Blackthorn

Security Review For **Morpho**

Collaborative Audit Prepared For:	**Morpho**
Lead Security Expert(s):	**hyh**
	pkqs90
	vinica_boy
	xiaoming90
Date Audited:	**November 24 - November 27, 2025**

Introduction

Morpho Vault v2 enables anyone to create non-custodial vaults that allocate assets to any protocols, including but not limited to Morpho Market v1, Morpho Market v2, and Morpho Vault v1. Depositors of Morpho Vault v2 earn from the underlying protocols without having to actively manage the risk of their position. Management of deposited assets is the responsibility of a set of different roles (owner, curator and allocators). The active management of invested positions involves enabling and allocating liquidity to protocols.

Morpho Vault v2 is ERC-4626 and ERC-2612 compliant. The VaultV2Factory deploys instances of Vaults v2. All the contracts are immutable.

Scope

Repository: morpho-org/morpho-blue-irm

Audited Commit: 2bce6da3ebe910ceeec963781aa3f4b4322c296e

Final Commit: 2bce6da3ebe910ceeec963781aa3f4b4322c296e

Files:

- src/adaptive-curve-irm/libraries/periphery/AdaptiveCurveIrmLib.sol

---

Repository: morpho-org/vault-v2

Audited Commit: 169044ef3066931b98a7ff32deca29ed30ecae67

Final Commit: 425f6b1fb3a1d25083a9515348bd5e7bd3ab9017

Files:

- src/adapters/interfaces/IMorphoMarketV1AdapterFactory.sol
- src/adapters/interfaces/IMorphoMarketV1Adapter.sol
- src/adapters/MorphoMarketV1AdapterFactory.sol
- src/adapters/MorphoMarketV1Adapter.sol

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	8

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@hyh

hyh

@pkqs90

[illegible]

@vinica_boy

Vinica Boy

@xiaoming90

[illegible]

Issue L-1: Burning shares cannot be abdicated [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/6

Vulnerability Detail

The `VaultV2` contract implements an abdication mechanism where the curator can permanently disable specific timelocked functions by calling `abdicate()`. Once a function selector is abdicated, it cannot be executed regardless of its timelock status.

The `MorphoMarketV1Adapter` implements a `submitBurnShares()` function that allows the curator to burn the adapter's shares for a specific market. This function uses the same timelock duration as `IVaultV2.removeAdapter.selector`:

```
function submitBurnShares(bytes32 marketId) external {
    require(msg.sender == IVaultV2(parentVault).curator(), NotAuthorized());
    require(burnSharesExecutableAt[marketId] == 0, AlreadyPending());
    burnSharesExecutableAt[marketId] =
        block.timestamp +
         ↪  IVaultV2(parentVault).timelock(IVaultV2.removeAdapter.selector);
    emit SubmitBurnShares(marketId, burnSharesExecutableAt[marketId]);
}
```

If the curator abdicates `removeAdapter` in `VaultV2` to permanently prevent adapter removal, the `burnShares()` function in the adapter still remains callable and depends on timelock of `IVaultV2.removeAdapter.selector`. This creates an inconsistency in this where removing an adapter is disabled while burning `MorphoMarketV1Adapter` shares is not.

Also, the following comment in `VaultV2` should be updated to clarify the special case for `IVaultV2.removeAdapter.selector`.

```
/// ABDICATION
/// @dev When a timelocked function is abdicated, it can't be called anymore.
/// @dev It is still possible to submit data for it or change its timelock, but it
↪  will not be executable / effective.
```

Impact

The abdication of `removeAdapter` does not prevent the adapter from burning its shares.

Recommendation

If this is intended, best to add documentation. If not, also add an abdicate feature for burning shares in `MorphoMarketV1Adapter`.

Discussion

MathisGD

this will be fixed with #840

Issue L-2: `submitBurnShares()` can be executed against markets that have not been added yet [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/7

This issue has been acknowledged by the team but won't be fixed at this time.

Vulnerability Detail

It was observed that it is possible for a curator (malicious or not) to preemptively execute `submitBurnShares()` against a `marketId` that has not yet been added to the adaptor, but is planned (though not yet announced) to be added in the future.

Assume the timelock is 7 days. At (T0 – 7 days), the curator can execute the `submitBurnShares()` function against Market B. When Market B is added to the adaptor at T0, its shares can be burned at any time the curator wishes, regardless of the timelock.

Technically, users can discover this by backtracking through the event logs, but users or bots might only track markets that have already been added from T0 onward when the `Allocate(marketId, newAllocation, mintedShares)` event is emitted, so they might not discover the past events.

```
File: MorphoMarketV1Adapter.sol
    function submitBurnShares(bytes32 marketId) external {
        require(msg.sender == IVaultV2(parentVault).curator(), NotAuthorized());
        require(burnSharesExecutableAt[marketId] == 0, AlreadyPending());
        burnSharesExecutableAt[marketId] =
            block.timestamp + IVaultV2(parentVault).timelock(IVaultV2.removeAdapter.selector);
        emit SubmitBurnShares(marketId, burnSharesExecutableAt[marketId]);
    }
```

Impact

Shares of markets can be burned immediately after adding to the adaptor, and some users might not be aware of this, since the `submitBurnShares()` transaction was executed some time before the market was added.

Code Snippet

[https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/blob/1ba174dcd4735ec11b923ba56b8087e4bc514e1e/morpho-org__vault-v2/src/adapters/MorphoMarketV1Adapter.sol#L81https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/blob/1ba174dcd4735ec11b923ba56b8087e4bc514e1e/morpho-org__vault-v2/src/adapters/MorphoMarketV1Adapter.sol#L81](https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/blob/1ba174dcd4735ec11b923ba56b8087e4bc514e1e/morpho-org__vault-v2/src/adapters/MorphoMarketV1Adapter.sol#L81)

Tool Used

Manual Review

Recommendation

Consider allowing the curator to only execute `submitBurnShares()` against a market that has already been added or exists in the adaptor. This can be done by adding an extra validation `require(supplyShares[marketId] > 0)` to ensure that only markets that have been "activated" (where market shares > 0) can execute `submitBurnShares()`.

```
    function submitBurnShares(bytes32 marketId) external {
        require(msg.sender == IVaultV2(parentVault).curator(), NotAuthorized());
        require(burnSharesExecutableAt[marketId] == 0, AlreadyPending());
        burnSharesExecutableAt[marketId] =
            block.timestamp +
             ↪  IVaultV2(parentVault).timelock(IVaultV2.removeAdapter.selector);
+       require(supplyShares[marketId] > 0, MarketDoesNotExist);
        emit SubmitBurnShares(marketId, burnSharesExecutableAt[marketId]);
    }
```

Discussion

MathisGD

We acknowledge this issue. This is something fundamental with timelocks in Vaults V2: to be fully non custodial, one must monitor all submissions (including past ones).

Issue L-3: Front-running of share burn [RE-SOLVED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/8

Vulnerability Detail

Assume that the `MorphoMarketV1Adapter` adaptor has two (2) markets:

- Market A - 50 USDC supplied
- Market B - 50 USDC supplied
- Vault's Total Asset = Market A's assets + Market B's assets = 100 USDC
- Vault's Total Share/Supply = 100 shares
- Price Per Share (PPS) = 1.0 (100 USDC/100 shares)
- Alice holds 50 vault shares, while Bob holds 50 vault shares

Assume that the collateral price in Market A depegged, and there is no chance of recovering the assets supplied to Market A. In this case, the positions in Market A will be abandoned.

It was observed that the timelock of the burn share feature relies on the timelock of the parent vault's remove adaptor selector in Line 85. Assume that the remove adaptor selector in the parent vault is subjected to a 7-day timelock.

```
File: MorphoMarketV1Adapter.sol
81:     function submitBurnShares(bytes32 marketId) external {
82:         require(msg.sender == IVaultV2(parentVault).curator(), NotAuthorized());
83:         require(burnSharesExecutableAt[marketId] == 0, AlreadyPending());
84:         burnSharesExecutableAt[marketId] =
85:             block.timestamp +
 ↪ IVaultV2(parentVault).timelock(IVaultV2.removeAdapter.selector);
86:         emit SubmitBurnShares(marketId, burnSharesExecutableAt[marketId]);
87:     }
```

In this case, there will be a 7-day waiting period before Market A's position is written off from the vault.

Bob observed the pending request to write off the losses in the timelock queue and attempted to withdraw from the vault as soon as possible. If needed, Bob will execute `forceDeallocate()` to deallocate the assets from Market B, and pay a small penalty. Since the PPS is still 1.0, Bob will receive around 50 USDC after he withdraws all his shares.

Alice will end up incurring all the losses from Market A and be unable to withdraw any assets. As a result, users who exit later will incur most, if not all, of the losses.

Impact

Users who exit later will incur most, if not all, of the losses.

Code Snippet

https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/blob/1ba174dcd4735ec11b923ba56b8087e4bc514e1e/morpho-org__vault-v2/src/adapters/MorphoMarketV1Adapter.sol#L81

Tool Used

Manual Review

Recommendation

This impact is similar to an existing risk in which a vault user can front-run the socialization of a position's bad debt in a market when that position is liquidated. Nevertheless, this impact should be documented so that users are aware of the similar effect that also occurs when a market's shares are written off.

As a partial control, consider forbidding the execution of `forceDeallocate()` for any market when burning shares is in progress, which is a reasonably rare action

In other words, an adapter wide flag (e.g., `burning = true`, without an id, just burning anything) can be set on `submitBurnShares()`, and then removed on `revokeBurnShares()`

and `burnShares()`. When `deallocate()` comes from `forceDeallocate()`, the function should check if burning shares is in progress (e.g., `require(!forcing || !burning)`).

Discussion

MathisGD

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.

Issue L-4: Shares can be directly burned from Vault's liquidity market [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/9

Summary

While Vault v2 relies on the `liquidityAdapter` and `liquidityData` defined market, this market can now be burned directly, blocking all the Vault's withdrawals.

Vulnerability Detail

It's possible to run `submitBurnShares()` -> `burnShares()` for the current liquidity market of the underlying Vault.

Impact

Vault's users will have to use `forceDeallocate()` to be able to perform any withdrawal from the Vault, incurring the corresponding `forceDeallocatePenalty`.

Recommendation

Consider blocking `burnShares()` when Vault's `liquidityAdapter` is `address(this)` and `Id.unwrap((abi.decode(liquidityData, (MarketParams))).id())` is `marketId`, implying that Vault's `setLiquidityAdapterAndData|()` needs to be run first.

Discussion

dmitriia

Suggestion to set market allocation to zero on the Vault level, which essentially removes the possibility to use liquidity market being burned and implies that some other market should be chosen for this, added to burn shares functionality description in https://github.com/morpho-org/vault-v2/pull/841

Issue L-5: It's possible to allocate to a market having shares burning in progress [RESOLVED]

Source:
https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/10

Summary

Vault's `allocate()` can be run by any allocator, while shares burning can be initiated by Vault's curator. The sychronization between these actors can be not perfect, while the mistake of allocating to a broken market is costly.

Vulnerability Detail

It's possible to successfully run `allocate()` for a market for which `submitBurnShares()` was run and `burnSharesExecutableAt[marketId] > 0`:

MorphoMarketV1Adapter.sol#L111-L116

```
function allocate(bytes memory data, uint256 assets, bytes4, address) external
 ↪  returns (bytes32[] memory, int256) {
    MarketParams memory marketParams = abi.decode(data, (MarketParams));
    require(msg.sender == parentVault, NotAuthorized());
    require(marketParams.loanToken == asset, LoanAssetMismatch());
    require(marketParams.irm == adaptiveCurveIrm, IrmMismatch());
    bytes32 marketId = Id.unwrap(marketParams.id());
```

Executing `burnShares()` for this market will write off this investment.

Impact

Even without an adapter level write off (e.g. if `revokeBurnShares()` will be then run instead of `burnShares()`) the funds allocated to a broken market, e.g. one with an collateral price misreporting Oracle covering bad debt, are likely to be lost.

I.e. `submitBurnShares()` is first of all a broken market signal, so when it was run the subsequent allocation is most likely either an operational mistake or an allocator run exploit. In both cases these funds can be lost whenever the corresponding market is indeed broken.

On the other hand, even if the market is in fact sound, and if `revokeBurnShares()` won't be run until timelock expires, then the permissionless `burnShares()` can permanently remove the funds from depositor's reach.

Recommendation

Consider forbidding the allocations when burning shares is in progress, e.g. by requiring `burnSharesExecutableAt[marketId] == 0` here.

Discussion

dmitriia

Suggestion to set market allocation to zero on the Vault level, which is an alternative way to fix the issue, added to burn shares functionality description in https://github.com/morpho-org/vault-v2/pull/841

Issue L-6: Dust sized withdrawals can be used to make Vault lose value to the underlying market depositors [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/11

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Upward rounding for required share amounts used both in Vault and underlying market withdrawals can lead to losing value by the Vault if underlying market shares are costly enough compared to Vault's.

Vulnerability Detail

Suppose a market A the Vault invested in has expensive shares compared to Vault itself, e.g. via having consistently high APY and sound collateral price feed, while the Vault is invested in lagging markets and had to burn shares for some of them. The attacker can be the large direct depositor in market A. They can deposit in Vault and then force deallocate the dust sized amount from market A many times over, paying that rounding up in the terms of Vault's shares, while Vault will be paying the similar rounding up, but in the terms of market's shares.

In a cheap gas environment the net economic effect can be close to the number of repetitions multiplied by the difference of shares valuation, i.e. market share price minus Vault share price, also multiplied by attacker's share in the market, which can be substantial. If share price difference be high enough then the attack can be profitable. I.e. Vault repeatedly losing value due to rounding produces gain for all the other market's depositors and for the biggest ones that can be higher than the cost of force deallocation from the Vault.

Impact

Vault can more lose value from a substantial amount of minimum number of shares burned on exit from the market A than attacker lost on the same amount of dust sized exits from the Vault. Then the attacker and other market A depositors gained, and Vault lost the shares valuation difference determined value. I.e. market A depositors stole from Vault depositors.

Recommendation

It looks like there is no valid use case for dust sized deallocations, i.e. the zero amount case can be used for market list update after shares burning and for removing defaulted markets, the allocator would use the substantial amounts only for deallocations across Vault lifecycle, and the depositors would force deallocate amounts corresponding to their withdrawal needs.

As dust sized deallocations look to be having no valid use cases consider introducing the minimum amount for deallocation on the adapter level.

Discussion

MathisGD

I don't think that preventing dust sized reallocations would fix anything. You could still allocate and deallocate big amounts, creating a rounding loss at each interaction.

We think that in practice this won't be an issue, especially with the addition of the share price check on allocate. Note that it's documented to use only Morpho markets that are protected against share price inflation anyway.

dmitriia

Users can deallocate from a specific market of Vault v2 using force deallocation only, which cost is proportional to the amount used. So controlling for a minimum amount provides a lower boundary for this cost, which significantly worsens the economics of such an attack.

The share price check is useful, but controls for the most extreme cases only.

What I see as the most direct vector here is a big number of dust sized force deallocations materially reducing Vault's share in the market. I.e. attacker deposits in the underlying market for the majority share of it, then deposits in the Vault, then burns the latter deposit completely via all these deallocations, gaining more via rising valuation of the former deposit, withdraws it.

Issue L-7: Vault's `forceDeallocate()` can be used to prevent allocations to other markets [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/12

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

As there is no throttling for the allocation updates, running them for a market with a broken oracle can be used as a limit utilizing griefing that prevents allocation to other markets until faulty market's shares be burned in the adapter.

Vulnerability Detail

Calling Vault's `forceDeallocate()` with zero amount for the faulty market periodically can fill the `absoluteCap` and `relativeCap` limits, preventing allocations with the same adapter or collateral token.

Relative limit is the most vulnerable in this regard as Vault's total assets' growth is limited by `maxRate`, while `expectedSupplyAssets(marketId)` isn't.

Impact

Allocations to other markets can be limited or denied for a `removeAdapter` timelock period used in `submitBurnShares()`.

Recommendation

Zero amount deallocations can be forbidden when `burnSharesExecutableAt[marketId] > 0`, since those are updates for a market that was marked for removal.

Discussion

MathisGD

this is a temporal grieffing, as it is reset once the market gets removed. we ack

Issue L-8: IMorphoMarketV1Adapter's market id naming isn't unified across signatures [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-morpho-adapter-nov-24th/issues/13

Summary

Market id is now named `marketId` in `Allocate` and `Deallocate`, `supplyShares` and `expectedSupplyAssets`, and named `id` elsewhere.

Recommendation

Consider unifying the interface signatures.

Discussion

dmitriia

Fixed in https://github.com/morpho-org/vault-v2/pull/841

Disclaimers

Blackthorn does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.