Morpho Vaults v2

Competition

November 5, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

A competition provides a broad evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While competitions endeavor to identify and disclose all potential security issues, they cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities, therefore, any changes made to the code would require an additional security review. Please be advised that competitions are not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
High	*Must* fix as soon as possible (if already deployed) and can be triggered by any user without significant constraints, generating outsized returns to the exploiter. For example: loss of user funds (significant amount of funds being stolen or lost) or breaking core functionality (failure in fundamental protocol operations).
Medium	Global losses <10% or losses to only a subset of users, requiring significant constraints (capital, planning, other users...) to be exploited. For example: temporary disruption or denial of service (DoS), minor fund loss or exposure or breaking non-core functionality
Low	Losses will be annoying but easily recoverable, requiring unusual scenarios or admin actions to be exploited.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above matrix. High severity findings represent the most critical issues that must be addressed immediately, as they either have high impact and high likelihood of occurrence, or medium impact with high likelihood.

Medium severity findings represent issues that, while not immediately critical, still pose significant risks and should be addressed promptly. These typically involve scenarios with medium impact and medium likelihood, or high impact with low likelihood.

Low severity findings represent issues that, while not posing immediate threats, could potentially cause problems in specific scenarios. These typically involve medium impact with low likelihood, or low impact with medium likelihood.

Lastly, some findings might represent improvements that don't directly impact security but could enhance the codebase's quality, readability, or efficiency (Gas and Informational findings).

2 Security Review Summary

Morpho is a trustless and efficient lending primitive with permissionless market creation.

From Jul 15th to Jul 25th Cantina hosted a competition based on morpho-vaults-v2 at commit hash 5938a924. The participants identified a total of **31** issues in the following risk categories:

- High Risk: 0
- Medium Risk: 3
- Low Risk: 9
- Gas Optimizations: 0
- Informational: 19

The present report only outlines the **high** and **medium** risk issues.

3 Findings

3.1 Medium Risk

3.1.1 Loss of funds for users as the manual VIC forgets about interest before the deadline

Submitted by samuraii77, also found by 0x37, Auditten, carlos404, JesJupyter, pkqs90, raiseUp, 0xpiken, patitonar, 0x9527, alix40, Rhaydden and kind0dev

Severity: Medium Risk

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

Description: The interest per second is computed like this:

```
uint256 tentativeInterestPerSecond = vic != address(0) ? IVic(vic).interestPerSecond(_totalAssets, elapsed) : 0;
```

For the manual VIC, the function is as follows:

```
function interestPerSecond(uint256, uint256) external view returns (uint256) {
        return block.timestamp <= deadline ? storedInterestPerSecond : 0;
}
```

However, the function is too simple and "forgets" past, non-accrued interest.

The following will happen:

1. The vault works with some underlying vault (doesn't matter which one), the manual VIC is used to get the interest.
2. Bob deposits 1000 assets, let's say he is the only depositor for simplicity.
3. The interest that has to be distributed is 100, thus the admin sets a deadline 10 seconds in the future with 10 interest per second.
4. After 10 seconds, the assets Bob must withdraw should be 1100 assets.
5. No one interacts with the vault for 20 seconds, Bob redeems all of his shares, supposed to get 1100 assets as enough time has passed.
6. As the deadline is over, then the interest per second that will be returned is 0. The VIC forgot about all of the interest during the initial 10 seconds.
7. Bob receives 1000 assets and 0 interest, he essentially lost 100 assets.

Recommendation: Refactor the VIC and the vault interest accrual logic to handle such a scenario.

3.1.2 Incorrect loss calculation

Submitted by Auditten, also found by franfran20, 0xvangrim, 0x37, 0x37 and patitonar

Severity: Medium Risk

Context: VaultV2.sol#L751

Summary: Values of `cap[id].allocation` are updated only when some assets are allocated or deallocated in corresponding adapters. When `realizeLoss()` is called, the value of `cap[id].allocation` does not include interest for the time since last update. This results in asynchrony between `_totalAssets` and allocations.

Finding Description: The `cap[id].allocation` values are updated only during allocation or deallocation events in the corresponding adapters, while `_totalAssets` is updated more frequently, on nearly every interaction with the vault.

When `VaultV2.realizeLoss` is called, it invokes `IAdapter(adapter).realizeLoss()` on the specified adapter. If the adapter is `MorphoVaultV1Adapter.sol`, the loss is calculated as follows:

```
uint256 loss = allocation() - IERC4626(morphoVaultV1).previewRedeem(shares);
```

Let's denote the following:

- A: The interest generated by this adapter that was included in `_totalAssets` during `accrueInterest`, but not reflected in `cap[adapterId].allocation` (due to no calls to allocate or deallocate).
- L: The loss that occurred in this adapter.

The value of `IERC4626(morphoVaultV1).previewRedeem(shares)` equals `allocation() + A - L`.

There are two possible scenarios:

- If $A > L$: Then allocation() $-$ (allocation() $+ A - L$) results in a negative value, causing the call to revert. This prevents reporting the loss, and `_totalAssets` remains higher than it should be.
- If $A \leq L$: The realized loss reported by the adapter will be allocation() $-$ (allocation() $+ A - L$) $= L - A$, which means `_totalAssets` will decrease by $L - A$ instead of the full loss L. As a result, `_totalAssets` remains higher than it should be.

Impact Explanation: High: "*Loss of User Funds: A vulnerability that could lead to a significant amount of funds being stolen or lost*".

The value of `_totalAssets` will be higher than expected. As a result:

- Users withdrawing after the loss happened may receive more funds even if `realizeLoss` was called.
- Users depositing after the loss happened may receive less shares than they should even if `realizeLoss` was called.

Likelihood Explanation: Low: "*Unusual scenarios, such as paused or exception state*".

Recommendation: There's no straightforward solution to this issue, as the core problem is that `_totalAssets` can be updated in `accrueInterest()` without updating the allocation of each adapter.

However, `_totalAssets` and `cap[id].allocation` can be synchronised by introducing a `realizeProfit` function, similar to the existing `realizeLoss`. With this approach, there's no need to retract `interestPerSecond` from Vault interest controllers, making them unnecessary.

Users can be incentivised to report profits just like losses, by rewarding them with 1% of the reported amount. A call to `realizeProfit` would be worthwhile as long as the gas cost is less than the incentive. The amount of gas needed to do this call will depend on the adapter's implementation of `realizeProfit`.

With the current implementation, the amount of gas required to make a `realizeLoss` call for a single Morpho Market is near 100'000. With a gas price of 0.8 gwei, the call will cost 80'000 gwei, and the minimum value that has incentive to be reported is 100 * 80'000 gwei = 8'000'000 gwei = 0.008 ETH. So the maximum difference of `_totalAssets` and real total assets will not be larger than this value, and from the perspective of the whole vault, it is acceptable.

Unfortunately, profit realisation for adapters that allocate in `VaultV1` with a large withdraw queue will require more gas, but use of such adapters doesn't make a lot of sense in the current implementation, either.

3.1.3 Incorrect interest calculation in `SingleMorphoVaultV1Vic`

Submitted by Auditten, also found by ZeroEx and Bigsam

Severity: Medium Risk

Context: SingleMorphoVaultV1Vic.sol#L36-L37

Summary: In `SingleMorphoVaultV1Vic.sol`, interest per second is derived from estimated profit, based on the difference between current and previously tracked asset values. However, if a loss occurs and the current vault balance exceeds the unallocated balance (due to donations or penalties from `forceDeallocate`) `vic` may prematurely account for part of the loss. This results in the same loss being recorded twice: once during interest accrual and again when `realizeLoss` is called, leading to double-counting.

Finding Description: In `SingleMorphoVaultV1Vic.sol`, interest per second is calculated based on the estimated profit profit $= \max(0, \text{realAssets} - \text{totalAssets})$.

- realAssets $=$ balance $+$ previewRedeem$_{\text{now}}$, where:
  - balance is the current vault balance.
  - previewRedeem$_{\text{now}}$ is the current value of assets redeemable from Morpho Vault V1 for the adapter's shares.

- totalAssets $=$ idleBalance $+$ previewRedeem$_{\text{prev}}$, where:
  - idleBalance is the unallocated balance remaining in the vault.
  - previewRedeem$_{\text{prev}}$ is the value of redeemable assets at the time `accrueInterest` was last called.

If a loss occurs, the difference loss $=$ previewRedeem$_{\text{now}} -$ previewRedeem$_{\text{prev}}$ is negative and should not be reported by `vic`, since it will be accounted by a call to `realizeLoss`. However, if balance $-$ idleBalance $>$ 0 (and less than maximum possible returned value -- `maxInterestPerSecond * elapsed`), then `vic` may report $\min(|\text{loss}|, \text{balance} - \text{idleBalance})$ amount of losses, causing part of the loss to be included prematurely. When `realizeLoss` is later called, `_totalAssets` will end up reflecting both this partial loss and the full loss again, resulting in double-counting.

The current vault balance can exceed the unallocated assets (i.e., those not held in the adapter) in two scenarios:

- A donation of the underlying asset is made directly to the vault.
- `forceDeallocate` has been called, and a non-zero penalty was applied. The value of penalty will be included in balance but not in idleBalance.

While donations are unlikely, the second scenario is common.

Impact Explanation: High: "*Loss of User Funds: A vulnerability that could lead to a significant amount of funds being stolen or lost*".

The value of `_totalAssets` will be lower than expected. As a result:

- The management fee will be charged for less amount than expected.
- Users withdrawing in the same block may lose funds.
- Users depositing during that block may receive more shares than they should.

Likelihood Explanation: Low: "*Unusual scenarios, such as paused or exception state*".

Recommendation: This issue can be resolved by correctly calculating the vault's profit as:

$$\text{balance} - \text{idleBalance} + \max(0, \text{previewRedeem}_{\text{now}} - \text{previewRedeem}_{\text{prev}})$$

To enable this, `VaultV2` should store the current `idleBalance` and update it within the `allocateInternal`, `deallocateInternal`, `enter`, and `exit` functions. At VaultV2.sol#L711, this stored value should be used instead of `IERC20(asset).balanceOf(address(this))`.

To account for the remaining balance as idle assets, similar to how penalty assets are currently included over time via `VaultV2.accrueInterest`, the `SingleMorphoVaultV1Vic.interestPerSecond` function can be modified accordingly.

```
function interestPerSecond(uint256 totalAssets, uint256 idleBalance, uint256 elapsed) external view returns
↪  (uint256, uint256) {
    uint256 currentPreviewRedeem =
    ↪  IERC4626(morphoVaultV1).previewRedeem(IMorphoVaultV1Adapter(morphoVaultV1Adapter).shares());
    uint256 previusPreviewRedeem = totalAssets - idleBalance;

    uint256 penaltyInterest = IERC20(asset).balanceOf(parentVault) - idleBalance;
    uint256 interest = currentPreviewRedeem.zeroFloorSub(previusPreviewRedeem);

    uint256 maxInterestPerSecond = uint256(totalAssets).mulDivDown(MAX_RATE_PER_SECOND, WAD);
    uint256 tentativeInterestPerSecond = interest / elapsed;
    if (tentativeInterestPerSecond >= maxInterestPerSecond)
        return (maxInterestPerSecond, 0);

    uint256 tentativePenaltyInterestPerSecond = penaltyInterest / elapsed;

    if (tentativeInterestPerSecond + tentativePenaltyInterestPerSecond <= maxInterestPerSecond)
        return (tentativeInterestPerSecond + tentativePenaltyInterestPerSecond,
        ↪  tentativePenaltyInterestPerSecond);
    return (maxInterestPerSecond, maxInterestPerSecond - tentativeInterestPerSecond);
}
```

In this updated function, the first returned value represents the change to `_totalAssets`, and the second represents the change to `_idleBalance`. The same should be done in `ManualVic.interestPerSecond`.