Blend Contracts PR 62
Security Review

Solo review by:
Rvierdiiev, 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 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 Sep 16th to Sep 19th the security researchers conducted a review of contracts on commit hash cd25c439. A total of **4** 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	1	1	0
Informational	3	0	3
Total	**4**	**1**	**3**

3 Findings

3.1 Gas Optimization

3.1.1 Avoid external call in `AcrossXChainAdapter.execute`

Severity: Gas Optimization

Context: AcrossXChainAdapter.sol#L123

Description: The `AcrossXChainAdapter.execute` function currently performs an external call to `isChainSupported()`. This adds unnecessary overhead, since the information is already available in the local `_enabledChainIds` mapping. By directly referencing the mapping, the check can be simplified and gas usage reduced.

Recommendation: Update the `require` statement to use `_enabledChainIds` instead of calling the external function:

```
- require(this.isChainSupported(chainId), InvalidChainId());
+ require(_enabledChainIds[uint8(chainId)], InvalidChainId());
```

BlendMoney: Fixed in PR 64.

Reviewer: Fix verified.

3.2 Informational

3.2.1 `AcrossXChainAdapter` has no ability to configure enabled chains after construction

Severity: Informational

Context: AcrossXChainAdapter.sol#L86-L88

Description: Enabled chains for `AcrossXChainAdapter` are currently configured during contract construction. After deployment, there is no mechanism to update this configuration (e.g., enabling or disabling specific chains). This lack of flexibility may cause issues if new chains need to be supported or if an existing chain becomes insecure or deprecated.

Recommendation: Introduce admin-controlled functions to manage the chain configuration post-deployment, for example:

- `enableChain(chainId)` to add support for a new chain.
- `disableChain(chainId)` to remove support for a chain.

BlendMoney: Acknowledged. We accept this. We want to avoid introducing keys and can redeploy and issue a new vaultconfig via the broadcaster if this changes.

Reviewer: Acknowledged.

3.2.2 Griefing attack on `SwapAdapter.swapToCollateral`

Severity: Informational

Context: SwapAdapter.sol#L79-L84

Description: Both `SwapAdapter.swapToCollateral()` and `SwapAdapter.swapToLoanToken()` use the `emptyAdapterBalances` modifier, which enforces that both the collateral token and loan token balances of the adapter are exactly zero after the swap.

```
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());
}
```

Since `SwapParams` are determined and passed in by the executor, it is likely that they rely on some calculations. An attacker could frontrun the transaction and deposit a minimal amount (e.g., 1 wei) of the token

into the adapter. This ensures the balance is non-zero at the end of execution, causing the transaction to revert even though the swap itself was correct.

Recommendation: Relax the post-condition by requiring only swapped out token balance to be zero for each function call, rather than enforcing both simultaneously.

BlendMoney: Acknowledged. We plan to deploy on a chain without mempool and also will use adapters that support `uint256.max` value as amount.

Reviewer: Acknowledged.

3.2.3 Slippage is checked on balance of safe instead of swapped amount

Severity: Informational

Context: SwapAdapter.sol#L113-L115

Description: The current slippage protection logic derives the minimum swap amount from the adapter's balance:

```
uint256 amount = collateralToken.balanceOf(address(this));
// ...
uint256 amountOutMin = PriceLib.applySlippage(
            PriceLib.quoteToBase(amount, strategyConfig.oracleCollateralPerLoan), swapParams.slippageBps
);
```

Since SwapParams are constructed by the executor before execution, an attacker can frontrun by depositing additional tokens into the adapter. This artificially inflates amount, which results in slippage checks being performed on an incorrect basis.

Recommendation: It's recommended to extract swap amount from the provided `swapParams.callData` and validate slippage based on that amount. This ensures slippage protection is based on the actual swap intent and not on potentially manipulated token balances.

BlendMoney: Acknowledged. We plan to deploy on a chain without mempool and also will use adapters that support `uint256.max` value as amount.

Reviewer: Acknowledged.