Zellic

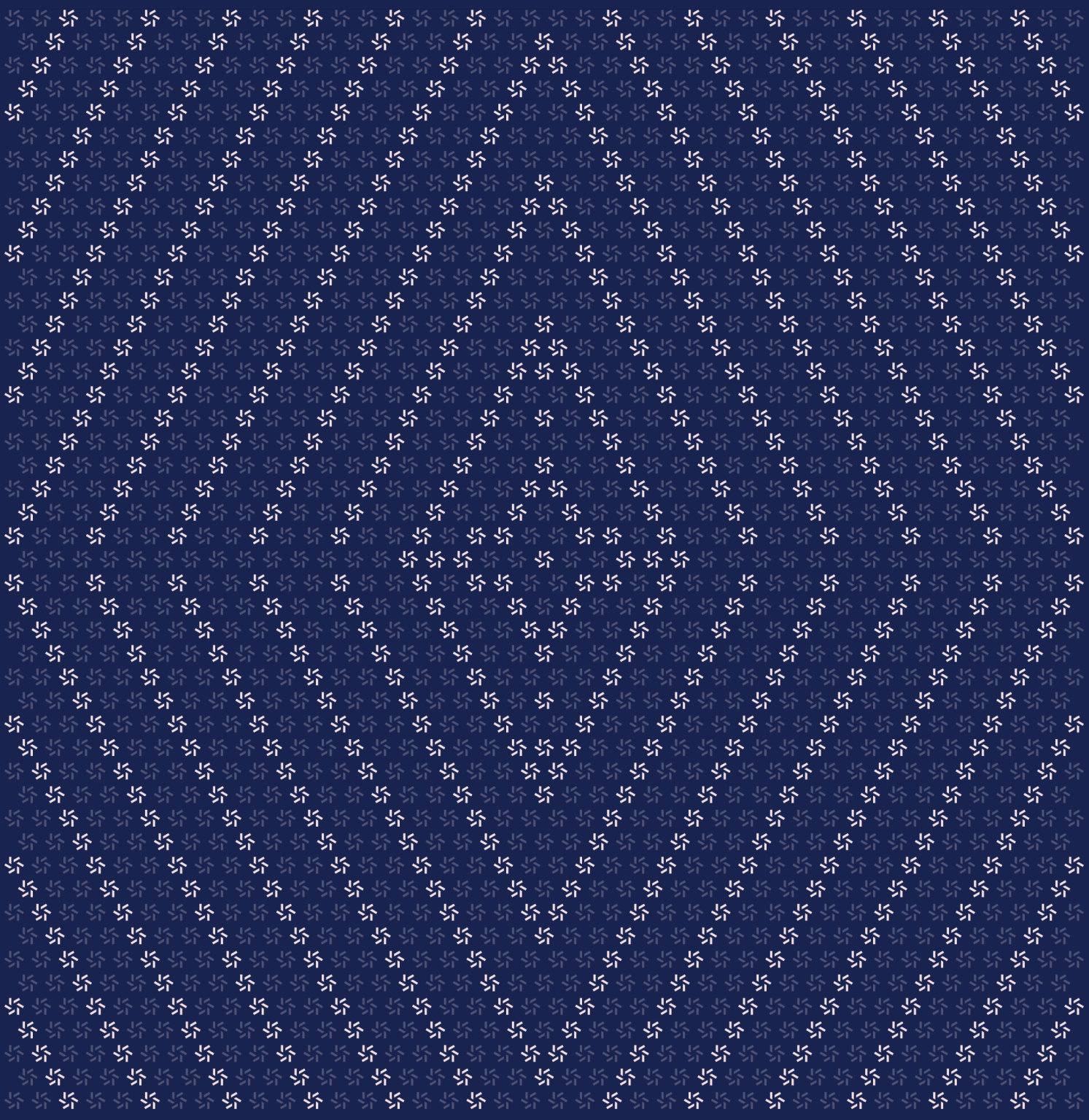
Prepared for
Haichen Shen
Scroll

Prepared by
Weipeng Lai
Chongyu Lv
Zellic

October 2, 2025

Blend Contracts

Smart Contract Security Assessment



Contents

About Zellic **3**

1. Overview **3**

1.1. Executive Summary 4

1.2. Goals of the Assessment 4

1.3. Non-goals and Limitations 4

1.4. Results 4

2. Introduction **5**

2.1. About Blend Contracts 6

2.2. Methodology 6

2.3. Scope 8

2.4. Project Overview 8

2.5. Project Timeline 9

3. Detailed Findings **9**

3.1. Missing configuration check in Constructor 10

4. System Design **11**

4.1. Component: WhitelistedSwapAdapter 12

5. Assessment Results **13**

5.1. Disclaimer 14

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Scroll from September 30th to October 1st, 2025. During this engagement, Zellic reviewed Blend Contracts's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Does PR #68 introduce any potential security vulnerabilities?
- Are temporal access controls correctly implemented?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Blend contracts, we discovered one finding, which was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	1

2. Introduction

2.1. About Blend Contracts

Scroll contributed the following description of the Blend contracts:

> A collection of smart contracts for Blend Money, built with Foundry. This repository contains the core contracts for Safe Guards, strategy management, cross-chain operations, and DeFi protocol adapters with comprehensive Morpho Blue integration.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect

its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Blend Contracts

Type	Solidity
Platform	EVM-compatible
Target	PR #68 up until commit 34b171931c7f9e707db74658a938184b7fdbd28e
Repository	https://github.com/BlendMoney/contracts ↗
Version	`34b171931c7f9e707db74658a938184b7fdbd28e`
Programs	`SwapAdapter.sol` `WhitelistedSwapAdapter.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 0.5 person-weeks. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

Pedro Moura
Engagement Manager
pedro@zellic.io

The following consultants were engaged to conduct the assessment:

Weipeng Lai
Engineer
weipeng.lai@zellic.io

Chongyu Lv
Engineer
chongyu@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 25, 2025	Kick-off call
September 30, 2025	Start of primary review period
October 1, 2025	End of primary review period

3. Detailed Findings

3.1. Missing configuration check in Constructor

Target	WhitelistedSwapAdapter		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

When both `ALLOW_ON_REBALANCE` and `ALLOW_ON_VAULT_ACTION` flags are false in the WhitelistedSwapAdapter contract, `inExecutionContext` will always fail, making `swapToCollateral` and `swapToLoanToken` unusable in any context.

```
modifier inExecutionContext() {
    // Check if at least one enabled operational state is active
    bool isAllowed = false;

    // If rebalancing is enabled, check if it's currently active
    if (ALLOW_ON_REBALANCE && STRATEGY_MANAGER.isRebalanceInitiated()) {
        isAllowed = true;
    }

    // If vault action is enabled, check if it's currently active
    if (ALLOW_ON_VAULT_ACTION && STRATEGY_MANAGER.isVaultActionInitiated()) {
        isAllowed = true;
    }

    // Block swap if neither enabled operational state is active
    require(isAllowed, Unauthorized());

    // Execute the function
    _;
}
```

Therefore, for the contract to be meaningful, at least one of `ALLOW_ON_REBALANCE` or `ALLOW_ON_VAULT_ACTION` must be true. The constructor should enforce this requirement.

Impact

When both `ALLOW_ON_REBALANCE` and `ALLOW_ON_VAULT_ACTION` are false, the contract becomes unusable.

Recommendations

We recommend requiring at least one of `ALLOW_ON_REBALANCE` or `ALLOW_ON_VAULT_ACTION` to be true in the constructor:

```
constructor(address _strategyManager, bool _allowOnRebalance,
    bool _allowOnVaultAction) SwapAdapter() {
    // [...]
    require(_allowOnRebalance || _allowOnVaultAction, "...");
    // Set permission flags that define when swaps are allowed
    ALLOW_ON_REBALANCE = _allowOnRebalance;
    ALLOW_ON_VAULT_ACTION = _allowOnVault_action;
}
```

Remediation

This issue has been acknowledged by Scroll, and a fix was implemented in commit 0c695f76 ↗.

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

4.1. Component: WhitelistedSwapAdapter

Description

The WhitelistedSwapAdapter contract, introduced in PR #68, extends SwapAdapter and wraps `swapToCollateral` and `swapToLoanToken` with a context-gating `inExecutionContext` modifier. The `inExecutionContext` modifier permits swaps only while `StrategyManager` is actively executing a rebalance or a vault action, controlled by immutable flags `ALLOW_ON_REBALANCE` and `ALLOW_ON_VAULT_ACTION`.

Invariants

Swaps execute only within allowed execution windows. This is enforced by `inExecutionContext`, which requires at least one of the following:

- `ALLOW_ON_REBALANCE && StrategyManager.isRebalanceInitiated()`
- `ALLOW_ON_VAULT_ACTION && StrategyManager.isVaultActionInitiated()`

Test coverage

- ☑ The `swapToCollateral` succeeds during the rebalance window when `ALLOW_ON_REBALANCE = true`.
- ☑ The `swapToLoanToken` succeeds during the rebalance window when `ALLOW_ON_REBALANCE = true`.
- ☑ The `swapToCollateral` succeeds during the vault-action window when `ALLOW_ON_VAULT_ACTION = true`.
- ☐ The `swapToLoanToken` succeeds during the vault-action window when `ALLOW_ON_VAULT_ACTION = true`.
- ☑ Swaps revert when called outside both windows.
- ☑ The constructor reverts on a zero `StrategyManager` address.

Attack surface

- The updates narrow callable timing to controlled execution windows, reducing the attack surface for unauthorized swaps.

- They introduce no new trust assumptions beyond those already present in SwapAdapter.

5. Assessment Results

During our assessment on the scoped Blend contracts, we discovered one finding, which was informational in nature.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.