Blackthorn

Security Review For **Morpho**

Collaborative Audit Prepared For:	**Morpho**
Lead Security Expert(s):	**0x73696d616f**
	hyh
	pkqs90
	xiaoming90
Date Audited:	**August 13 - August 20, 2025**
Final Commit:	**6f2af66**

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: sherlock-scoping/morpho-org__vault-v2

Audited Commit: ce661d820fb29307981f75eb42393db1c6e42758

Final Commit: 6f2af6602e05d9e123a87c1067712a4566608044

Files:

- src/adapters/interfaces/IMorphoMarketV1AdapterFactory.sol
- src/adapters/interfaces/IMorphoMarketV1Adapter.sol
- src/adapters/interfaces/IMorphoVaultV1AdapterFactory.sol
- src/adapters/interfaces/IMorphoVaultV1Adapter.sol
- src/adapters/MorphoMarketV1AdapterFactory.sol
- src/adapters/MorphoMarketV1Adapter.sol
- src/adapters/MorphoVaultV1AdapterFactory.sol
- src/adapters/MorphoVaultV1Adapter.sol
- src/imports/MetaMorphoImport.sol
- src/imports/MetaMorphoV1_1Import.sol
- src/imports/MorphoImport.sol

- src/interfaces/IAdapter.sol
- src/interfaces/IERC20.sol
- src/interfaces/IERC2612.sol
- src/interfaces/IERC4626.sol
- src/interfaces/IGate.sol
- src/interfaces/IVaultV2Factory.sol
- src/interfaces/IVaultV2.sol
- src/libraries/ConstantsLib.sol
- src/libraries/ErrorsLib.sol
- src/libraries/EventsLib.sol
- src/libraries/MathLib.sol
- src/libraries/SafeERC20Lib.sol
- src/VaultV2Factory.sol
- src/VaultV2.sol

Final Commit Hash

6f2af6602e05d9e123a87c1067712a4566608044

Findings

Each issue has an assigned severity:

- 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.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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
1	5	18

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Security Experts Dedicated to This Review

@0x73696d616f

@hyh

@pkqs90

@xiaoming90

Issue H-1: Fixed total assets within a transaction allow for avoiding underlying market bad debt loss

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/45

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

Summary

Interest accrual happens only once per transaction, which makes it possible to avoid underlying market loss realization.

Vulnerability Detail

Suppose there is a Vault with only one adapter that have only one underlying Morpho market enabled that was invested before from the Vault, but now have no available liquidity and a big liquidable bad debt position, so it's impossible to exit this market without liquidation and loss realization, and say there is Bob the Vault depositor who wants to exit the Vault, and also there are other material depositors there.

Bob can atomically run 3 operations:

1) first dust sized Vault entry/exit to pinpoint `firstTotalAssets = _totalAssets`,
2) underlying market liquidation, realizing bad debt,
3) full exit from the Vault, which just became possible since liquidation provided the available funds.

Bob's exit will be done at a before liquidation share price, since adapter's assets will not be recalculated inside the transaction after being computed in (1).

This will happen at the expense of the remaining depositors, who will bear the increased loss right afterwards: for them it will be the original bad debt loss and the impact of Bob's lossless exit combined.

Impact

All other depositors will pay for Bob fully avoiding the bad debt induced loss in the underlying market.

Code Snippet

`_totalAssets` are calculated once in the beginning of a transaction, then being adjusted by entries and exits only:

VaultV2.sol#L579-L585

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
>>      if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
        uint256 elapsed = block.timestamp - lastUpdate;
        uint256 realAssets = IERC20(asset).balanceOf(address(this));
        for (uint256 i = 0; i < adapters.length; i++) {
            realAssets += IAdapter(adapters[i]).realAssets();
        }
```

This way it's possible to batch a loss making action into a transaction after any not material interest accruing Vault action, making Vault ignore the loss for any subsequent atomic interactions.

Recommendation

While transient nature of `firstTotalAssets` fits flash loan control purpose, it looks like there are not many reasons besides gas optimization to ignore total assets update within the transaction as adapter's `realAssets()` change might not be linked to the passage of time. In other words, total assets shouldn't be transaction persistent.

Consider recalculating the total assets, e.g.:

VaultV2.sol#L579-L588

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
-       if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
        uint256 elapsed = block.timestamp - lastUpdate;
        uint256 realAssets = IERC20(asset).balanceOf(address(this));
```

```
        for (uint256 i = 0; i < adapters.length; i++) {
            realAssets += IAdapter(adapters[i]).realAssets();
        }
        uint256 maxTotalAssets = _totalAssets + (_totalAssets *
        ↪  elapsed).mulDivDown(maxRate, WAD);
        uint256 newTotalAssets = MathLib.min(realAssets, maxTotalAssets);
+       if (elapsed == 0) return (newTotalAssets, 0, 0);
        uint256 interest = newTotalAssets.zeroFloorSub(_totalAssets);
```

Discussion

MathisGD

Nice catch. We didn't have in mind that this has a real impact when the market is illiquid (if the market was liquid, bob could exit anyway at the initial share price by exiting before the liquidation). We think though that it's very edge case (the problem of liquidation front-running is still here in the general case).

Note that we can't do the fix you are proposing, because it would allow to attack the vault by shorting shares if they are flashloanable (flashloan shares, withdraw, realize, deposit, repay flashloan). It would require to re-add the enterBlocked flag that we had before.

dmitriia

Additionally prohibiting Vault's entry and exit in the same transaction with `transient enterBlocked` after a down tick was recorded in total assets looks like a good idea.

Issue M-1: `VaultV2::withdraw/redeem()` are vulnerable to slippage, so another function could be added to protect users

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/29

Summary

As per the ERC4626 spec, VaultV2::withdraw/redeem() don't receive a slippage parameter, which makes them vulnerable to slippage. This is relevant as the underlying vaults can register losses, and could surprise users redeeming who get these losses. Having a function or router that includes slippage protection could be useful for users to protect them.

Vulnerability Detail

The underlying adapters can register losses, which would be socialized among users of the VaultV2. In case one of them decides to withdraw or redeem and be frontrunned by one of these losses, they would receive a surprisingly lower amount of assets, incurring losses.

Impact

User loss on withdrawal.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R701-R714

Tool Used

Manual Review

Recommendation

Add a function or router that includes slippage protection.

Discussion

MathisGD

Proposed a simple comment. Is it really medium though? If the vault had some losses, the users lost, independently of whether they revert on withdraw or not

0xsimao

Slippage protection is usually medium/high, though I see your point. It is indeed more severe when the amount withdrawn can be sandwiched such as in exchanges, which doesn't seem to be the case here. I think an important consideration is that users may want/expected at least X assets, and they would get less, so in theory they would need to redeposit if they wanted to get their desired assets, which wouldn't be needed if it reverted instead. Hence, the loss is realized either way, but their desired behavior may be different depending on the value of the withdrawal, and could force them to redeposit, which may incur fees and/or make them miss out on interest.

MathisGD

alright, you have the final decision

Issue M-2: Assets could be allocated to an adaptor that is about to be disabled

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/33

Vulnerability Detail

Assume a vault with two adaptors:

- Adaptor A has 500 assets
- Adaptor B has 500 assets
- Vault's Idle asset = 0
- Total supply = 1000 shares, Price = 1.0

At some point, curator decided to decommission Adaptor B, and thus Allocator deallocated all funds from Adaptor B, and the state of the vault is as follows:

- Adaptor A has 500 assets
- Adaptor B has 0 assets
- Vault's Idle asset = 500
- Total supply = 1000 shares, Price = 1.0 (1000/1000)

Curator submits `setIsAdapter` payload to disable Adaptor B, which has to go through a 2-week time lock. During this period, users saw that the funds in the Adaptor B had been deallocated and assumed that it could be removed safely.

The `setIsAdapter` payload can be executed by anyone (e.g., Curator, Allocator or even public users) after the 2-week timelock.

2 weeks have passed. However, right before the Allocator executes the `setIsAdapter` payload to disable Adaptor B, Allocator allocates 500 assets from Idle assets to Adaptor B. Allocator proceeds to disable Adaptor B. At this point, the only supported adaptor is

Adaptor A. Thus, when calculating the total assets, it will only take into account of idle asset + Adaptor A, but not Adaptor B.

The state of the vault is as follows:

- Adaptor A has 500 assets
- Adaptor B has 500 assets (No longer supported)
- Vault's Idle asset = 0
- Total supply = 1000 shares, Price = 0.5 (500/1000)

Thus, users suffer a loss in this case.

Impact

The vault share price will decrease.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/c108b078de7a89b226ccdff50f65ca70433e473c/VaultV2.sol#L348

Tool Used

Manual Review

Recommendation

The cap of the adaptor to be disabled has to be set to zero to prevent any ”last-minute” allocation by the allocator.

Consider documenting the need for this action or programming it into the codebase to avoid this issue.

Discussion

xiaoming9090

This potential scenario has been added to the codebase's comment for awareness.

Issue M-3: Some deposits might be blocked when the relative cap is enabled

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/34

Vulnerability Detail

Assume that the vault state is as follows:

- `absoluteCap` is set to a sufficiently large value (e.g., 10M USDC)
- `relativeCap` is set to 10% for the `MorphoVaultV1Adapter`
- `caps.allocation` is zero
- Vault's idle total asset is zero

It was understood that some vault curators might consider using `MorphoVaultV1Adapter` as the liquidity adaptor. In this case, when a user deposits 1000 USDC, if the liquidity adaptor (set to `MorphoVaultV1Adapter`) is configured, the 1000 USDC will be automatically deposited/allocated to the external market. Also, when the user calls the `deposit()` function, the `accrueInterest()` function will be executed at the start of the transaction and will lock the `firstTotalAssets` to zero.

However, the condition in the `require` statement will fail and revert, blocking the initial deposit.

```
_caps.allocation <= firstTotalAssets.mulDivDown(_caps.relativeCap, WAD)
1000 <= (0 * 10%)
1000 <= 0
False // Revert
```

Even outside of the first deposit, it might still cause some revert.

Assume that the vault state is as follows:

- `absoluteCap` is set to a sufficiently large value (e.g., 10M USDC)

- `relativeCap` is set to 10% for the `MorphoVaultV1Adapter`
- `caps.allocation` is zero
- Vault's idle total asset is 10,000 USDC

If a user deposits 1000 USDC or less, the deposit will succeed. However, if the user deposits more than 1000 USDC, the deposit will revert. So, the amount of deposit a user can make in each deposit is restricted or capped.

If a user deposits 1000 USDC, the vault's total assets will be 11,000 USDC, and 10% (relative cap) of this amount will be 1100 USDC. However, since the `firstTotalAssets` is locked at the total assets before the deposit (10,000 USDC), the maximum allowable deposit is 1000 instead of 1100.

This seems to be a trade-off to mitigate the risk of an allocator using flash-loan to bypass the relative cap. This design will work reasonably well as long as the vault's total assets before and after the deposit do not deviate significantly because the relative cap based on `firstTotalAssets` should give a good enough approximation to limit the risk of over-allocation.

However, if the deviation is significant, it will block the deposit. I'm not sure if this is intended, but it seems unusual that the relative cap ends up indirectly limiting the maximum deposit size. Especially, if the total assets of the vaults are small during the initial stage, the deposit size will be very limited, and the deposit size has to ”grow” progressively with the total assets.

Impact

Some deposits might be blocked when the relative cap is enabled.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/c108b078de7a89b226ccdff50f65ca70433e473c/VaultV2.sol#L522

Tool Used

Manual Review

Recommendation

The relative cap should not apply when the total asset value is zero, since there is nothing to benchmark against to determine the deviation.

In addition, consider further documenting that configuring a relative cap may indirectly restrict the maximum amount allowed in a single deposit

Discussion

MathisGD

there is this comment already:

```
/// @dev If a cap (absolute or relative) associated with the ids returned by the
 ↪  liquidity adapter on the liquidity data
/// is reached, deposit/mint will revert.
```

don't you think that it's is enough? should we explicitly recommend using a market with no relative cap as a liquidity market?

xiaoming9090

@MathisGD The below comment should cover the scenario I mentioned in the second half of the report.

```
/// @dev If a cap (absolute or relative) associated with the ids returned by the
 ↪  liquidity adapter on the liquidity data
/// is reached, deposit/mint will revert.
```

However, I don't think the above comment covers the scenario I mentioned in the first half of the report. It doesn't seem obvious to me that the users should not enable `relativeCap` at the start if their vault uses a liquidity adapter to automatically allocate liquidity because the `_caps.allocation <= firstTotalAssets.mulDivDown(_caps.relativeCap, WAD)` check will revert during deposit.

It would be good to expand the documentation on this.

MathisGD

fair, we could add a remark on this too

xiaoming9090

Acknowledged by updating comments.

Issue M-4: Honest users can lose funds if adapters had deposit fees

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/36

Summary

Due to the fast-return logic that the total assets are fixed in the beginning of the transaction, there is a scenario where honest users can lose funds if adapters had deposit fees.

Vulnerability Detail

Consider the following scenario:

1. Adapter has a deposit fee, so whenever we allocate idle assets to the adapter, the total realAssets() decrease.
2. Vault-v2 has a single adapter that collects 1% deposit fee, initially it has 10000 assets allocated all to the adapter. The adapter is also set as liquidity adapter.
3. Attacker calls `deposit()` for 10000. `_totalAssets` is now 20000, because the `enter()` function adds 10000 to `_totalAssets` directly.
4. Liquidity adapter automatically deposits the 10000 to the adapter, but since it collects 1% deposit fee, the real totalAsset should be 19900, while the fast-return logic still returns 20000.
5. Attacker withdraws his shares, and he still receives 10000, because it uses fast-returned 20000 for total assets during calculation.
6. The 100 loss is now beared by other users.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/VaultV2.sol#L577

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
```

```
@>      if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
        ...
    }

    function enter(uint256 assets, uint256 shares, address onBehalf) internal {
        ...
@>      _totalAssets += assets.toUint128();
        ...
    }
```

Note the attacker does not lose anything, so he can keep on doing the attack until funds are drained.

Impact

Attacker can drain honest user's funds.

Recommendation

Do not allow vaults with deposit fees, or rethink the fast-return logic.

Discussion

MathisGD

Answer similar to #32. See the associated fix.

pkqs90

Acknowledged. Added a comment adapters with deposit fees should not be used.

Issue M-5: Delayed yield can be stolen

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/43

Summary

Total assets upwards evolution is limited by `maxRate` setting. Whenever it triggers the arbitrage opportunity of receiving close to guaranteed profit arises and it has high probability of exploitation when Vault's total assets are substantial enough.

With `maxRate` level of limitation itself being one source of the delayed profit realization, `forceDeallocate()` usage is an another as it provides instant `penaltyAssets` gain for depositors that becomes delayed at least until next block no matter how big `maxRate` is.

Vulnerability Detail

Max rate truncated profit realization allows for arbitrage by entering the Vault when it's triggered and exiting after that delayed revenue realization period ends. The higher max rate is set the lower the probability of this happening overall, but the bigger the probability of arbitrage actors acting on it. The absence of direct entry and exit fees also enhances the profitability.

`forceDeallocate()` provides a routine case of instant Vault level profit for arbitragers to dilute. The room for entry exists in the same block for any `maxRate` as for other transactions in this block `elapsed == 0`. That is, Bob the arbitrager can setup a bot back running `forceDeallocate()` to take part in above market return distribution at the expense of all other Vault depositors.

Impact

Vault depositors can systematically lose a portion of yield. One venue is a part of the yield that exceeds the `maxRate` speed, the other is an instant yield provided by `forceDeallocate()` sourced penalty assets.

First part is conditional on `maxRate` breach, which can be rare if the value is set high enough, and so have medium severity. Second part is unconditional and can be run

routinely, tracking `forceDeallocate()` usage, which is a core user control functionality, i.e. it have to be used in the case of allocator failing to do their job for any reason. Given `forceDeallocate()` is used the penalty assets profit can be stolen from Vault's depositors deterministically, so this surface have high severity.

Code Snippet

`_totalAssets` will not rise faster than `maxRate` and is not allowed to discover any yield in the same block after the update:

VaultV2.sol#L579-L587

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
        if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
>>      uint256 elapsed = block.timestamp - lastUpdate;
        uint256 realAssets = IERC20(asset).balanceOf(address(this));
        for (uint256 i = 0; i < adapters.length; i++) {
            realAssets += IAdapter(adapters[i]).realAssets();
        }
>>      uint256 maxTotalAssets = _totalAssets + (_totalAssets *
↪   elapsed).mulDivDown(maxRate, WAD);
        uint256 newTotalAssets = MathLib.min(realAssets, maxTotalAssets);
```

Recommendation

Given that `maxRate` is set high enough its breaches can be rare and the most probable scenario is `forceDeallocate()` back running. Consider saving the penalty assets and adding them to the total on any next interaction, e.g.:

VaultV2.sol#L182

```
    uint128 public _totalAssets;
+   uint128 public _penaltyAssets;
```

VaultV2.sol#L749-L754

```
    function forceDeallocate(address adapter, bytes memory data, uint256 assets,
    ↪   address onBehalf)
```

```
        external
        returns (uint256)
    {
        bytes32[] memory ids = deallocateInternal(adapter, data, assets);
        uint256 penaltyAssets = assets.mulDivUp(forceDeallocatePenalty[adapter],
        ↪  WAD);
+       _penaltyAssets += penaltyAssets.toUint128();
```

VaultV2.sol#L579-L588

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
        if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
+       if (firstTotalAssets == 0 && _penaltyAssets != 0) {_totalAssets +=
↪  _penaltyAssets; _penaltyAssets = 0;}
        uint256 elapsed = block.timestamp - lastUpdate;
        uint256 realAssets = IERC20(asset).balanceOf(address(this));
        for (uint256 i = 0; i < adapters.length; i++) {
            realAssets += IAdapter(adapters[i]).realAssets();
        }
        uint256 maxTotalAssets = _totalAssets + (_totalAssets *
        ↪  elapsed).mulDivDown(maxRate, WAD);
        uint256 newTotalAssets = MathLib.min(realAssets, maxTotalAssets);
        uint256 interest = newTotalAssets.zeroFloorSub(_totalAssets);
```

This way, on the one hand flash loan based manipulation will still be prohibited, i.e. flash loan fueled big depositor using `forceDeallocate()` will lose funds as penalty assets will not be recognized until next transaction, on the other any next depositor will deal with actualized total assets, so won't be able to enter at a depressed valuation.

The `firstTotalAssets == 0` condition was added to make it compatible with issue 45: while in current transaction there is no need to add penalty assets, otherwise it would be possible to reallocate for free with flash loans, but once it's over the penalty assets have to be added instantly and in full as otherwise this delayed yield is free to be grabbed, so the resulting logic is to add them at the start of the next transaction.

Discussion

MathisGD

Great finding! We didn't have this in mind.

We think that the best thing to do is to document the behavior: donation and penalties can be ”stolen” by malicious depositors if the maxRate is way above the rate. Thus to prevent that, maxRate should be kept not too far from the rate. If people don't plan to do donations, and don't see the attack happening on penalties, they can go optimistic (keep it high) and change after. Allocators have this role, but they can already set the rate at zero (they are trusted for the yield, not for the principal though, which is consistent here).

nb: penalties are here to compensate for the loss of yield due to the forceDeallocate during the time that the allocator didn't rebalance.

MathisGD

about the severity of the issue, this is loss of yield not principal, do you generally consider this as ”high”?

dmitriia

The criteria is more limitations/conditions and materiality based.

This issue is limited to the one scenario within the ordinary workflow, i.e. `forceDeallocate()` can be ordinary used from time to time only, but it requires no specific preconditions. On the other hand, the loss is bounded to a part of the yield.

I think it's borderline, there are valid arguments both for High and Medium. Given that the max possible impact is full cancellation of the loss of yield compensation (deallocator pays, but other depositors don't receive anything), it can be treated as Medium overall.

Issue L-1: `submit()` could be more verbose when a selector has been abdicated

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/26

Summary

`VaultV2::submit()` overflows here when the timelock of the selector has been set to `type(uint256).max`, which works as intended but could be more verbose.

Vulnerability Detail

`VaultV2::abdicateSubmit()` sets the timelock to the maximum, making the `submit()` function overflow to disable function calls with this selector.

```
function abdicateSubmit(bytes4 selector) external {
    timelocked();
    timelock[selector] = type(uint256).max;
    emit EventsLib.AbdicateSubmit(selector);
}
```

Impact

Error handling is not very verbose

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R384

Tool Used

Manual Review

Recommendation

Add an error message.

Issue L-2: Performance and management fee updates may technically still apply to already earned interest

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/27

Summary

Due to the max rate mechanism, interest earned will be capped by the max rate. Thus, even if the past interest is accrued as per the rate before changing the fees, the actual interest earned that will be attributed to the Vault over time will use the new rate, but it corresponds to deposits made with the old rate.

Vulnerability Detail

`VaultV2::setPerformanceFee()` correctly accrues interest first, and only changes the fee afterwards, link. However, note that the interest accrued is capped by the max rate, link. This means that the pending interest earned while the old rate was active will be now applied the new rate.

Impact

Under/over charging fees.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R406

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R584

Tool Used

Manual Review

Recommendation

There are ways to fix this but not quite trivial.

Issue L-3: Performance and management fees could round up to protect the protocol

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/28

Summary

`VaultV2::accrueInterestView()` rounds down when applying the perfomance and management fees, leading to reduce interest payments for the protocol.

Vulnerability Detail

Since the Vault uses a decimal offset to always have 18 decimals, the difference should be negligible, but it is something to keep in mind, especially as gas prices keep decreasing and $WBTC price keeps increasing. For this reason protocols often round up when calculating fees, though it may not be necessary in this case.

Impact

Wei interest loss.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R589-R596

Tool Used

Manual Review

Recommendation

Consider rounding up, though the impact is negligible here.

Discussion

MathisGD

We decided to acknowledge this one. It's non trivial if one would prefer the fees to be rounded up in the case of a high value asset (could mean that users don't accrue anything). Note btw that there is a fundamental a problem with high value assets (interest don't accrue consistently).

Issue L-4: Adapters could store the index instead of true in the `VaultV2` to remove the O(n) search

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/30

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

Summary

`VaultV2::setIsAdapter()` removes the adapter from the array by doing a O(n) search, which is ineffective.

Vulnerability Detail

The number of adapters is admin controlled so DoS should never happen due to OOG. However, it could not even be a concern in case the index of the adapter plus one (1) was stored in the mapping, such that no search was needed.

Impact

Gas optimization and better security guarantees.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/25/files#diff-b9b86210e027003894f79227889d79167f92c0aa2b2a1b0291f4606002e22540R355

Tool Used

Manual Review

Recommendation

Store the index plus one (1) in the mapping instead and use it to remove from the array.

Discussion

MathisGD

We decided against it because

- we loop anyway in accrueInterest
- we don't try to optimize admin functions

Issue L-5: A `deallocateAll()` function could be useful to successfully guarantee market removal in the Morpho market adapter

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/31

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

Summary

`MorphoMarketV1Adapter::deallocate()` withdraws the given assets, removing a market in case the allocation is null, but it may be hard to achieve this without using the exact shares amount.

Vulnerability Detail

`MorphoMarketV1Adapter::updateList()` is called after every deallocation to remove the market from the list if the assets are null there. However, due to interest accrual, leftover dust may be left after deallocation, which may make it annoying to fully remove a market.

Impact

Removing a market from the list can be delayed, which should just slightly affect gas costs.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/pull/6/files#diff-29a9f0002168ac012f2dbea176776275c39c841f56d6e756b80f663e491f236fR87

Tool Used

Manual Review

Recommendation

Add a `deallocateAll()` function to forcefully remove all funds from the market.

Discussion

MathisGD

Not sure to understand this one? If everything works as expected, you are always able to do deallocate(allocation) (if the market is liquid enough), which effectively removes the market from the list right?

0xsimao

Yeah my point was if interest accrued since the assets were calculated offchain from the shares and the moment the tx settles, leaving an annoying leftover dust allocation which prevents the market from being removed.

Issue L-6: Inconsistency in `accrueInterest()` during deallocation

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/32

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

Vulnerability Detail

In the `allocateInternal()` function, the `accrueInterest()` is executed before the vault's assets are allocated to external protocols. The total assets of the vault before and after allocation are not always the same. For example, some protocols charge a deposit fee. As a result, after allocation, the vault's total assets may decrease slightly. During allocation, these fees are calculated based on the total assets before allocation, without accounting for the fees that will be incurred shortly.

Similarly, during deallocation, some protocols may charge a withdrawal fee, which can cause the vault's total assets to decrease slightly after the withdrawal. However, in the `deallocateInternal()` function, the `accrueInterest()` is executed after assets are withdrawn from external protocols. Thus, the fee is applied to the already reduced total assets following deallocation, due to the withdrawal fee incurred.

Impact

There will be a slight inconsistency in the approach of accruing interest during allocation and deallocation.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/b301a28490d98e100e771f4cc92f91ecf8507ae5/morpho-org__vault-v2/src/VaultV2.sol#L534

Tool Used

Manual Review

Recommendation

Review if the specification is intended to always accrue interests against the total assets, excluding any potential deposit/withdraw fee that might incur during allocation/deallocation. If so, the `accrueInterest()` function should be executed before actual allocation/deallocation takes place.

Discussion

MathisGD

The vault has various problems if adapters loose a lot on allocate/deallocate (for example an allocator can loop allocate/deallocate and make the vault loose assets).

thus we think a comment like that is needed:
https://github.com/morpho-org/vault-v2/pull/712

Though we don't think that a specific fix to this is a good idea.

xiaoming9090

Acknowledged by updating comments.

Issue L-7: Potential out-of-gas revert

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/35

Vulnerability Detail

Understood that `realAssets()` should not revert, in accordance with the liveness requirements:

```
/// LIVENESS REQUIREMENTS
/// @dev List of assumptions that guarantees the vault's liveness properties:
/// - Adapters should not revert on realAssets, otherwise accrueInterestView
↪  reverts.
```

However, there is a (rare) possibility that `MorphoMarketV1Adapter.realAssets()` could revert. The number of markets supported by the adapter is not constrained. Therefore, if the number of supported markets grows sufficiently large, an out-of-gas error could occur when attempting to iterate through all markets to compute the total assets.

Since adding a new market (via `increaseAbsoluteCap` and `increaseRelativeCap`) is timelocked, it significantly limits the possibility that the curator can intentionally DOS by adding a large number of rug markets, as users can monitor it. Thus, marking this as informational.

The same issue is also applicable to the number of adaptors added to the vault, since the number of adaptors is also not restricted.

Impact

The liveness of the vault will be affected if `realAssets()` revert, leading to various issues, such as being unable to deposit or withdraw.

Code Snippet

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/a48ae6219882b693f9a7d8f36c5824c3af1fc8f7/MorphoMarketV1Adapter.sol#L134

Tool Used

Manual Review

Recommendation

Consider documenting the risks in the codebase so that curators are aware of potential pitfalls (e.g., adding too many markets/adaptors can lead to out-of-gas errors) and users are informed of the associated risks.

Discussion

xiaoming9090

The potential risk of OOG revert is added to the codebase's comment.

Issue L-8: MorphoMarketV1Adapter::updateList won't work as expected if there are duplicate adapterIDs for different markets.

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/37

Summary

The `updateList()` function in `MorphoMarketV1Adapter` only handles transitions from allocation > 0 to 0 (remove) and from 0 to > 0 (add), it assumes that each market has a unqiue adapterId. However, if such assumption breaks, the `MorphoMarketV1Adapter` won't work as expected.

Vulnerability Detail

The `updateList()` function in MorphoMarketV1Adapter.sol manages the `marketParamsList` array based on allocation changes, where the allocation is `IVaultV2(parentVault).allocation(keccak256(abi.encode("this/marketParams", address(this), marketParams)));`.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/adapters/MorphoMarketV1Adapter.sol#L106

```
function updateList(MarketParams memory marketParams, uint256 oldAllocation,
↪   uint256 newAllocation) internal {
    if (oldAllocation > 0 && newAllocation == 0) {
        // Remove from list
    } else if (oldAllocation == 0 && newAllocation > 0) {
        // Add to list
    }
}
```

The function only handles two scenarios:

1. `oldAllocation > 0 && newAllocation == 0` → removes market from list

2. `oldAllocation == 0 && newAllocation > 0` → adds market to list

Since the id `abi.encode("this/marketParams", address(this), marketParams)` is not always unique to a single adapter (because adapters can return any `ids[]` array they want), the allocator may allocate assets to another adapter with the same id. This will break the assumption of `allocation(marketParams)` always being zero upon first allocation for the MorphoMarketV1Adapter, and will fail to push the `marketParams` to `marketParamsList` in the `updateList()` function.

Impact

If duplicate adapter IDs existed, a may be allocated with funds, but not pushed into `marketParamsList`, and thus not counted to realAssets().

Recommendation

Add explicit comments that there adapters shouldn't return duplicate ids for market-specific ids.

Discussion

MathisGD

the id is supposed to be unique (thanks to the `this` address(this)). Though as the impact is important, maybe worth explicitly writing at the top of the adapter.

pkqs90

Acknowledged by updating comments.

Issue L-9: SafeERC20Lib `safeApprove` function does not handle USDT like tokens where approval is required to set to 0 before approving.

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/38

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

Summary

SafeERC20Lib `safeApprove` function does not handle USDT like tokens where approval is required to set to 0 before approving.

Vulnerability Detail

The `safeApprove()` function does not set approval to zero before approving. But considering this is only used in adapter constructors where initial approvals are always 0, it doesn't have a large impact. Better to update for future use.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/libraries/SafeERC20Lib.sol#L25

```
function safeApprove(address token, address spender, uint256 value) internal {
    require(token.code.length > 0, ErrorsLib.NoCode());

    (bool success, bytes memory returndata) =
    ↪  token.call(abi.encodeCall(IERC20.approve, (spender, value)));
    require(success, ErrorsLib.ApproveReverted());
    require(returndata.length == 0 || abi.decode(returndata, (bool)),
    ↪  ErrorsLib.ApproveReturnedFalse());
}
```

Discussion

MathisGD

the library isn't meant to be general purpose, so we decide to ack the issue

Issue L-10: Deallocation actions may surpass allocation caps.

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/39

Summary

Deallocation actions may surpass allocation caps.

Vulnerability Detail

When deallocating funds from an adapter, the actual allocation may increase due to accrued interest. However, the allocation caps are not checked against.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/VaultV2.sol#L545

```
    function deallocateInternal(address adapter, bytes memory data, uint256 assets)
        internal
        returns (bytes32[] memory)
    {
        require(isAdapter[adapter], ErrorsLib.NotAdapter());

        (bytes32[] memory ids, int256 change) = IAdapter(adapter).deallocate(data,
        ↪  assets, msg.sig, msg.sender);

        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();
        }

        SafeERC20Lib.safeTransferFrom(asset, adapter, address(this), assets);
        emit EventsLib.Deallocate(msg.sender, adapter, assets, ids, change);
        return ids;
```

```
    }
```

Impact

When deallocation is performed by an allocator, the allocation cap may be surpassed.

Recommendation

Probably a by-design behavior, since we don't want deallocation to revert. Best to comment this.

Discussion

pkqs90

Acknowledged by updating comments.

Issue L-11: Improve comments on share price mechanics

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/40

Vulnerability Detail

1. The comments here mentions "donations are possible but they do not directly increase the share price" is outdated. The current implementation counts the idle `balanceOf(assets)` into totalAssets, so making a donation does directly increase the share price.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/VaultV2.sol#L20-L23

```
/// @dev The vault has 1 virtual asset and a decimal offset of max(0, 18 -
 ↪  assetDecimals). Donations are possible but
/// they do not directly increase the share price. Still, it is possible to inflate
 ↪  the share price through repeated
/// deposits and withdrawals with roundings. In order to protect against that,
 ↪  vaults might need to be seeded with an
/// initial deposit. See
 ↪  https://docs.openzeppelin.com/contracts/5.x/erc4626#inflation-attack
```

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
        ...
@>      uint256 realAssets = IERC20(asset).balanceOf(address(this));
        for (uint256 i = 0; i < adapters.length; i++) {
            realAssets += IAdapter(adapters[i]).realAssets();
        }
        ...
    }
```

2. The total assets (and asset/share ratio) are fixed at the start of the transaction, even if a liquidation of the underlying market happpens, or someone makes a

donation. It is recommended to explicitly comment this for integrators, as this is not common for ERC4626 vaults.

https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/blob/main/morpho-org__vault-v2/src/VaultV2.sol#L577

```
    function accrueInterestView() public view returns (uint256, uint256, uint256) {
@>      if (firstTotalAssets != 0) return (_totalAssets, 0, 0);
        ...
    }
}
```

Discussion

pkqs90

Fixed by updating comments.

Issue L-12: `sharesGate` or `receiveAssetsGate` malfunction can block Vault withdrawals for a substantial period

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/41

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

Summary

Any gates malfunction, being a targeted attack or not, can block withdrawals, while gate setters, being sensitive operations, might have an extended timelock.

Vulnerability Detail

There is no quick way to enable users exit on `sharesGate` or `receiveAssetsGate` unavailability as the only way to disable them is to go through potentially lengthy `setSharesGate()` and `setReceiveAssetsGate()` timelocks. That is, forward operation, gate setting from zero address or another gate, requires a lengthy timelock, while backward operation, dropping the gate, especially in a situation when current gate was hacked or turned malicious, needs to be much quicker, and there is no possibility for that.

Impact

User funds can be temporary frozen with the protocol for the maximum of `setSharesGate()` and `setReceiveAssetsGate()` timelocks.

Code Snippet

`exit()` will fail if `sharesGate` or `receiveAssetsGate` are set and revert the calls:

VaultV2.sol#L719-L721

```
function exit(uint256 assets, uint256 shares, address receiver, address onBehalf)
↪  internal {
    require(canSendShares(onBehalf), ErrorsLib.CannotSendShares());
    require(canReceiveAssets(receiver), ErrorsLib.CannotReceiveAssets());
```

Updating the gates is timelocked and it cannot be too short as at least setting new gates when there are none is a material operation that needs a review:

VaultV2.sol#L332-L342

```
function setSharesGate(address newSharesGate) external {
    timelocked();
    sharesGate = newSharesGate;
    emit EventsLib.SetSharesGate(newSharesGate);
}

function setReceiveAssetsGate(address newReceiveAssetsGate) external {
    timelocked();
    receiveAssetsGate = newReceiveAssetsGate;
    emit EventsLib.SetReceiveAssetsGate(newReceiveAssetsGate);
}
```

Recommendation

Consider adding zero address setters, e.g. `resetSharesGate()`, `resetReceiveAssetsGate()`, having either no or a much shorter timelock.

Discussion

MathisGD

this is documented already:

- https://github.com/morpho-org/vault-v2/blob/b6243f8ba099eb53d16e6577b8ba9201f03a9274/src/VaultV2.sol allowbreak #L119

- https://github.com/morpho-org/vault-v2/blob/b6243f8ba099eb53d16e6577b8ba9201f03a9274/src/VaultV2.sol allowbreak #L123

dmitriia

The issue is not about the general possibility of blocking, it's about what to do when it happens. Comments don't generally solve it.

MathisGD

we decided to ack this.

lpetroulakis

Acknowledged but recognized as a non-issue by the team.

Issue L-13: There is no quick way to cancel a permit

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/42

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

Summary

There is no way for permit cancellation other than using it and resetting the allowance atomically. This would require some preparation and so is not suitable as a quick fix in an operational mistake case.

Impact

Users cannot quickly cancel issued permits for shares allowances.

Code Snippet

`nonces[owner]` is increased on permit usage only:

VaultV2.sol#L805-L810

```
function permit(address _owner, address spender, uint256 shares, uint256 deadline,
↪   uint8 v, bytes32 r, bytes32 s)
    external
{
    require(deadline >= block.timestamp, ErrorsLib.PermitDeadlineExpired());

    uint256 nonce = nonces[_owner]++;
```

Recommendation

Consider adding a `nonces[msg.sender]++` function.

Discussion

MathisGD

Acknowledged. A lot of tokens have this limitation. Also the benefit would be limited by the fact that it's not part of the permit eip.

lpetroulakis

Acknowledged but recognized as a non-issue by the team.

Issue L-14: `relativeCap` can be surpassed by capital intensive manipulation with no funding cost

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/44

Summary

Curator set `relativeCap` can be manipulated by an allocator with making a big enough zero term deposit.

Vulnerability Detail

`firstTotalAssets` protect from flash loan based attacks, but total assets still can be manipulated overall: allocator can enter the Vault with a substantial amount (1st transaction), then run the desired allocation, ignoring the relative caps as total assets will be bloated, and then exit atomically (2nd transaction, even in the same block).

This kind of attack requires capital and so is less accessible than a flash loan based one, but still possible. If done in a single block it requires no funding, just capital access itself along with covering the corresponding transaction costs.

Impact

Vault allocation can be performed not adhering to the `relativeCap` setting by any allocator with capital access.

Code Snippet

`firstTotalAssets` will be updated to a bigger figure, that includes any deposit just made, on a next transaction even in the same block:

VaultV2.sol#L565-L569

```
    function accrueInterest() public {
        (uint256 newTotalAssets, uint256 performanceFeeShares, uint256
        ↪  managementFeeShares) = accrueInterestView();
        emit EventsLib.AccrueInterest(_totalAssets, newTotalAssets,
        ↪  performanceFeeShares, managementFeeShares);
        _totalAssets = newTotalAssets.toUint128();
>>      if (firstTotalAssets == 0) firstTotalAssets = newTotalAssets;
```

Recommendation

Consider using TWA total assets instead of `firstTotalAssets`, this way adding funding to the attacker's cost.

Issue L-15: Adapter's `forceDeallocatePenalty` needs to exceed withdrawal fee of any underlying market

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/46

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

Summary

If underlying market or Vault have withdrawal fee then `forceDeallocatePenalty` of the corresponding adapter have to exceed it with `1/(1-fee)` multiplier, otherwise exiting depositors would be able to pass a part of withdrawal fee to the remaining ones.

Vulnerability Detail

Since total assets are fixed at the start of transaction and exact quantity is being pulled out from the adapter, so it's the net after fee assets, the `forceDeallocatePenalty[adapter]` should always exceed underlying market's withdrawal fee, if any is present, with `1/(1-fee)` multiplier.

Impact

If `forceDeallocatePenalty[adapter]` isn't high enough then the depositors can routinely exit with the remaining ones collectively paying a part of the adjusted fee difference, i. e. what Vault / all the depositors paying for the withdrawal can exceed what exiting user paying as the penalty, so they can steal from all the other depositors.

Code Snippet

`assets` is what requested and delivered, with Vault additionally paying withdrawal fee, if any:

VaultV2.sol#L749-L755

```
function forceDeallocate(address adapter, bytes memory data, uint256 assets,
↪  address onBehalf)
    external
    returns (uint256)
{
    bytes32[] memory ids = deallocateInternal(adapter, data, assets);
    uint256 penaltyAssets = assets.mulDivUp(forceDeallocatePenalty[adapter], WAD);
    uint256 penaltyShares = withdraw(penaltyAssets, address(this), onBehalf);
```

Recommendation

Consider documenting this dependency in Curator guidelines.

Discussion

MathisGD

same answer as #32 and #36 (see the fix of #36)

dmitriia

It's recommended to document the exact withdrawal and penalty fee relationship.

Issue L-16: `maxRate` set too low can steal yield from depositors

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/47

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

Summary

Setting `maxRate` too low is a griefing surface available to curator: if this level is then being exceeded systematically, the unrealized part will be pulled over and grow indefinitely, with Vault's depositors not ever having access to it.

Impact

For example, if `maxRate` is set to zero, then any profit realization is prevented, Vault's depositors will have no yield and are forced to deallocate, paying the penalties.

Code Snippet

Zero is the minimum value for `newMaxRate`:

VaultV2.sol#L484-L491

```
function setMaxRate(uint256 newMaxRate) external {
    timelocked();
    require(newMaxRate <= MAX_MAX_RATE, ErrorsLib.MaxRateTooHigh());

    accrueInterest();

    // Safe because newMaxRate <= MAX_MAX_RATE < 2**64-1.
    maxRate = uint64(newMaxRate);
```

Recommendation

Consider adding and controlling for a minimum for `maxRate`, e.g. `MIN_MAX_RATE = 10e16 / uint256(365 days); // 10% APR`.

Discussion

MathisGD

The vault doesn't claim to provide trustlessness on yield (only on principal) (and not only for this reason), thus we decided to acknowledge.

lpetroulakis

Acknowledged but recognized as a non-issue by the team.

Issue L-17: Taking management fee off the back propagated current assets artificially increases it

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/48

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

Summary

For `managementFeeAssets` calculation `newTotalAssets` is being back propagated to the whole `elapsed` seconds period, during which total assets drifted from `_totalAssets` to `newTotalAssets` with exact path being unknown.

Vulnerability Detail

Using `newTotalAssets` is akin to stating that assets tend to decrease and so it must have been at least `newTotalAssets` during the period.

That is, using upper boundary means fee overstating, while using lower boundary means using old figure, `_totalAssets`, as the assets tend to grow over time.

Impact

Management fee taken is bloated compared to the publicly visible `managementFee` setting.

Code Snippet

`managementFeeAssets` is computed off `newTotalAssets` with the comment that shorter formulas somehow justify the additional error:

VaultV2.sol#L594-L598

```
// The management fee is taken on newTotalAssets to make all approximations
 ↪  consistent (interacting less
// increases fees).
uint256 managementFeeAssets = elapsed > 0 && managementFee > 0 &&
 ↪  canReceiveShares(managementFeeRecipient)
    ? (newTotalAssets * elapsed).mulDivDown(managementFee, WAD)
    : 0;
```

Recommendation

More conservative take here is `(_totalAssets * elapsed).mulDivDown(managementFee, WAD)`, i.e. 'as assets tend to grow most of the time it most probably were at least `_totalAssets` there, so using it as a lower boundary'. To accommodate for that the `newTotalAssetsWithoutFees` base needs to be separated for two cases, for management and performance fees correspondingly.

Discussion

MathisGD

The reason why we choose this direction is to make all approximations consistent: interacting less increases the fees (assuming no losses here, which should be true on most cases). Note that this is documented already.

Taking the other side has the downside of making people want to ping the vault, and it is not consistent with rounding down the fees.

Thus we decided to not fix (ack).

Issue L-18: `abdicateSubmit` timelock can be decreased to zero, which allows Curator to instantly freeze all Vault's deposits

Source: https://github.com/sherlock-audit/2025-08-morpho-vault-v2-aug-13th/issues/49

Summary

Instant `abdicateSubmit` can allow Curator to freeze all Vault's deposits permanently.

Vulnerability Detail

Example attack is as follows:

Suppose there is a Vault with material holdings, which are fully invested with no liquidity adapter, so any depositor needs to run force deallocate to exit.

1) Curator can run `setSharesGate()`, setting the shares gate to some upgradeable contract,
2) simultaneously run `decreaseTimelock()` for `abdicateSubmit`, setting timelock to zero for it,
3) while (1) and (2) actions look plausible and can go through without alarming the depositors, but when they be executed curator becomes able to atomically upgrade `sharesGate` to be ever reverting,
4) run `abdicateSubmit()` for `setSharesGate()` and `setIsAdapter()`, locking current gate and adapters,
5) which results in permanently freezing all the user funds in the Vault as `exit()` will be blocked and it also be impossible to craft and add a rescue adapter, while normal adapters won't directly send funds to the users. Due to (4) setting new curator doesn't look to solve this either.

Impact

Curator have an option to permanently freeze all the user funds without timelock delays.

Code Snippet

VaultV2.sol#L390-L398

```
function decreaseTimelock(bytes4 selector, uint256 newDuration) external {
    timelocked();
    require(selector != IVaultV2.decreaseTimelock.selector,
    ↪  ErrorsLib.TimelockCapIsFixed());
    require(timelock[selector] != type(uint256).max, ErrorsLib.InfiniteTimelock());
    require(newDuration <= timelock[selector], ErrorsLib.TimelockNotDecreasing());

    timelock[selector] = newDuration;
    emit EventsLib.DecreaseTimelock(selector, newDuration);
}
```

Recommendation

Consider adding `abdicateSubmit` selector to the `require(selector != IVaultV2.decreaseTimelock.selector, ...` control.

Discussion

MathisGD

there are these comments on the abdicate function, don't you think that they are enough?

```
/// @dev Irreversibly disable submit for a selector.
/// @dev Be particularly careful as this action is not reversible.
```

MathisGD

Btw, I don't really see this as a Medium issue if people consider abdicate carefully (notably it should have a long timelock)

dmitriia

The issue is about the ability to decrease the timelock for `abdicateSubmit` [by Curator, with a malicious intent], not about possible misunderstanding the nature of the function or its implications. I don't think there is much valid usage for instant or even just fast `abdicateSubmit`, so removing the ability of timelock decrease for it basically doesn't touch valid flows, but does remove the surface. Since it's dishonest Curator based it can be Low.

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.