Blend Contracts PRs 47, 48 & 49

Security Review

Solo review by:
Sujith Somraaj, Security Researcher

September 29, 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 security review is a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While the review endeavors to identify and disclose all potential security issues, it 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 that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that a security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
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 table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Blend's Intent Engine delivers sustainable yield and auto-derisks your capital.

From Aug 27th to Aug 28th the security researchers conducted a review of contracts on PR 47, PR 48, and PR 49. A total of **5** issues were identified:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	5	5	0
Total	**5**	**5**	**0**

3 Findings

3.1 Informational

3.1.1 Redundant controller existence validation

Severity: Informational

Context: StrategyManager.sol#L277

Description: The `executeVaultAction()` function contains a redundant validation check that duplicates logic already performed by the `validActionController` modifier. The function performs a loop to verify that the provided `actionController` exists in the vault's approved action controllers list:

```
bool found = false;
for (uint256 i = 0; i < vaultConfig.actions.length; i++) {
    if (vaultConfig.actions[i].controller == actionController) {
        strategyData = vaultConfig.actions[i].data;
        found = true;
        break;
    }
}
require(found, InvalidActionController());
```

However, the `validActionController` modifier already performs an identical validation check to ensure the action controller is approved for the given vault:

```
modifier validActionController(address vault, IVaultActionController actionController) {
    // ... vault validation ...

    bool found = false;
    for (uint256 i = 0; i < vaultConfig.actions.length; ++i) {
        if (vaultConfig.actions[i].controller == actionController) {
            found = true;
            break;
        }
    }
    require(found, InvalidActionController());
    _;
}
```

Recommendation: Remove the redundant validation check from the executeVaultAction function and rely solely on the validActionController modifier for validation:

```
function executeVaultAction(
    address safe,
    address vault,
    IVaultActionController actionController,
    bytes calldata data
) external onlyExecutor validVault(vault) validActionController(vault, actionController) initiateVaultAction {
    VaultConfig memory vaultConfig = strategyConfig[vault];
    bytes memory strategyData;

    // Since the validActionController modifier guarantees the controller exists,
    // we can safely retrieve the strategy data without additional validation
    for (uint256 i = 0; i < vaultConfig.actions.length; i++) {
        if (vaultConfig.actions[i].controller == actionController) {
            strategyData = vaultConfig.actions[i].data;
            break;
        }
    }

    // ....
}
```

BlendMoney: Fixed in PR 53.

Reviewer: Fix verified.

3.1.2 Inefficient loop structure

Severity: Informational

Context: MorphoVaultController.sol#L178-L182

Description: The current `executeRebalance()` implementation uses two separate loops for processing rebalance operations, of which the decreaseAllocation path is gas inefficient and could be optimized.

Recommendation: Consider fixing the function as follows:

```
- for (uint256 i = 0; i < rebalanceData.length; ++i) {
-     if (!rebalanceData[i].isIncrease) {
-         if (rebalanceData[i].amount > 0) {
-             rebalanceAmounts[i] = _decreaseAllocation(vaultConfig.markets[i], rebalanceData[i]);
-         }
-         rebalanceAreIncreases[i] = false;
-     }
+ for (uint256 i = 0; i < rebalanceData.length; ++i) {
+     if (!rebalanceData[i].isIncrease && rebalanceData[i].amount > 0) {
+         rebalanceAmounts[i] = _decreaseAllocation(vaultConfig.markets[i], rebalanceData[i]);
+     }
```

BlendMoney: Fixed in PR 55.

Reviewer: Fix verified.

3.1.3 Redundant External Calls to `WRAPPED_NATIVE()`

Severity: Informational

Context: MorphoVaultController.sol#L149

Description: In the `executeRebalance function()`, there are two external calls to `GENERAL_ADAPTER.WRAPPED_NATIVE()` that retrieve the same immutable value. Since `WRAPPED_NATIVE()` returns a constant address that doesn't change during contract execution, making multiple external calls to retrieve the same value is inefficient and wastes gas.

Recommendation: Cache the result of `GENERAL_ADAPTER.WRAPPED_NATIVE()` in a local variable at the beginning of the function and reuse this cached value throughout the function execution.

BlendMoney: Fixed in PR 54.

Reviewer: Fix verified.

3.1.4 Increase test coverage

Severity: Informational

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

Description: Of the three PRs (47, 48, 49) currently under review, two of them (47, 48) lack test coverage for the contract changes introduced. Ensuring comprehensive testing and consistent reporting is essential to verify that the codebase operates correctly. Without adequate testing, there is a risk of unforeseen errors and regressions when smart contracts are updated.

Recommendation: Enhance test coverage to ensure all new execution paths are covered.

BlendMoney: Fixed in PR 57.

Reviewer: Fix verified.

3.1.5 Possible division by zero in share price validation

Severity: Informational

Context: MorphoVaultController.sol#L161

Description: In the `executeRebalance()` function, there is a potential division by zero vulnerability when validating the share price slippage. The code performs a division operation `balanceOfSafe.rDivUp(shares)` without first ensuring that the shares value is greater than zero.

```
// Line ~85-90 in executeRebalance function
uint256 shares = IERC4626(vault).deposit(balanceOfSafe, address(this));
```

```
SafeERC20.forceApprove(IERC20(vaultToken), address(vault), 0);
require(balanceOfSafe.rDivUp(shares) <= maxSharePriceE27, SlippageExceeded());
```

Recommendation: Add a check to ensure shares is greater than zero before performing the division operation:

```
+ require(shares > 0, "Zero shares minted");
  require(balanceOfSafe.rDivUp(shares) <= maxSharePriceE27, SlippageExceeded());
```

BlendMoney: Fixed in PR 58.

Reviewer: Fix verified.