are not a vault limit in the case of P2P matchings on Morpho.

Morpho: Given the complexity and gas cost that it would bring, I tend to think that we should not implement it and keep things simple even though in rare case the standard is not met (ie it would revert instead of returning 0).

Spearbit: Acknowledged.

5.3 Low Risk

5.3.1 DDoS Morpho rewarding system scenarios that prevent users from accruing any rewards

Severity: Low Risk

Context: Morpho.sol#L259-L282, RewardsManager.sol#L428, RewardsManager.sol#L393

Description: There are different level of DDoS attack surfaces, some of them are on Aave directly and some of them are on the Morpho's system.

In general, both Aave and Morpho have two different indexes, one for the `(asset, reward)` combo and one for the `(asset, reward, user)` combo. On both platforms, both of those indexes are updated when the user performs an action that changes the user's balance (mint/burn/transfer `AToken` or `DebtToken`) or an action that claims (and accrues) the rewards.

There are different points in both platforms codebase where, if misconfigured, the indexes could be not updated resulting in a loss for all the users (if the `(asset, reward)` is not updated) or the single user (if `(asset, reward, user)` is not updated)

- Scenario 1: the Aave platform is DDoSed preventing the `(asset, reward)` index to increasing

In particular, `_updateRewardData` calls `_getAssetIndex` that will calculate the new `(asset, reward)` index based on the elapsed time

```
uint256 currentTimestamp = block.timestamp > distributionEnd
  ? distributionEnd
  : block.timestamp;
uint256 timeDelta = currentTimestamp - lastUpdateTimestamp;
uint256 firstTerm = emissionPerSecond * timeDelta * assetUnit;
assembly {
  firstTerm := div(firstTerm, totalSupply)
}
return (oldIndex, (firstTerm + oldIndex));
```

If `firstTerm < totalSupply` the index is not updated but the `rewardData.lastUpdateTimestamp` (in `_updateRewardData`) will be updated anyway to `block.timestamp`.

Considering that the minimum value for `timeDelta` is ~12 seconds (on ETH L1) this scenario is possible if `emissionPerSecond` is low enough and `totalSupply` is high enough.

In this case, all the users will not accrue any rewards for the combo `(asset, reward)`. If the emission is not well configured, it's very likely that this will happen considering that, as we said, Aave updates those indexes each time a user performs an operation that mint/burn/transfer tokens.

- Scenario 2: the Aave platform is DDoSed preventing the `(asset, reward, user)` index from accruing enough rewards

Let's assume that the `emissionPerSecond` parameter is configured in a way that `(asset, reward)` index is "safe" and will increase even if updated each block.

When those mint/burn/transfer or reward claims happen, the user's index for a specific `(asset, reward)` is also updated in `_updateUserData`

```
function _updateUserData(
  RewardsDataTypes.RewardData storage rewardData,
  address user,
  uint256 userBalance,
  uint256 newAssetIndex,
  uint256 assetUnit
) internal returns (uint256, bool) {
  uint256 userIndex = rewardData.usersData[user].index;
  uint256 rewardsAccrued;
  bool dataUpdated;
  if ((dataUpdated = userIndex != newAssetIndex)) {
    // already checked for overflow in _updateRewardData
    rewardData.usersData[user].index = uint104(newAssetIndex);
    if (userBalance != 0) {
      rewardsAccrued = _getRewards(userBalance, newAssetIndex, userIndex, assetUnit);

      rewardData.usersData[user].accrued += rewardsAccrued.toUint128();
    }
  }
  return (rewardsAccrued, dataUpdated);
}
```

If the `(asset, reward)` index has increased compared to the `(asset, reward, user)` index, it means that the user has accrued some rewards. Aave updated the user's index and calculated the amount of rewards accrued in `_getRewards`

```
function _getRewards(
  uint256 userBalance,
  uint256 reserveIndex,
  uint256 userIndex,
  uint256 assetUnit
) internal pure returns (uint256) {
  uint256 result = userBalance * (reserveIndex - userIndex);
  assembly {
    result := div(result, assetUnit)
  }
  return result;
}
```

Like before, the resulting `result` (amount of rewards accrued by the user in the time delta between the blocks) is equal to zero, the user does not accrue any rewards even if he should (`reserveIndex - userIndex > 0`)

This scenario is possible (because of round errors) if the user's balance is low enough, the diff between the indexes is low enough (we saw before that this depends on `emissionPerSecond`, `totalSupply` and `timeDelta` that could be as low as ~12 seconds)

In any case, the `_updateUserData` function will update the user's index to the asset's index without checking if the user had really accrued any rewards (`rewardsAccrued > 0`).

If the user performs these operations with a time delta not big enough, he could risk to not accruing rewards at all.

The problem in the Morpho Platform On Morpho, the "Morpho user" on Aave is the aggregation of all the user's balances. This decreases the possibilities that the `_getRewards` function returns an amount of rewards equal to zero. The problem for the single Morpho users remains the same on the Morpho `RewardsControler` that minim the same logic of Aave. The single user on the Morpho platform could see their accrued rewards not being increased if it performs the operations within a small time delta between each of them.

The problem is even aggravated by the fact that Morpho allows anyone to call `Morpho.claimRewards` on behalf of a Morpho user's without any authorization checks (unlike Aave `RewardsController` contract that checks if the `onBehalf` has authorized `msg.sender` to claim the rewards on behalf of him).

This means that anyone could DDoS the `onBehalf` user by calling each block the `Morpho.claimRewards` and possibly preventing the user from accruing any rewards if the `onBehalf` balance is low enough and the reward configuration on Aave (`emissionPerSecond`, `totalSupply`, `assetUnit`) allows is within a block timespan of 12 seconds.

Note: on different chains, the minimum value of `timeDelta` (seconds between two blocks) could be lower than ~12 seconds. For example, on Polygon it's ~2 seconds.

Recommendation: A change that could alleviate the possibility of a scenario where the user loses the accrual of rewards would be to not allow the `Morpho` contract to claim rewards `onBehalf` of anyway.

This change would have at least two side effects that require additional changes:

- `SupplyVault` should have a function that allows an external actor to directly call the `_MORPHO.claimRewards` function because after the PR no one would be able to call `_MORPHO.claimRewards(assets, address(supplyVault))`
- The `BulkerGateway` `_claimRewards` function will not work anymore because the `Bulker` does not own any assets and the user cannot specify anymore the `onBehalf` parameter

Morpho: I believe we can acknowledge this issue as a low probability of happening considering the reward assets traditionally listed by Aave and the fact that Aave doesn't incentivize liquidity on mainnet as of today. We can still take actions following a change of reward policy, with the upgrade required by PR 860.

Spearbit: Acknowledged.

5.3.2 Users that supply via `SupplyVault` will not be eligible and be able to claim Aave rewards

Severity: Low Risk

Context: SupplyVault.sol

Description: When users supply/borrow directly from `Morpho` they will be eventually being eligible for Aave's rewards that will be claimed via `Morpho.claimRewards`.

By supplying to Morpho through the `SupplyVault`, users are not directly eligible for those rewards because the one that is supplying to `Mopho` is the vault itself. Because of that, the only way to claim rewards is to call `Morho..claimRewards([assets], address(supplyVault))`. After that, users can only skim those rewards to the `_recipient` of the `SupplyVault` and not directly to the original supplier (user that have supplied through the vault).

Recommendation: Morpho should document the fact that user's that supply via the `SupplyVault` will not be able to claim directly the Aave rewards. Morpho should implement an off-chain mechanism that allows vault's users to claim those rewards once the vault has claimed them and transferred to the `_recipient` address.

Morpho: It will be documented and we'll implement en offchain mechanism so I think we can set it as acknowledged.

Spearbit: Acknowledged.

5.3.3 Vault's `totalAsset` increase after supply can be different from `amount` parameter

Severity: Low Risk

Context: SupplyVault.sol#L132, MorphoInternal.sol#L182

Description: The user specifies an `assets` amount parameter in `deposit` that is used for the shares computation. However, the actual `totalAssets` contribution of the user can be different due to rounding errors:

- Aave / Morpho credits an `assets.rayDiv(index)` supply balance to the vault (can even be zero as Morpho performs an early return).
- `totalAssets` calls `_MORPHO.supplyBalance(_underlying, address(this))` which scales the scaled balance back to an underlying amount by computing `scaledBalance.rayMulDown(index)`.

The `totalAssets` increase is thus roughly `assets.rayDiv(index).rayMulDown(index)`, which can be different from the initial `assets` parameter. It's even possible to mint shares by not increasing `totalAssets`, for example, depositing 1 asset can mint 1 share (as the initial deposit mints assets <> shares 1-to-1), but given an index of `2.2`, the `_MORPHO.supplyBalance` does not increase as `1.rayDiv(2.2e27) = 0`. It's possible to inflate the shares and grow a bigger share on the existing `totalAssets` each time but the cost of the attack is still the `assets` amount that ends up stuck in Morpho.

Recommendation: The maximum difference `|assets - totalAssetsIncrease|` is bound by the `index`, so the attack doesn't change the share price from the expected share price much given a non-trivial initial `totalAssets` deposit and should not be profitable overall. One could consider minting / burning shares according to the actual `totalAssets` change after supplying/withdrawing.

Morpho: Unfortunately, OZ's ERC4626 isn't built for this and it would require rewriting the whole vault. Considering the impact is low and the attack is not profitable, I don't think it's worse spending time and resources on this. We acknowledge this issue.

Spearbit: Acknowledged.

5.3.4 Reentrancy can steal temporary balances

Severity: Low Risk

Context: BulkerGateway.sol#L237-L239C25

Description: There is a reentrancy on the `receiver` parameter who could steal temporary balances left in the contract for subsequent actions but it requires the user to specify a malicious receiver.

Recommendation: Consider adding a reentrancy guard on the `execute` function for added security.

Morpho: I'd say it's the user's own fault, not sure if it's worth protecting against with reentrancy guards.

Spearbit: Acknowledged.

5.3.5 `BulkerGateway` allows the user to leave funds in the contract after calling `execute`

Severity: Low Risk

Context: BulkerGateway.sol#L74-L83

Description: The `BulkerGateway` contract `execute` functions allow the user to perform a sequence of actions within the same transaction. Those actions, if not performed in the correct order, with the proper action's configuration and with the proper inclusion of `skim` final calls, could allow the user to leave funds in the contracts.

Those funds can be later (with a following tx) be used or skimmed by another user, making the original user (of the first `execute`) lose funds.

Recommendation: Morpho should consider two options:

1) Implement an on-chain check that prevents the user from leaving ETH or ERC20 tokens inside the contract after the execution of all the actions
2) Document and disclose this possibility in the contract and in the documentation. In addition to that, Morpho should ensure that the UI on the Morpho website, that interacts with the contracts, includes all the needed actions that will skim the funds held by the contract as the result of the internal actions.

Morpho: Considering the bulker is made to interact with plenty of ERC20s, keeping track of what ERC20 the user interacted with during a single tx should be saved in the bulker's storage. It requires non-trivial code, adds complexity and involves additional gas. For this reason, I'd rather warn users about this edge case in the appropriate documentation.

Added a comment in the BulkerGateway contract.

Spearbit: Acknowledged.

5.4 Gas Optimization

5.4.1 Vault uses two different storage variables for the underlying

Severity: Gas Optimization

Context: SupplyVault.sol#L28, ERC4626Upgradeable.sol#L30

Description: The supply vault inherits from `ERC4626Upgradeable` which uses the storage address `_asset` for the underlying, and the `SupplyVault` itself defines the storage address `_underlying` for the same underlying.

Recommendation: Consider using the inherited `_asset` (through `asset()`) and removing the `SupplyVault._-underlying` storage field.

Morpho: Recommendation implemented in PR 601.

Spearbit: Verified.

5.4.2 gas saving with unchecked{++i}

Severity: Gas Optimization

Context: RewardsManager.sol#LL77C1, RewardsManager.sol#L445

Description: `++i/i++` should be `unchecked{++i}/unchecked{i++}` when it is not possible for them to overflow, as is the case when used in for- and while-loops

```
for (uint256 i; i < assets.length; ++i) {...}
```

Recommendation:

```
for (uint256 i; i < assets.length;) {
  ...
  unchecked{++i} ;
}
```

Morpho: We tend to avoid doing this except in low level libraries because it makes the code less readable and thus makes it easier to make mistakes.

Spearbit: Acknowledged.

5.5 Informational

5.5.1 The `SupplyVault.skim` function allows users to transfer the vault shares to the configured `_recipi-ent`

Severity: Informational

Context: SupplyVault.sol#L84-L94

Description: The `skim` function allows anyone to transfer any `ERC20` tokens owned by the `SupplyVault` to the state variable `recipient` when this variable is not equal to `address(0)`. Given that this variable can be updated only by the **owner** of the `SupplyVault`, we can assume that it will be updated to an address **trusted** by Morpho itself.

Because `skim` can transfer **any** `ERC20` token, it means that it can also transfer the `SupplyVault` share itself.

If the `SupplyVault` is initialized with `initialDeposit > 0` it will automatically mint to itself some amount of vault's shares. Those shares, owned by the vault, can be skimmed by any user and sent directly to the `_recipient` address.

Recommendation: Morpho can consider two options:

1) Keep allowing users to skim vault shares and skim the initial minted shares to a trusted account that cannot burn them.

2) Remove the ability to skim the vault shares with the consequences that users who mistakenly send shares to the vault will not be able to rescue them to the `_recipient` address.

Spearbit: The PR 108 forces the `ERC4626UpgradeableSafe` to accept a custom `recipient` address when the contract is initialized. If `initialDeposit > 0`, the initial shares minted with the initial deposit will be sent to the `recipient` address.

Because `ERC4626UpgradeableSafe` inherits from OpenZeppelin `ERC4626Upgradeable` that inherits from `ERC20Upgradeable` if `recipient` is equal to `address(0)` the whole transaction will revert.

The PR 857 forces the `SupplyVault` to initialize the `ERC4626UpgradeableSafe` with the `recipient` equal to the address `address(0xdead)`. The result is that the initial shares will be minted and transferred to the "burn address" `0xdead`.

Morpho is still allowing externals actors to `skim` the vault's shares owned by the vault but by implementing those PRs is preventing minting the initial shares to the vault itself that would be later skimmable by users.

5.5.2 Consider adding an event to `SupplyVault.initialize` and user internal setters to leverage sanity checks/event emissions

Severity: Informational

Context: SupplyVault.sol#L57-L77

Description: The current implementation of `SupplyVault.initialize` does not perform any input sanity checks and does not emit any events.

Like described in PR 22 the function should prevent the deployer from initializing a vault that uses an invalid `underlying` and set a too high `_maxIterations`.

The `_maxIterations` sanity check could be moved to the `setMaxIterations` setter and the `initialize` function could call it instead of directly assigning the input parameter to the state variable.

The `initialize` function could also emit an event that logs the actions and the parameters used to initialize the new vault.

Recommendation: Consider implementing the sanity check of the max iterations parameter into the `setMaxIterations` function, and replacing the state variable initialization with a call to the setter. Consider also emitting an event that logs the initialization of the Vault and which parameters have been used to initialize it.

Spearbit: The PR morpho-org/morpho-aave-v3#854 make the `initialize` execution to emit the `MaxIterationsSet` and `RecipientSet` event, but does not implement any check about the `newMaxIterations` value or emit any "global" event that recaps all the parameters used to initialize the vault (as a single event).

Morpho has stated that they will not implement a "gloabl" event for the initialization of the `SupplyVault`

> We consider the other 4 parameters immutable (only set at initialization: asset, name, symbol, initialDeposit), so we are not planning to emit a dedicated event for these parameters (an integrator should only have to query these values from initialization because they are not supposed to change, except via a `SELFDESTRUCT` or implementation upgrade) and will not add a lower/upper bound check on the `newMaxIterations` in the `_setMaxIterations` function.
>
> We rely on the DAO's goodwill to set an appropriate value, maximizing the expected portion of supply matched peer-to-peer while keeping the vault in service (not ending up in OOG or any other unexpected behaviors).

5.5.3 `SupplyVault` allows the deployment and initialization of a vault that will not be usable

Severity: Informational

Context: SupplyVault.sol#L57-L77

Description: The `initialize` function of the `SupplyVault` contract is not performing all the possible sanity checks that should be done on the function's input

- The `newUnderlying` parameter could not be a valid Morpho market
- The `newUnderlying` parameter could be a valid market (created on the Morpho contract) but the supply/withdraw functionality could be paused
- The `newMaxIterations` parameter could be a too high value that will make the `_MORPHO.supply` and `_MORPHO.withdraw` consume too much gas or revert because of Out of Gas exception

Recommendation: Morpho should consider reverting the execution of the transaction if

- `newUnderlying` is not a valid market (not created or not usable)
- set an upper bound limit to `newMaxIterations`

Morpho: Considering vaults should be managed and promoted by the DAO, we can safely rely on the DAO to verify the validity of a deployment and initialization before promoting vaults. The same applies for the lifetime of the vault (the DAO being able to update `maxIterations` at anytime): we can safely rely on the goodwill of the DAO.

Spearbit: Acknowledged.

5.5.4 `SupplyVault` missing natspec documentation

Severity: Informational

Context: SupplyVault.sol

Description: Some `external` and `internal` functions of the contract are missing part of its natspec documentation.

- `skim` function misses the natspec of the `tokens` input parameter.
- `setMaxIterations` function misses the natspec of the `newMaxIterations` input parameter.
- `setRecipient` function misses the natspec of the `newMaxIterations` input parameter.
- `MORPHO` function misses the natspec of the returned value.
- `recipient` function misses the natspec of the returned value.
- `underlying` function misses the natspec of the returned value.
- `maxIterations` function misses the natspec of the returned value.
- `totalAssets` function misses the natspec of the returned value.
- `_deposit` function misses the natspec documentation for `@notice`, all the `@param` and `@return`.
- `_withdraw` function misses the natspec documentation for `@notice`, all the `@param` and `@return`.

Recommendation: Morpho should add the missing natspec documentation of the listed functions.

Spearbit: Morpho has acknowledged the issue and has decided to not implement the recommendations.

5.5.5 `ISupplyVault` natspec errors/typos

Severity: Informational

Context: ISupplyVault.sol#LL22C18-L22C18, ISupplyVault.sol#L28

Description:

- In the `event Skimmed` the `amount` parameter is not the "The amount of rewards transferred" but rather the amount of tokens transferred.
- The natspec `@notice` description of the `error ZeroAddress` is wrong. This error is thrown not in the context of the `transferRewards` function (that is not present in this contract) but when the users try to execute one `address` is equal to `address(0)` when it should not.

Recommendation: Morpho should fix the typos and errors in the natspec documentation.

Spearbit: The recommendations have been implemented in the PR 853.

5.5.6 Inconsistent logic in handling when deadline == block.timestamp

Severity: Informational

Context: Morpho.sol#L249

Description: A signature's deadline sent via Permit2 is considered valid if `deadline >= block.timestamp`.

A signature's deadline sent via _MORPHO.approveManagerWithSig is considered valid if `deadline > block.timestamp`. Notably, if `deadline == block.timestamp` it is considered invalid.

From the BulkerGateway, a user can issue signatures via either method, and the handling of when `deadline == block.timestamp` should be consistent between the two.

Recommendation: In `_MORPHO.approveManagerWithSig`, set the logic to match that of `Permit2`.

Morpho: True it would have been cleaner to harmonize, but we won't upgrade for that I think. We acknowledge the issue.

Spearbit: Acknowledged.

5.5.7 Unused error

Severity: Informational

Context: IBulkerGateway.sol#L9

Description: Unused error.

Recommendation: Remove the error.

Morpho: Fixed in PR 852.

Spearbit: Verified.

5.5.8 Unused import

Severity: Informational

Context: ISupplyVault.sol#L6

Description: Unused import.

Recommendation: Remove the import.

Morpho: Fixed in PR 852.

Spearbit: Verified.

5.5.9 Missing argument name for interface

Severity: Informational

Context: IWETH.sol#L7

IWSTETH.sol#L27

IWSTETH.sol#L26

Description: There is a missing argument name on the `withdraw` interface function. Also the `wrap` and `unwrap` function arguments don't match exactly with the contract the interface is meant for interacting with.

Recommendation:

1. On the IWETH.sol `withdraw` function the argument name `wad` should be added here for code consistency as all the other functions inside this interface have named arguments.
2. In the IWSTETH.sol interface `stETHAmount` should be named `_stETHAmount` and `wstETHAmount` should be named `_wstETHAmount` like in the original WstETH contract.

Morpho:

1. Fixed in PR 858.
2. Acknowledged, but won't implement.

Spearbit: Verified, and Acknowledged.

5.5.10 `ISupplyVault` events are missing `indexed` parameters

Severity: Informational

Context: ISupplyVault.sol#L17, ISupplyVault.sol#L23

Description: Both `RecipientSet` and `Skimmed` events are missing the `indexed` keyword for the `address` parameters. When event parameters are indexed, it allows dApps and monitoring tools to better filter those events.

Recommendation: Morpho should consider setting `recipient` as `indexed` in the `RecipientSet` event and both `token` and `recipient` as `indexed` in the `Skimmed` event.

Morpho: Fixed in PR 853.

Spearbit: Verified.

5.5.11 `BulkerGateway` should avoid calling `_MORPHO.claimRewards` if the internal action `_claimRewards` is called with `assets.length` equal to zero

Severity: Informational

Context: BulkerGateway.sol#L274

Description: The call to `_MORPHO.claimRewards` should be avoided if the `assets.length` is equal to zero.

Recommendation: Consider avoiding the call to `_MORPHO.claimRewards` or reverting if `assets.length` is equal to zero.

Morpho: We acknowledge the issue, but we don't think it's worth implementing a change for it.

Spearbit: Acknowledged.

5.5.12 `BulkerGateway` should prevent the user from transferring funds to `address(0)`

Severity: Informational

Context: BulkerGateway.sol#L239, BulkerGateway.sol#LL260C24-L260C36, BulkerGateway.sol#L268, BulkerGateway.sol#L274

Description:

- `Solmate SafeTransferLib.safeTransferETH` and `ERC20(asset).safeTransfer` do not internally check if the `receiver` is `address(0)`. The check is the responsibility of the `caller` (in this case the Bulker) or the implementation of the `ERC20` token itself (that should do it internally).
- `stETH` seems to internally check if the `recipient` is not `address(0)`
- `_MORPHO.claimRewards` internally does not check if `onBehalf` is `address(0)` but that specific address theoretically should have no balance inside Morpho
- `BulkerGateway._skim` does not check if the `receiver` is `address(0)` and leaves the responsibility to the `ERC20` implementation

Note: `_MORPHO.supply`, `_MORPHO.supplyCollateral`, `_MORPHO.borrow`, `_MORPHO.repay`, `_MORPHO.withdraw`, `_MORPHO.withdrawCollateral` will internally revert if the on `onBehalf/receiver` parameter is indeed `address(0)`.

Recommendation:

- Consider checking in `_unwrapEth` internal action that the `receiver` parameter of the `safeTransfer` call is not `address(0)`
- Consider checking in the `_unwrapStEth`, `_skim` and `_claimRewards` internal actions that the `receiver/onBehalf` parameter is not equal to `address(0)`.

Spearbit: The recommendations have been implemented in the PR 856. The PR has implemented the following changes

- Removed `amount == 0` check in the internal functions `_supply`, `_supplyCollateral`, `_borrow`, `_repay` and `_withdraw`. Morpho has stated that the check has been removed because Morpho itself will internally check it on the Morpho platform.
- Added `onBehalf == address(this)` check in the internal functions `_supply`, `_supplyCollateral`, `_repay` and `_claimRewards`
- Added `receiver == address(0)` check in the internal functions `_unwrapEth`, `_unwrapStEth` and `_skim`

The PR 862 removes the `amount == 0` check from `_withdrawCollateral` to be consistent with the other removals reported in the PR 856.

5.5.13 Consider emitting events for each `BulkerGateway` execution

Severity: Informational

Context: BulkerGateway.sol#L74-L83

Description: Event emissions are useful to be later used by integrators, dApps and monitoring tools. The current implementation of `BulkerGateway` does not emit any event.

Morpho should consider emitting an event in

- `execute` to monitor the execution of `execute` and track who has called it and with how many actions.
- Every single internal action emitting a specific event for it with the parameters decoded from the `data`.

Recommendation: Consider adding events emission to cover the `execute` call and all the internal actions performed by the user within the `execute` call.

Morpho: We think it's unnecessary to add events to the Bulker. Each Bulker action is necessarily associated to an event emitted from the called contract, and therefore no ambiguous events occur using this method. We acknowledge the issue.

Spearbit: Acknowledged.