Zellic

Prepared for
Bearlusconi
Dezzly
Berachain

Prepared by
Aaron Esau
Weipeng Lai
Zellic

September 8, 2025

BEND v2

Smart Contract Security Assessment

Contents

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 Berachain from September 8th to September 9th, 2025. During this engagement, Zellic reviewed BEND v2'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:

- Can an attacker leverage any input to steal funds due to changes introduced in the scope?
- Are access controls properly implemented?
- Are fees properly split when calculating and when minting them in the vault?

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 BEND v2 contracts, we discovered one finding, which was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Berachain in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	0
⬜ Informational	1

2. Introduction

2.1. About BEND v2

Berachain contributed the following description of BEND v2:

> BEND is the native lending and borrowing protocol of Berachain. BEND is based on Morpho v1.1 code. The main code changes regards how fees are managed.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

BEND v2 Contracts

Type	Solidity
Platform	EVM-compatible

Target	contracts-metamorpho-v1.1
Repository	https://github.com/berachain/contracts-metamorpho-v1.1 ↗
Version	`Only changes betwen ff447631...1aa3c467`

Programs	`MetaFeePartitioner.sol`
	`MetaMorphoV1_1.sol`
	`MetaMorphoV1_1Factory.sol`
	`libraries/EventsLib.sol`
	`utils/MetaFeePartitionerDeployer.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-days. 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:

Aaron Esau
Engineer
aaron@zellic.io ↗

Weipeng Lai
Engineer
weipeng.lai@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 8, 2025	Kick-off call
September 8, 2025	Start of primary review period
September 9, 2025	End of primary review period

3. Detailed Findings

3.1. Compromised MetaFeePartitioner owner could halt MetaMorphoV1_1 vaults

Target	MetaFeePartitioner		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The core MetaMorphoV1_1 vault operations — `deposit`, `mint`, `withdraw`, and `redeem` — all invoke `_accrueInterest()` before executing their main logic. Additionally, administrative functions `setFee()` and `setFeeRecipient()` also call `_accrueInterest()` to ensure fees are properly accrued before parameter changes.

Within `_accrueInterest()`, when `feeShares` is nonzero, the function calls `FEE_PARTITIONER.getShares()` to determine the fee split between the platform and the vault's fee recipient:

```
function _accrueInterest() internal {
    // [...]
    if (feeShares != 0) {
        (uint256 platformShare, uint256 recipientShare)
    = FEE_PARTITIONER.getShares(address(this), feeShares);
        // [...]
    }
    // [...]
}
```

The `FEE_PARTITIONER` address is immutable and set during contract deployment, creating a permanent dependency:

```
IMetaFeePartitioner internal immutable FEE_PARTITIONER;
```

Because MetaFeePartitioner is an upgradable contract (inheriting from UUPSUpgradeable), there is a centralization risk. If the MetaFeePartitioner owner's private key is compromised, an attacker could upgrade the contract to a malicious implementation where `getShares()` reverts. This would cause `_accrueInterest()` to revert whenever `feeShares != 0`, thereby blocking MetaMorphoV1_1 vault operations that require fee accrual.

Impact

If a wallet controlling MetaFeePartitioner ownership were compromised, an attacker could halt all MetaMorphoV1_1 vaults with nonzero accrued `feeShares` by upgrading MetaFeePartitioner to an implementation that reverts in `getShares`.

Recommendations

We recommend holding MetaFeePartitioner ownership in a timelocked multi-sig, enforcing upgrade delays, and monitoring upgrade proposals to detect and respond to malicious implementations before they take effect to reduce centralization risk.

Remediation

This issue has been acknowledged by Berachain.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Consider adopting two-step ownership transfer in MetaFeePartitioner

The MetaMorphoV1_1 contract uses a two-step ownership-transfer pattern via `Ownable2Step`.

```
contract MetaMorphoV1_1 is ERC4626, ERC20Permit, Ownable2Step, Multicall,
    IMetaMorphoV1_1StaticTyping {
```

By contrast, MetaFeePartitioner inherits OwnableUpgradeable and relies on a single-step transfer, which can mistakenly assign ownership to an address outside the protocol's control.

```
contract MetaFeePartitioner is IMetaFeePartitioner, Initializable,
    OwnableUpgradeable, UUPSUpgradeable {
```

For consistency and safer ownership handovers, we recommend adopting a two-step transfer pattern in MetaFeePartitioner by inheriting from Ownable2StepUpgradeable instead.

4.2. Consider adding a parameter for the previous value in the `FeePercentageSet` event

In the MetaFeePartitioner contract, `setDefaultPlatformFeePercentage` emits the `DefaultPlatformFeePercentageSet` event, which includes the previous (`old`) value.

```
function setDefaultPlatformFeePercentage(uint256 newFee) external onlyOwner {
    // [...]
    emit EventsLib.DefaultPlatformFeePercentageSet(old, newFee);
}
```

However, `setFeePercentage` emits `FeePercentageSet` without the previous value.

```
function setFeePercentage(address vault, uint256 newFee) external onlyOwner {
    // [...]
    emit EventsLib.FeePercentageSet(vault, newFee);
}
```

We recommend adding a parameter for the previous value to the `FeePercentageSet` event for consistency and improved observability.

5. 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.

5.1. MetaFeePartitioner

The scope includes a new contract, the MetaFeePartitioner, which is responsible for keeping track of what percentage of the fee should be allocated to the platform and what percentage should be allocated to the vault.

Its `getShares` function is the externally visible view function that returns the two shares, given the `fee` amount.

Since the setters are `onlyOwner` and access control is written correctly, the only security concern is centralization risk (see Finding 3.1. ↗). The UUPSUpgradeability pattern looks correct.

5.2. Minting shares

This is the code that actually distributes the shares by minting them, and it is safe. There are no reentrancy risks here, and MetaFeePartitioner guarantees that the sum of shares is less than (in the case of precision loss during share-splitting calculations) or equal to the original, unsplit fee.

5.3. Refactoring

The scope also incudes refactoring of access-control modifiers to call internal functions that perform the same behavior. The refactoring was correctly done.

Additionally, some events were added.

6. Assessment Results

During our assessment on the scoped BEND v2 contracts, we discovered one finding, which was informational in nature.

6.1. 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.