Morpho Vaults v2 Fix Review Security Review

Auditors

MiloTruck, Lead Security Researcher

Saw-mon and Natalie, Lead Security Researcher

Report prepared by: Lucas Goiriz

November 26, 2025

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 trustless and efficient lending primitive with permissionless market creation.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Morpho Vaults v2 Fix Review 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 32 days in total, Morpho engaged with Spearbit to review the vault-v2 protocol. The review focused on the following scope:

- PR 723.
- PR 724.

In this period of time a total of **13** issues were found.

Summary

Project Name	Morpho
Repository	vault-v2
Commit	ce661d82
Type of Project	DeFi, Vaults
Audit Timeline	Aug 10th to Sep 12th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	7	1	6
Gas Optimizations	0	0	0
Informational	6	1	2
Total	**13**	**2**	**8**

The Spearbit team reviewed Morpho's `vault-v2` protocol holistically on commit hash 6f2af660 and concluded that all findings were addressed and no new vulnerabilities were identified.

5 Findings

5.1 Low Risk

5.1.1 Convert to shares and assets take into consideration the fees

Severity: Low Risk

Context: VaultV2.sol#L636-L644

Description: when converting shares to assets to vice versa using `convertToShares` and `convertToAssets` call the corresponding preview functions which call the `accrueInterestView()` and finally the performance and management fee shares are used when deriving the final value. According to `EIP-4626` these conversion endpoints:

> MUST NOT be inclusive of any fees that are charged against assets in the Vault.

Recommendation: Either make sure the `convertToShares` and `convertToAssets` do not take into consideration the fees to perhaps a `@dev` NatSpec comment should be added to highlight this.

Morpho: Fixed in PR 724.

Spearbit: NatSpec comments have been added.

5.1.2 Allocation role

Severity: Low Risk

Context: VaultV2.sol#L166, VaultV2.sol#L324-L328, VaultV2.sol#L502-L503, VaultV2.sol#L529-L530, VaultV2.sol#L554-L555

Description: Currently there is only one role identified by `isAllocator` that can call:

- `allocate`.
- `setLiquidityAdapterAndData`.
- `deallocate`.

The role is not fine-grained enough. It might make sense to define separate role for each of these endpoints as some are more critical that the others. Perhaps even an allocation role and a deallocation role.

Moreover the `setIsAllocator` endpoint is time locked. That means for both assigning and removing an entity from the allocation role, the curator and the vault might have to wait till the delegated executable timestamp after the intent of submitting to call `setIsAllocator` is reached. This can introduce potential problems when the curator wants to remove an allocation role for a potentially malicious entity. There are other operations related to change of configuration for the vault that when the operation would potentially expose the vault to some risks, that operation is time locked. But there are also the anti-operations (the reverse operations) where performing those operations would remove some potential risks from the vault. These reverse operations are not time locked currently and can be called/performed either by the curator or one of the sentinels. In that light, it would make sense that removing an entity from the allocation role should in theory remove some unwanted risk from the vault and thus, perhaps should not be behind a time lock.

Recommendation:

1. Assign more segregated roles for the allocation endpoints.
2. Allow the curator or one of the sentinels to remove an allocator without going through a time-lock. Assigning an allocation role to an entity should still be time locked.
3. The curator should also be able to call `deallocate`.

If either of these recommendations are not applied, it might make sense to add documentation to expose these potential scenarios to the users and the devs.

Morpho: As allocators are only trusted for future yield, we don't think that there is a use-case to timelock adding them. though there could be one for removing them (not that clear neither), thus we decided to acknowledge this feature. Added a comment in PR 730.

About other comments, we acknowledge them as well. We think that this role is well defined, and if someone needs separation it's not too hard to do on top.

Spearbit: Acknowledged.

5.1.3 Some market params with non-zero shares might get removed from the `marketParamsList`

Severity: Low Risk

Context: MorphoMarketV1Adapter.sol#L34, MorphoMarketV1Adapter.sol#L77-L78, MorphoMarketV1Adapter.sol#L98-L99, MorphoMarketV1Adapter.sol#L106-L119, MorphoMarketV1Adapter.sol#L134-L140, MorphoVaultV1Adapter.sol#L108-L110

Description: In `allocate` and `deallocate` if the `newAllocation` is `0`:

1. Allocation flow: If the `oldAllocation` is also `0` and the `marketParams` is not in the `marketParamsList`, in the allocation flow it would not get added to the list. Assume the amount of assets provided is a where the total assets and total supply for this market are A and S respectively. Then the shares minted would be $s = \left\lfloor \frac{a(S+10^6)}{A+1} \right\rfloor$ and assume that $1 \leq s$, ie a non-zero share is minted for this allocation but the returned new allocation is 0. That means:

$$0 = \left\lfloor \frac{s(A+a+1)}{S+s+10^6} \right\rfloor$$

Let:

$$\epsilon = \frac{a(S+10^6)}{A+1} - \left\lfloor \frac{a(S+10^6)}{A+1} \right\rfloor \in [0,1)$$

We get:

$$1 \leq s = \frac{a(S+10^6)}{A+1} - \epsilon \Rightarrow$$
$$\frac{A+1}{S+10^6} \leq a - \epsilon \left(\frac{A+1}{S+10^6} \right)$$

from (1) we can deduce that:

$$(1) \Rightarrow a < 1 - \epsilon \left(\frac{A+1}{(S+10^6)(A+1+a)} \right) + \epsilon \left(\frac{A+1}{S+10^6} \right)$$

or:

$$\frac{A+1}{S+10^6} \leq a - \epsilon \left(\frac{A+1}{S+10^6} \right) < 1 - \epsilon \left(\frac{A+1}{(S+10^6)(A+1+a)} \right) < 1$$

Thus:

$$(4,5) \Rightarrow a < 2 \Rightarrow a \in \{0,1\}$$

But a cannot be 0 since then s would also end up being 0. So a should be 1 and we should have:

$$\frac{A+1}{S+10^6} < 1$$

and

$$\frac{S+10^6}{A+1} \notin \mathbb{Z}$$

Thus if $\frac{S+10^6}{A+1} \in (1,\infty) \setminus \mathbb{Z}$ and the asset to be allocated $a = 1$, a non-zero share would be minted, but the market parameter would not get added to the list, since right after supplying the value of those shares would be 0 although in the future they might accumulate more value.

Also in some cases, one might have a non-zero old allocation for a market parameter, but the new allocation would turn 0 and thus the market parameter would get removed from the list, even though there might have been some non-zero shares left.

2. Deallocation flow: Same as the allocation flow, it might be the case that the current new allocation might be 0 but there are still non-zero shares left for the market parameter in the context.

Note that in this route market parameter could not get added to the list since for that to happen the old allocation must have been 0 and the new allocation should end up non-zero. But after the deallocation flow in the adapter, in the main deallocation internal flow of the vault v2 we have:

```
for (uint256 i; i < ids.length; i++) {
    Caps storage _caps = caps[ids[i]];
    require(_caps.allocation > 0, ErrorsLib.ZeroAllocation());
    _caps.allocation = (int256(_caps.allocation) + change).toUint256();
}
```

which makes sure the must fine-grained unique allocation cap for this market paramter and adapter (ie the old allocation) must have been non-zero before getting updated to the new one.

So in the allocation flow, the only possible outcome is that the market parameter could only potentially get removed from the list. Addition is not possible in this flow (which also makes sense).

Recommendation: For both flows, one can see that a market parameter could potentially get removed from the `marketParamsList` even though for that paramter a non-zero share amount might have been left in the `Morpho` market contract. Therefore in the interest accural routes when the `realAssets()` of this adapter is called, the value of those left-over shares would not get accounted for.

Possible solutions would be to only remove the market params from the list if the share for the market parameter would end up being 0 (this applies to both allocation and deallocation flows). Although having lots of market paramters that would have small values might add up a lot of gas when querying the `realAssets()` of this adapter.

Also note that if the values not considered for the removed market parameters in question become significant on the next call to `allocate` they would get added back to the list even if one calls `allocate` with 0 assets.

In the allocation flow, it might also make sense to `revert` if `newAllocation` is `0`. This suggestion does not apply for the deallocation flow.

Morpho: Acknowledged. Related comment has been added in PR 733

Spearbit: Acknowledged.

Footnote: The above logic also applies to `MorphoVaultV1Adapter` where the new allocation might be `0` but the adapter might still have non-zero shares in the yield source. And at a future timestamp the call to `IERC4626(morphoVaultV1).previewRedeem(IERC4626(morphoVaultV1).balanceOf(address(this)))` might return a non-zero value. But currently the decision to make this call or not in `realAssets()` is based on the last updated allocation which would have a stale value.

5.1.4 Revert if zero shares are minted for an adapter in a morpho market

Severity: Low Risk

Context: MorphoMarketV1Adapter.sol#L74

Description: In some cases when the `IMorpho(morpho).supply(...)` is called 0 shares could be minted for the `MorphoMarketV1Adapter`. Thus in these cases the `assets` provided would get donated to `Morpho`.

Recommendation: Use the returned values from `IMorpho(morpho).supply(...)` to make sure there are non-zero shares minted.

Morpho: Acknowledged. There is this comment already:

```
/// @dev This adapter must be used with Morpho Market v1 that are protected against inflation attacks
↪  with an initial
/// supply. Following resource is relevant:
↪  https://docs.openzeppelin.com/contracts/5.x/erc4626#inflation-attack.
```

And note that preventing minting zero shares is not enough, you can loose as much but still mint one share (etc...).

Spearbit: Acknowledged.

5.1.5 Zero shares could be minted for a non-zero provided asset

Severity: Low Risk

Context: VaultV2.sol#L692

Description: In the `mint` and `deposit` flow of the `VaultV2` there might be cases where the share to be minted for the user might end up being `0` even when the provided `asset` is non-zero.

Recommendation: It would be best to revert in `enter` if `shares` is `0`.

Morpho: Acknowledged. The vault allows no-op everywhere.

Spearbit: Acknowledged.

5.1.6 `firstTotalAssets` analysis

Severity: Low Risk

Context: VaultV2.sol#L30-L34, VaultV2.sol#L179, VaultV2.sol#L563-L571, VaultV2.sol#L578

Description: In the NatSpec comments we have:

```
/// FIRST TOTAL ASSETS
/// @dev The variable firstTotalAssets tracks the total assets after the first interest accrual of the
↪  transaction.
/// @dev Used to implement a mechanism that prevents bypassing relative caps with flashloans.
/// @dev This mechanism can generate false positives on relative cap breach when such a cap is nearly
↪  reached,
/// for big deposits that go through the liquidity adapter.
```

The comment is not quite accurate. We have:

```
function accrueInterest() public {
    (uint256 newTotalAssets, ...) = accrueInterestView();
    // ...
    _totalAssets = newTotalAssets.toUint128();
    if (firstTotalAssets == 0) firstTotalAssets = newTotalAssets;
    // ...
    lastUpdate = uint64(block.timestamp);
}
```

```
function `accrueInterestView()` ... {
    if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
    //...

    return (newTotalAssets, ...);
}
```

And so the `transient` storage parameter `firstTotalAssets` at any specific point in the call frame tree of a root transaction is either `0` or the first non-zero `newTotalAssets` in the ordered call-tree frame. Thus it could be `0` for a few frames and then as soon as it becomes non-zero it would stay the same value and the calls to `accrueInterestView()` would return early from this point on.

- Example 1: assume A_0 is a non-zero value. Initially in the beginning of the root tx T, one starts with A_0 as the `_totalAssets`, then the maybe due to some losses, `newTotalAssets` becomes `0` and thus for all the following call frames if we enter back into `accrueInterestView()` and `accrueInterest()` both `_totalAssets` and `newTotalAssets` stay `0`, except somehow we increase `_totalAssets` to a non-zero value. This would only be possible through `mint` or `deposit` at this point. After this `firstTotalAssets` would be set to a non-zero value:

$$A_{tot} = A_0 \to 0 \to \cdots \to 0 \to A_1 = A_{tot}^{1st}$$

- Example 2: This is like example 1. but at the a state where there are no total assets in the vault due to perhaps using a vault that is freshly deployed or perhaps due to the fact that all previous share holders have exited the vault.

  As one can see in both examples the value of the `firstTotalAssets` is not the value new total assets calculated in the first visit of `accrueInterestView()` in the whole call-frame tree of a root transaction. Moreover until this value is set to a non-zero value the flow in `accrueInterestView()` performs all the operations without exiting early.

```
function allocateInternal(/*...*/) internal {
    // ...

    accrueInterest();

    // ...

    for (...) {
        // ...

        require(/*...*/ ||
            _caps.allocation <= firstTotalAssets.mulDivDown(_caps.relativeCap, WAD),
        // ...);
    }
}
```

The value of `firstTotalAssets` is used to limit the relative cap during an allocation process. Thus due to above the correct value is not used in the above examples. The value that should have been used, must have been 0. But currently the last non-zero value is used.

Recommendation: One can store a flag in `firstTotalAssets` to indicate that the call-tree flow has gone through the `accrueInterest()` process even though the new total assets could be 0. In all cases, we can set `firstTotalAssets` to store the returned value from `accrueInterestView()` in its `[1:255]` bits and the first bit would be flag bit indicating that `accrueInterest()` was called. Here is a rough patch:

```
diff --git a/src/VaultV2.sol b/src/VaultV2.sol
index c86ec7f..90e68a4 100644
-- a/src/VaultV2.sol
++ b/src/VaultV2.sol
@@ -518,8 +518,10 @@ contract VaultV2 is IVaultV2 {
```

```
              require(_caps.absoluteCap > 0, ErrorsLib.ZeroAbsoluteCap());
              require(_caps.allocation <= _caps.absoluteCap, ErrorsLib.AbsoluteCapExceeded());
+
+             uint256 cleanedFirstTotalAssets = firstTotalAssets » 1;
              require(
-                 _caps.relativeCap == WAD || _caps.allocation <=
↪  firstTotalAssets.mulDivDown(_caps.relativeCap, WAD),
+                 _caps.relativeCap == WAD || _caps.allocation <=
↪  cleanedFirstTotalAssets.mulDivDown(_caps.relativeCap, WAD),
                  ErrorsLib.RelativeCapExceeded()
              );
          }
  @@ -561,10 +563,11 @@ contract VaultV2 is IVaultV2 {
      /* EXCHANGE RATE FUNCTIONS */

      function accrueInterest() public {
+         if (firstTotalAssets != 0) return;
          (uint256 newTotalAssets, uint256 performanceFeeShares, uint256 managementFeeShares) =
          ↪  accrueInterestView();
          emit EventsLib.AccrueInterest(_totalAssets, newTotalAssets, performanceFeeShares,
          ↪  managementFeeShares);
          _totalAssets = newTotalAssets.toUint128();
-         if (firstTotalAssets == 0) firstTotalAssets = newTotalAssets;
+         if (firstTotalAssets == 0) firstTotalAssets = (newTotalAssets « 1) + 1;
          if (performanceFeeShares != 0) createShares(performanceFeeRecipient, performanceFeeShares);
          if (managementFeeShares != 0) createShares(managementFeeRecipient, managementFeeShares);
          lastUpdate = uint64(block.timestamp);
```

Moreover, one can modify the accrue interest flow even further by strategically returning early if the last timestamp has not changed:

```
  diff --git a/src/VaultV2.sol b/src/VaultV2.sol
  index c86ec7f..12aefde 100644
-  -- a/src/VaultV2.sol
+  ++ b/src/VaultV2.sol
  @@ -518,8 +518,10 @@ contract VaultV2 is IVaultV2 {

              require(_caps.absoluteCap > 0, ErrorsLib.ZeroAbsoluteCap());
              require(_caps.allocation <= _caps.absoluteCap, ErrorsLib.AbsoluteCapExceeded());
+
+             uint256 cleanedFirstTotalAssets = firstTotalAssets » 1;
              require(
-                 _caps.relativeCap == WAD || _caps.allocation <=
↪  firstTotalAssets.mulDivDown(_caps.relativeCap, WAD),
+                 _caps.relativeCap == WAD || _caps.allocation <=
↪  cleanedFirstTotalAssets.mulDivDown(_caps.relativeCap, WAD),
                  ErrorsLib.RelativeCapExceeded()
              );
          }
  @@ -561,10 +563,15 @@ contract VaultV2 is IVaultV2 {
      /* EXCHANGE RATE FUNCTIONS */

      function accrueInterest() public {
+         if (firstTotalAssets != 0) return;
+         if (lastUpdate == block.timestamp) {
+             firstTotalAssets = (_totalAssets « 1) + 1;
+             return;
+         }
          (uint256 newTotalAssets, uint256 performanceFeeShares, uint256 managementFeeShares) =
          ↪  accrueInterestView();
```

```
            emit EventsLib.AccrueInterest(_totalAssets, newTotalAssets, performanceFeeShares,
            ↪  managementFeeShares);
            _totalAssets = newTotalAssets.toUint128();
-           if (firstTotalAssets == 0) firstTotalAssets = newTotalAssets;
+           if (firstTotalAssets == 0) firstTotalAssets = (newTotalAssets « 1) + 1;
            if (performanceFeeShares != 0) createShares(performanceFeeRecipient, performanceFeeShares);
            if (managementFeeShares != 0) createShares(managementFeeRecipient, managementFeeShares);
            lastUpdate = uint64(block.timestamp);
  @@ -575,8 +582,12 @@ contract VaultV2 is IVaultV2 {
        /// @dev The performance and management fees are taken even if the vault incurs some losses.
        /// @dev Both fees are rounded down, so fee recipients could receive less than expected.
        function accrueInterestView() public view returns (uint256, uint256, uint256) {
-           if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
            uint256 elapsed = block.timestamp - lastUpdate;
+
+           if (elapsed == 0 || firstTotalAssets != 0) {
+               return (_totalAssets, 0, 0);
+           }
+
            uint256 realAssets = IERC20(asset).balanceOf(address(this));
            for (uint256 i = 0; i < adapters.length; i++) {
                realAssets += IAdapter(adapters[i]).realAssets();
```

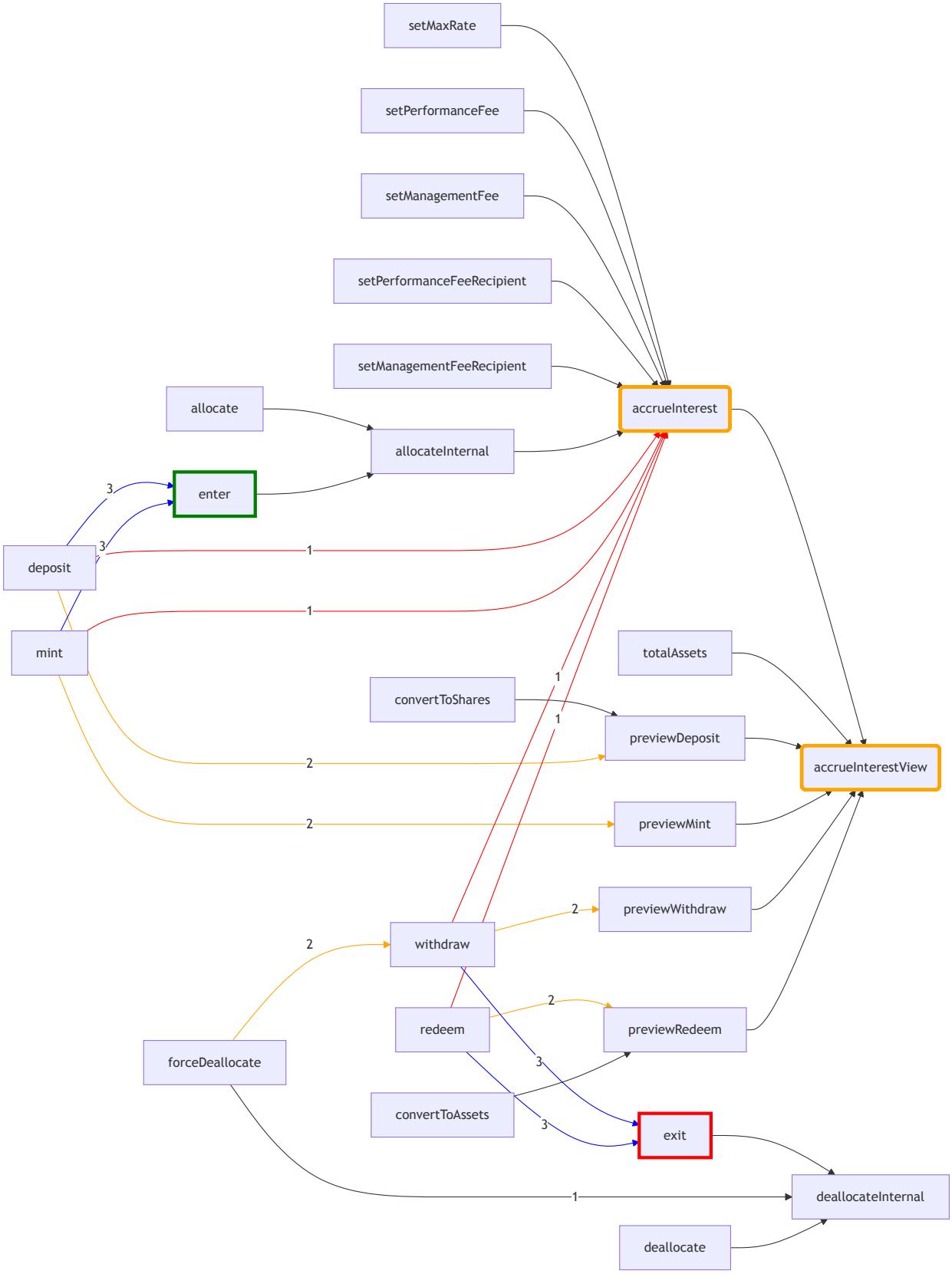
Morpho: Acknowledged.

Spearbit: Client has added the following comment in commit 2ed9336b:

> @dev The behavior of firstTotalAssets is different when the vault has totalAssets=0, but it does not matter. internally because in this case there are no investments to cap.

Footnote:

Call flow diagram:



Related Competition Findings:

Finding #	Title	Notes
#770	Allocators can bypass the relative cap check	
#53	Malicious allocator can manipulate `firstTotalAssets` to bypass relative cap policies	and many other dups
#817	Batch deposit calls will fail on a Vaults with Liquidity Adapter	
#855	FirstTotalAssets Stale Value Enables Relative Cap Bypass in Same-Transaction Deposits	
#847	Fund Loss via Relative Cap Validation Bypass in allocate() Function of VaultV2 Contract. Concern regarding the specs for relative caps being at `WAD`	X
#803	Morpho Vault V2: Missing `accrueInterest()` in `deallocateInternal` Leads to Potential Accounting Inconsistencies	X
#790	Relative Cap Bypass Through Rounding Error in `VaultV2`	Incorrect, should be rejected

And a few other ones that are not comepletely related. Search is based on looking for `firstTotalAssets` when on the finding page and pressing `CMD + K`.

Related test: `testRelativeCapManipulationProtection`.

5.1.7 A malicious adapter could break the allocation flow

Severity: Low Risk

Context: VaultV2.sol#L519-L524

Description: Some of the allocation ids returned are coarse allocation ids corresponding to multiple allocation routes. For example the allocation id for a specific collateral or the allocation id for the morpho market adapter.

parameter	description	type
a_i	allocation for i	`uint256`
c_i^{abs}	absolute cap of allocation i	`uint128`
c_i^{rel}	relative cap of allocation i	`uint128`

Now assume a malicious or buggy adapter returns a very high allocation change (even something like `type(uint128).max`), if that happens one would not be able to allocate any funds to any other allocation id j such that R_j has an id in R_i where i is the allocation used in the buggy adapter since the new allocation including the change would be bigger than in value that could fit into a `uint128` and thus bigger than $c_{i_f}^{abs}$.

An example would be for the specific collateral C id, if the above scenario happens. The vault would not be able to allocate to any market using any morpho adapter if the market parameters would use that specific collateral.

Recommendation: Perhaps bound-checks could be performed on the allocation changes. The checks mixed with some time and estimate analysis should be performed to make sure the above issue could not happen within a certain time range. But even this might not cover all the potential cases.

But also note that the above could only happen in some extreme cases when the values c_i^{abs} were already at or close to maximum. If they were low enough, one could remove the buggy adapter and add to the caps for the any incorrect amount of allocation change was introduced by the buggy adapter. This change would need to be applied to all j such that $R_j \cap R_i \neq \emptyset$.

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.2 Informational

5.2.1 Minor issues, typos, comments, events

Severity: Informational

Context: MorphoMarketV1AdapterFactory.sol#L20, VaultV2.sol#L549-L550, VaultV2.sol#L751-L754

Description/Recommendation:

Events:

1. MorphoMarketV1AdapterFactory.sol#L20: Also emit `morpho`.
2. VaultV2.sol#L552: The returned `ids` parameter from `deallocateInternal` is only used in `forceDeallocate` for the event `ForceDeallocate(..., ids, ...)` but this parameter is also emitted in `deallocateInternal`'s `Deallocate(..., ids, ...)` event and thus it can be deduced from the earlier `Deallocate` event. If it is not necessary to also re-emit this parameter in `ForceDeallocate`. The `ids` parameter can be removed from `ForceDeallocate` and also `deallocateInternal` would not need to return this value.

Morpho: Fixed and acknowledged.

Spearbit:

1. Has been fixed in PR 723.
2. Has been acknowledged. The client has added the `ids` in the event of `forceDeallocate` because it helps indexing (recovering the right associated deallocate isn't that easy) on purpose.

5.2.2 Token donations to adapters

Severity: Informational

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

Description/Recommendation: In the current two implementation of the adapters:

- `MorphoMarketV1Adapter`.
- `MorphoVaultV1Adapter`.

If at some point tokens are donated to these adapters the donated tokens would would not get used during the allocation and dellocation flows the assets for this flows are transiently moved to and from the adapter in atomic transactions from the parent vault.

Any donated assets can be skimmed out of these adapters using the `skimRecipient` set by the parent vault owner.

The exceptions are:

- For `MorphoVaultV1Adapter` if `morphoVaultV1` tokens/shares are donated they cannot be skimmed out and their values will be accounted for in the next allocation, deallocation or `realAssets()` query.
- Any donated tokens to `MorphoMarketV1Adapter` would not affect any of the accountings and can be skimmed out. Note that asset/funds can be directly supplied (donated) for loan or collateral tokens on behalf of the adapter in `Morpho` That would affect only the morpho market accountings that have been used by the privileged allocators in the `VaultV2`. Donations to unallocated market parameters would also not change any of the accountings in the current context in `VaultV2`.

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.2.3 Vault type

Severity: Informational

Context: VaultV2.sol#L584-L586

Description/Recommendation: With the new implementation of `VaultV2`, this vault is not a fixed interest type vault where those fixed interests are dripped into that can be shared. The current type could be called a bounded variable interest vault where the upper bound in on the variable interest is determined by `maxRate`. Also there are no safeguards regarding the max loss rate.

Morpho: Acknowledged.

Spearbit: Acknowledged.

5.2.4 Adapter Approvals

Severity: Informational

Context: MorphoMarketV1Adapter.sol#L48-L49, MorphoMarketV1Adapter.sol#L74, MorphoMarketV1Adapter.sol#L95, MorphoVaultV1Adapter.sol#L43-L44, MorphoVaultV1Adapter.sol#L69, MorphoVaultV1Adapter.sol#L87, VaultV2.sol#L540-L548

Description: Currently both adapter implementations give unlimited approvals to the:

- Parent vault.
- And their yield source.

1. In the allocation flow the approval for the asset token to the yield source is needed, since the yield source supply/deposit endpoints would need to transfer assets/tokens on behalf of the adapter.
2. In the deallocation flow, the deallocated assets are not transferred back to the parent vault. Instead the parent vault afterwards pull/transfer those deallocated tokens/assets from the adapter to itself afterwards.

Recommendation: One needs to evaluate these unlimited approval decisions. Alternatives would be:

1. The adapter would give atomic approvals equal to the amount of assets it would want to supply/deposit into the yield source and afterwards to make sure to reset the approval to `0`. This would eliminate potential cases where unnecessary approvals can be used in the future to pull more assets in some unexpected ways from the adapter. Currently such cases are not possible. Note this change would introduce extra gas cost for the allocation flow.
2. The adapter can transfer the deallocated assets in its `deallocate` endpoint and thus giving unlimited approval to the parent vault would not be necessary. Applying this change would put more trust into the adapter, but also currently the parent vault has no way of pulling unused extra tokens from the adapters (they can only be skimmed). This change would also introduce a change into the order of external call flows and storage checks updates for allocation amounts in the parent vault, thus extra caution and analysis would need to be applied.

5.2.5 Potential underestimates values might not enforce the desired allocation caps

Severity: Informational

Context: MorphoMarketV1Adapter.sol#L77, MorphoVaultV1Adapter.sol#L71-L75

Description: In both adapter implementations and also in general, one should try to if possible overestimate the new allocation value in the allocation flows. That way the allocation upper bounds used with the absolute and relative caps would be more strict.

Currently, in the implemented adapters endpoints are used that might slightly underestimate the values of the allocated yield sources. Thus the endorsed upper bounds might not be as accurate as one might hope.

Recommendation: To combat the above issue, the curator could enforce a slightly stricter upper bound. Although this might not be a solution that would work in all cases. Moreover, it would be great to document the above so the curators would be aware when adding new adapters to be used during the allocation and deallocation flows.

5.2.6 The allocation upper bound checks are not accurate for aggregated coarse ids

Severity: Informational

Context: VaultV2.sol#L513-L525

Description: In this finding we use the notations from the following finding from a previous review:

- Incorrect `allocation` calculation for `ids` corresponding to coarser set of routes.

  For a coarse id i we have:

  $$a_i = \sum_{i_f \in R_i} a_{i_f}$$

  An example of a coarse id would be:

  - Allocation id for a specific collateral.
  - Allocation id for the morpho market adapter (without identifying the specific market parameters).

  In each call to the current implementations of the `IAdapter`, `IAdapter(adapter).allocate(...)` only the change in allocation of one specific fine-grained allocation id is returned back, let's call this id $i_\star$. Then the allocations for the `ids` returns by the call get updated one by one and for each `id` I only the term related to the specific $i_\star$ gets updated. The other terms for the coarse id allocation sum keep their stale value. ie:

  $$a_i = \sum_{i_f \in R_i} a_{i_f} \rightarrow \left(\sum_{i_f \in R_i \setminus \{i_\star\}} a_{i_f} \right) + (a_{i_\star} + \blacksquare a) = a_i^{new}$$

  All the red terms above are stale. Thus the value a_i^{new} might be lower that its actual value if all the stale terms would have been updated to their live value. Thus the upper bound checks performed are not accurate.

  The only accurate upper bound checks would be only for the only fine-grained id $i_\star$, ie $a_{i_\star}$.

  For example the upper bound checks performed for an id to corresponds to a collateral C which is used in two different market parameters would not be accurate as only of the terms would be stale. Whereas the upper bound check performed for the specific market parameter of the morpho adapter which was used for the allocation call would be accurate.

Recommendation: The above can only be better documented. Applying a fix could be costly and complicated.