Security Assessment Report

Morpho Vault v2

December 2025

Prepared for Morpho Team

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Morpho Vault v2	https://github.com/morpho-org/vault-v2 https://github.com/morpho-org/morpho-blue-irm https://github.com/morpho-org/vault-v2-adapter-registries	425f6b1 PR 789 PR 139 PR 9	Solidity

Project Overview

This document describes the manual code review findings of **Morpho Vault v2**. The following contract list is included in our scope:

- `src/adapters/MorphoMarketV1AdapterV2.sol`
- `src/adapters/MorphoMarketV1AdapterV2Factory.sol`
- `src/adapters/interfaces/IMorphoMarketV1AdapterV2.sol`
- `src/adapters/interfaces/IMorphoMarketV1AdapterV2Factory.sol`
- `src/libraries/ErrorsLib.sol`
- `src/interfaces/IERC4626.sol`
- `src/adaptive-curve-irm/libraries/periphery/AdaptiveCurveIrmLib.sol`
- `src/MorphoMarketV1Registry.sol`
- `src/MorphoMarketV1RegistryV2.sol`
- `src/MorphoVaultV1Registry.sol`
- `src/RegistryList.sol`
- `src/interfaces/IMorphoMarketV1Registry.sol`
- `src/interfaces/IMorphoMarketV1RegistryV2.sol`
- `src/interfaces/IMorphoVaultV1Registry.sol`
- `src/interfaces/IRegistryList.sol`

The work was undertaken from **December 9, 2025,** to **December 15, 2025**. During this time, Certora's security researchers performed a manual audit of all the Solidity contracts and discovered several bugs in the codebase, which are summarized in the subsequent section.

Protocol Overview

Morpho Vault V2 integrates with Morpho Blue markets through adapter contracts that track market balances, compute real asset exposure, and enforce vault-level constraints such as caps, share pricing, and interest accrual. In this iteration, the protocol introduced targeted improvements intended to strengthen safety and reduce operational friction.

A key addition is a clean mechanism to force-remove a market when it becomes unhealthy - for example, when a hardcoded oracle depegs and liquidations stop. To accommodate this change, burnShares functions have been introduced.

The update also enhances shares accounting and share-price checks, ensuring markets cannot be exploited through donations or artificial balance inflation.

Finally, the introduction of AdaptiveCurveIrmLib delivers a meaningful gas optimization by enabling interest-rate calculations using only the marketId. This reduces the cost of realAssets() and similar functions without altering underlying logic.

This audit focuses solely on reviewing these incremental changes and their security implications.

Threat and Security Overview

This review focused on the incremental changes introduced to improve the adapter's handling of unhealthy markets and to strengthen share-accounting correctness. The main new feature is the timelocked **burnShares** function, which lets the curator force-remove a market when its oracle stops giving correct prices. We assume that zeroing out the market's shares is intended to realize the loss upfront so the vault does not continue accumulating bad debt. For this review, we assume the protocol roles (**Owner**, **Curator**, **Allocator**, **Sentinel**) are trusted to act honestly and are not fully malicious.

Because **burnShares** clears a market's share balance, we evaluated whether this could be abused to affect other vault operations or markets. We checked how the state updates flow through the adapter and vault, and verified that the removal behaves as expected once the corresponding deallocation happens. No unintended interactions were found.

The update also adds adapter-level share accounting to prevent cases where users donate shares directly to Adapter in Morpho Blue markets in an attempt to distort the vault's asset calculations. We reviewed the **share/asset** conversions and the **allocation/deallocation** updates to ensure these paths cannot be manipulated.

The test coverage for the new logic is solid and exercises the main scenarios, including market removal, share accounting, and the new interest-rate helper library.

Across the reviewed changes, the highest-impact behaviors involve users being able to anticipate **burnShares** executions and shift losses to others, as well as the owner losing direct control over **skimRecipient**. We also noted several lower-severity issues and improvements to make the system more robust.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	2	2	-
Informational	4	4	-
Total	6	6	-

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Detailed Findings

ID	Title	Severity	Status
L-01	Owner Can No Longer Update skimRecipient	Low	Acknowledged
L-02	Users Can Sandwich Scheduled burnShares to Shift Losses to Others	Low	Acknowledged
I-01	Stale Allocation Values Block Legitimate Allocations After burnShares	Informational	Acknowledged
I-02	Abdicating burnShares Selector Can Permanently Block Removal of Markets	Informational	Acknowledged
I-03	Missing event emission in updateList when modifying marketIds	Informational	Acknowledged
I-04	Suggestions	Informational	Acknowledged

Low Severity Issues

L-01. Owner Can No Longer Update skimRecipient		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: MorphoMarketV1AdapterV2.sol	Status: Acknowledged	

Description:

The updated adapter version changes the authority responsible for updating skimRecipient. Previously, the setSkimRecipient function in MorphoMarketV1Adapter was callable directly by the vault owner, allowing immediate updates:

```
JavaScript
    function setSkimRecipient(address newSkimRecipient) external {
        require(msg.sender == IVaultV2(parentVault).owner(), NotAuthorized());
        skimRecipient = newSkimRecipient;
        emit SetSkimRecipient(newSkimRecipient);
   }
```

In MorphoMarketV1AdapterV2, the function is now gated by the adapter's timelock mechanism:

```
JavaScript
  function setSkimRecipient(address newSkimRecipient) external {
        timelocked();
        skimRecipient = newSkimRecipient;
        emit SetSkimRecipient(newSkimRecipient);
   }
```

As a result, control over updating skimRecipient is transferred from the owner to the curator, and the owner can no longer modify this parameter directly.

Recommendations:

Consider updating the new implementation to allow the owner to set the skimRecipient as in the previous version.

Customer's response: Yes this is an intended change, see this comment, thus we acknowledge this issue. The rationale is that there are other critical settings that can be managed by the curator under timelock. The purpose of the owner is mostly to "own" the vault, by being able to change the curator and manage it otherwise. Previously, the owner was authorized to do this change, but it makes more sense to use the same mechanism as in the vault itself.

L-02. Users Can Sandwich Scheduled burnShares to Shift Losses to Others

Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: MorphoMarketV1AdapterV2.sol	Status: Acknowledged	

Description:

When burnShares executes, it sets supplyShares[marketId]=0, causing the adapter's realAssets() to decrease(since expectedSupplyAssets(marketId) returns 0 when supplyShares[marketId] == 0). This reduces the vault's total assets when accrueInterest() is called, lowering the share price.

Users can exploit this behavior by exiting before the share price reduction and re-entering later at a lower price. Although frontrunning is theoretically possible even without a timelock (e.g., via mempool monitoring), placing burnShares behind a timelock significantly amplifies the issue by giving users time to react and creating unfair loss distribution.

Attack Scenario:

- Curator submits burnShares(marketId) via submit(), creating a visible pending operation
- Sophisticated users detect the pending burnShares operation
- Before the timelock expires, these users withdraw their shares at the current (higher) share price
- After execution, burnShares() reduces the vault's total assets and share price
- Remaining users bear the full loss, while frontrunners will re-enter at the lower price

Recommendations:

Given the current design, there is no straightforward mitigation that preserves both functionality and fairness. However, to reduce the attack surface, consider allowing burnShares to be executed immediately rather than through the timelock. This removes the advance visibility window that enables users to anticipate and avoid their share of losses.

Customer's response: Acknowledged. We cannot block forceDeallocate because a malicious curator could be submitting something bad in the meanwhile, effectively being able to rug everybody (and the

non-custodial aspect would be lost). We will document this fundamental limitation though.

Fix Review: Documentation has been added here.

Informational Issues

I-01. Stale Allocation Values Block Legitimate Allocations After burnShares

Description:

The `burnShares` operation is initiated by the curator, whereas allocation management is performed by the allocator. When `burnShares` is executed, it clears `supplyShares[marketId]` but **does not update the vault's caps**. The cap state is only corrected once `deallocate(0)` or `forceDeallocate(0)` is called.

During the small gap between `burnShares` execution and the subsequent cap-syncing call, the vault continues to reflect **stale allocation values**. This creates inconsistent state for collateral-level caps shared across multiple markets for a brief period.

If `burnShares` is applied to Market A, but the cap is not yet updated, the allocator may be prevented from allocating to Market B because the cap still includes Market A's pre-burn allocation, even though those shares are no longer active. This leads to unnecessary allocation failures and operational friction.

```JavaScript
    function burnShares(bytes32 marketId) external {
        timelocked();
        uint256 supplySharesBefore = supplyShares[marketId];
        supplyShares[marketId] = 0;
        emit BurnShares(marketId, supplySharesBefore);
    }
```

Recommendations: Although the code comment notes that deallocate should be called after `burnShares`, we want to highlight the potential risks that can arise in the interim - especially given that these operations are controlled by different roles and could lead to a temporary period of stale cap data.

Customer's response: Acknowledged. Because the allocation stored in the vault contract is always the one computed at the last interaction, it can lag behind the "true assets of the adapter" (what it would be if it was updated now). This design should only prevent some allocation, so it is conservative, that is why we believe this issue to only be a liveness issue and not a security issue: allocators have to do an interaction to be able to allocate even more than what they could before burnShares was executed.

I-02. Abdicating burnShares Selector Can Permanently Block Removal of Markets

Description:

With the introduction of a timelock system in the adapter, the curator is able to abdicate individual selectors. If the curator mistakenly or intentionally abdicates the `burnShares` selector, the system loses the ability to execute burnShares entirely.

Since burnShares is the mechanism used to clear supply shares for markets that have accumulated bad debt, abdication would make it impossible to remove such markets, effectively locking them into the vault with no easy recovery path.

Recommendations: Consider adding an explicit check in the abdication logic to prevent removing the `burnShares` selector.

Customer's response: Acknowledged. Other core functionalities of the vault can already be abdicated (a similar one would be removeAdapter), so it makes sense to be consistent in this case.

I-03. Missing event emission in updateList when modifying `marketIds`

Description:

The updateList function adds or removes marketIds from the marketIds array when allocations change (to/from zero). However, it does not emit any events to signal these list modifications. This makes it difficult for off-chain systems to track the set of active markets efficiently without processing all allocation events.

Recommendations: Define and emit events (e.g., MarketIdAdded(bytes32 marketId) and MarketIdRemoved(bytes32 marketId)) inside updateList whenever marketIds is updated.

Customer's response: Acknowledged. The list can be retrieved by indexing the newly added Allocate and Deallocate events in the adapter. A market should belong to the list exactly when newAllocation is different from 0.

I-04. Suggestions

1. **Improve Error Clarity in createMorphoMarketV1AdapterV2:**
   - The createMorphoMarketV1AdapterV2() function is public and can be called by anyone. If an adapter already exists for a parentVault, a second call will revert due to CREATE2 address collision, but the error message would be unclear. If someone frontruns the protocol team's transaction, the team's call will fail with an opaque CREATE2 revert, making it difficult to understand that an adapter already exists.
   - **Recommendations:** Consider adding an explicit check such as:

```javascript
require(morphoMarketV1AdapterV2[parentVault] == address(0), "AdapterAlreadyExists");
```

2. **Make Authorization Error Messages Consistent:**
   - There is an inconsistency in authorization errors across adapters:
     - MorphoMarketV1AdapterV2 uses `Unauthorized`
     - MorphoVaultV1Adapter still uses `NotAuthorized`
   - **Recommendations:** Consider standardizing on a single error name to keep the codebase consistent

3. **burnShares can "burn" zero shares and emit a misleading event**
   - burnShares(bytes32 marketId) sets supplyShares[marketId] to 0 and emits BurnShares(marketId, supplySharesBefore) even when supplySharesBefore is already 0. This allows "burn" events to be emitted for markets where no shares existed, creating misleading onchain events.

Customer's response: Acknowledged.

1. Acknowledged.

2. Vault v2 and the adapter for vault v1 are already deployed with this inconsistency, so it seems like there is no way to fix it in the adapter for market v1.

3. This event allows to compute the difference of shares (like in other functions), which makes it easy to index the supply shares accounted in the adapter. If the shares were 0 before, it will still give a difference of 0 which is correct, and would not lead to wrong indexing if done properly.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.