Blend PR 69
Security Review

Solo review by:
Sujith Somraaj, Security Researcher

October 10, 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 level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 Oct 3rd to Oct 4th the security researchers conducted a review of contracts on commit hash a30b9e0f. A total of **7** 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	2	1	1
Gas Optimizations	1	1	0
Informational	4	4	0
Total	**7**	**6**	**1**

3 Findings

3.1 Low Risk

3.1.1 Event `SwapExecuted` missing actual output amount

Severity: Low Risk

Context: SwapAdapter.sol#L135

Description: The `SwapExecuted` event only emits the minimum required output amount (amountOutMin) but does not include the actual amount received from the swap:

```
event SwapExecuted(
    address indexed fromToken,
    address indexed toToken,
    uint256 amountIn,
    uint256 amountOutMin,
    address indexed recipient
);
```

In both `swapToCollateral()` and `swapToLoanToken()` functions, the actual swap output amount is known before emitting the event (it's the balance transferred to the recipient), but this important information is not recorded:

```
// In swapToCollateral
collateralToken.safeTransfer(recipient, collateralToken.balanceOf(address(this)));
emit SwapExecuted(address(loanToken), address(collateralToken), amount, amountOutMin, recipient);

// In swapToLoanToken
loanToken.safeTransfer(recipient, loanToken.balanceOf(address(this)));
emit SwapExecuted(address(collateralToken), address(loanToken), amount, amountOutMin, recipient);
```

Consequently, off-chain systems are unable to monitor actual swap execution prices and slippage experienced precisely.

Recommendation: Add an amountOut parameter to the event and emit the actual received amount:

```
event SwapExecuted(
    address indexed fromToken,
    address indexed toToken,
    uint256 amountIn,
    uint256 amountOutMin,
    uint256 amountOut,  // Add this parameter
    address indexed recipient
);
```

BlendMoney: Fixed in PR 75.

Sujith Somraaj: Fix verified.

3.1.2 Intermediary token dust not refunded in multi-hop swaps leading to value leakage

Severity: Low Risk

Context: SwapAdapter.sol#L192

Description: The `SwapAdapter.sol` contract explicitly supports multi-hop swaps through sequential SwapCall[] executions, as documented in the `_executeSwap()` function:

```
/**
 * @dev Iterates through the calls array, approving each target, executing the call, then resetting approval to
 ↪  zero.
 *  Allows complex multi-hop swaps (e.g., token A → token B → token C). No intermediate amount checks are
 ↪  performed.
 */
```

However, when executing multi-hop swaps (e.g., loanToken → intermediaryToken → collateralToken), the contract only validates and transfers the final output token to the recipient. Any amounts of intermediary tokens left in the contract after the swap are neither refunded to the user nor validated by the emptyAdapterBalances modifier:

```
modifier emptyAdapterBalances(IERC20 loanToken, IERC20 collateralToken) {
    _;
    // Only checks loan and collateral tokens - intermediary tokens ignored
    require(loanToken.balanceOf(address(this)) == 0, ZeroAmount());
    require(collateralToken.balanceOf(address(this)) == 0, ZeroAmount());
}
```

For example:

- User swaps 1000 USDC → WETH → wstETH.
- Swap leaves 0.001 WETH dust in the adapter (standard due to rounding in DEX routers).
- The dust remains stuck in the contract.
- Next user performing any swap involving WETH could steal this dust, or it remains permanently locked.

Recommendation: Consider refunding intermediary tokens to recipient at the end of the swap call.

BlendMoney: We acknowledge that the `SwapAdapter` does not refund dust from intermediary tokens in multi-hop swaps. This is a deliberate design choice prioritizing gas efficiency and simplicity.

Our Rationale → Generic "Black Box" Design: The `SwapAdapter` is designed to execute a series of swaps without needing to know the specific path. Its only job is to verify the final output against the user's slippage setting. Tracking every possible intermediary token would violate this simple, robust design and add significant complexity.

Gas Overhead: Implementing a refund mechanism would require passing an array of all intermediary tokens and iterating through them to check balances and perform transfers. This would increase gas costs for all swaps, even single-hop ones that don't leave dust.

Dust as Acceptable Slippage: We view leftover dust as a form of positive slippage or a rounding artifact from the underlying DEXs. The user is always guaranteed to receive their minimum expected output (amountOutMin). The minuscule value of any dust is well within the acceptable slippage cost they've already defined for the trade.

Risk Assessment: The risk is low. The value of the dust is negligible, and the possibility of another user capturing it is unpredictable and not a reliable attack vector. Most importantly, the user initiating the swap never loses funds they were guaranteed to receive.

Conclusion: We accept this finding as a conscious design trade-off. The benefits of maintaining a simple, gas-efficient, and generic adapter outweigh the low risk associated with unrefunded dust. We will update our documentation to make this behavior explicit to future developers and auditors.

Sujith Somraaj: Acknowledged.

3.2 Gas Optimization

3.2.1 Inefficient use of memory for `strategyData` Parameter increases gas costs

Severity: Gas Optimization

Context: SwapAdapter.sol#L151

Description: Both `swapToCollateral()` and `swapToLoanToken()` functions declare the strategyData parameter as bytes memory:

```
function swapToCollateral(
    IERC20 loanToken,
    IERC20 collateralToken,
    address recipient,
    bytes memory strategyData,  // Using memory
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ...
    StoredStrategyData memory strategyConfig = abi.decode(strategyData, (StoredStrategyData));
}

function swapToLoanToken(
    IERC20 loanToken,
```

```
    IERC20 collateralToken,
    address recipient,
    bytes memory strategyData,  // Using memory
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ...
    StoredStrategyData memory strategyConfig = abi.decode(strategyData, (StoredStrategyData));
}
```

Since `strategyData` is:

- Only read from, never modified.
- Passed as an external parameter.
- Decoded directly without requiring in-memory manipulation.

Using `bytes memory` incurs unnecessary gas costs by copying the data from `calldata` to `memory`.

Recommendation: Change the `strategyData` parameter to `bytes calldata` in both functions:

```
function swapToCollateral(
    IERC20 loanToken,
    IERC20 collateralToken,
    address recipient,
    bytes calldata strategyData,  // Changed to calldata
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ... rest remains the same
}

function swapToLoanToken(
    IERC20 loanToken,
    IERC20 collateralToken,
    address recipient,
    bytes calldata strategyData,  // Changed to calldata
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ... rest remains the same
}
```

BlendMoney: Fixed in PR 78.

Sujith Somraaj: Fix verified.

3.3 Informational

3.3.1 Missing upper bound validation for `maxSlippageBps` in `strategyConfig`

Severity: Informational

Context: SwapAdapter.sol#L115, SwapAdapter.sol#L159

Description: The `StoredStrategyData.maxSlippageBps` parameter lacks validation to ensure it does not exceed 10,000 basis points (100%). When strategyData is decoded in both `swapToCollateral()` and `swapToLoanToken()` functions, the contract assumes the maxSlippageBps value is valid without enforcing an upper bound check. Since basis points represent percentages where 10,000 BPS = 100%, any value exceeding this threshold is semantically invalid. Setting a value above 10,000 causes underflow; however, it is advisable to perform an explicit check.

Recommendation: Add validation wherever StoredStrategyData is created or set:

```
+ require(strategyConfig.maxSlippageBps <= 10000, "Invalid max slippage: exceeds 100%");
```

BlendMoney: Fixed in PR 74.

Sujith Somraaj: Fix verified.

3.3.2 Unused library imports

Severity: Informational

Context: SwapAdapter.sol#L18-L20

Description: The `SwapAdapter.sol` contract imports Math library from OpenZeppelin and BytesLib from bundler3, along with their corresponding using statements:

```
import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";
import {BytesLib} from "bundler3/src/libraries/BytesLib.sol";

using Math for uint256;
using BytesLib for bytes;
```

However, neither library is utilized anywhere in the contract code.

Recommendation: Consider removing the unused imports and their corresponding using statements.

BlendMoney: Fixed in PR 73.

Sujith Somraaj: Fix verified.

3.3.3 Missing zero address validation for recipient parameter

Severity: Informational

Context: SwapAdapter.sol#L106

Description: Both `swapToCollateral()` and `swapToLoanToken()` functions accept a recipient parameter but do not validate that it is not the zero address (address(0)) before transferring tokens:

```
function swapToCollateral(
    IERC20 loanToken,
    IERC20 collateralToken,
    address recipient,  // No validation
    bytes memory strategyData,
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ... swap logic ...
    collateralToken.safeTransfer(recipient, collateralToken.balanceOf(address(this)));
}

function swapToLoanToken(
    IERC20 loanToken,
    IERC20 collateralToken,
    address recipient,  // No validation
    bytes memory strategyData,
    bytes calldata extraData
) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
    // ... swap logic ...
    loanToken.safeTransfer(recipient, loanToken.balanceOf(address(this)));
}
```

If `address(0)` is passed as the recipient (either accidentally or due to a bug in calling contracts), the swapped tokens will be permanently lost as they are sent to the zero address, which is an unrecoverable burn address.

Recommendation: Consider adding zero address validation at the beginning of both functions:

```
  error InvalidRecipient();

  function swapToCollateral(
      IERC20 loanToken,
      IERC20 collateralToken,
      address recipient,
      bytes memory strategyData,
      bytes calldata extraData
  ) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
+     require(recipient != address(0), InvalidRecipient());
      // ... rest of function
  }

  function swapToLoanToken(
      IERC20 loanToken,
      IERC20 collateralToken,
      address recipient,
      bytes memory strategyData,
```

```
      bytes calldata extraData
  ) public virtual override emptyAdapterBalances(loanToken, collateralToken) {
+     require(recipient != address(0), InvalidRecipient());
      // ... rest of function
  }
```

BlendMoney: Fixed in PR 76.

Sujith Somraaj: Fix verified.

3.3.4 Misleading error message in `emptyAdapterBalances` modifier

Severity: Informational

Context: SwapAdapter.sol#L90-L91

Description: The emptyAdapterBalances modifier uses the `ZeroAmount()` error to validate that token balances are zero after swap execution:

```
modifier emptyAdapterBalances(IERC20 loanToken, IERC20 collateralToken) {
    _;
    // Adapter should not retain any tokens after swap
    require(loanToken.balanceOf(address(this)) == 0, ZeroAmount());
    require(collateralToken.balanceOf(address(this)) == 0, ZeroAmount());
}
```

However, this creates semantic confusion because the `ZeroAmount()` error is defined and used elsewhere in the contract to indicate when an amount IS zero.

Recommendation: Create a dedicated, descriptive error for this specific validation:

```
/// @notice Thrown when the adapter retains tokens after swap execution
error AdapterNotEmpty();

modifier emptyAdapterBalances(IERC20 loanToken, IERC20 collateralToken) {
    _;
    // Adapter should not retain any tokens after swap
    require(loanToken.balanceOf(address(this)) == 0, AdapterNotEmpty());
    require(collateralToken.balanceOf(address(this)) == 0, AdapterNotEmpty());
}
```

BlendMoney: Fixed in PR 77.

Sujith Somraaj: Fix verified.