Zellic

Maia DAO V2 Ecosystem

Smart Contract Security Assessment

September 7, 2023

Prepared for:

Maia DAO

Prepared by:

Katerina Belotskaia and Ulrich Myhre

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Maia DAO from August 15th to August 17th, 2023. During this engagement, Zellic reviewed Maia DAO V2 Ecosystem's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 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:

- Could a malicious user manipulate balances or fees to empty the pool?
- Is the vault reentrancy check implemented correctly?
- Are reentry attacks possible for `optIn` and `optOut` functions?
- Can a distributed denial-of-service (DDOS) attack be conducted against the Flywheel?
- Are the calculations in the FlywheelBoosterGaugeWeight contract performed correctly?

1.2 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
- Balancer implementation

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

1.3 Results

During our assessment on the scoped Maia DAO V2 Ecosystem contracts, we discovered one finding, which was of high impact.

Breakdown of Finding Impacts

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

2 Introduction

2.1 About Maia DAO V2 Ecosystem

Maia DAO V2 Ecosystem consists of multiple protocols, one being Hermes V2, which allows for users to rent liquidity by allocating emissions to gauges by voting. Voting to a gauge makes users eligible for all bribes/fees allocated to that gauge. Bribing is permissionless, users are in charge of choosing what bribes they want to accrue per gauge.

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 code base 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, upgradeability 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:

Maia DAO V2 Ecosystem Contracts

Repositories	https://github.com/Maia-DAO/eco-c4-contest https://github.com/Maia-DAO/ComposableStablePoolWrapper
Versions	eco-c4-contest: `950b91a72978cf43d25e7927eefad8e44501e588` ComposableStablePoolWrapper: `3a3eb9689fd0c00e751c05ba8765d2e4c10e6104`
Programs	ComposableStablePoolWrapper VaultReentrancyLib ERC4626 FlywheelBoosterGaugeWeight
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of one person-week. The assessment was conducted over the course of one calendar week.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

Ulrich Myhre, Engineer
unblvr@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

August 15, 2023	Start of primary review period
August 18, 2023	End of primary review period
August 23, 2023	Draft report delivered

3 Detailed Findings

3.1 Flywheel index mismatch issue during `optOut`

- **Target**: FlywheelBoosterGaugeWeight
- **Category**: Coding Mistakes
- **Likelihood**: High
- **Severity**: High
- **Impact**: High

Description

The `optOut` function is called to stop accruing bribes for each gauge.

The `userGaugeFlywheels[msg.sender][strategy]` keeps the all `flywheel` addresses that were `optIn` for the `msg.sender` and `strategy` address. Also, the `userGaugeflywheelId[msg.sender][strategy][flywheel]` keeps the corresponding index of `flywheel` in the `userGaugeFlywheels` array.

During the `optOut` function execution, there is an optimization if the index of the `flywheel` address is not the last in the `userGaugeFlywheels[msg.sender][strategy]` array. To optimize the deletion of the `flywheel` address from `userGaugeFlywheels[msg.sender][strategy]`, this `flywheel` address will be rewritten by the last `flywheel` address and the last element will be popped from the array.

The problem is that `userGaugeflywheelId[msg.sender][strategy][flywheel]`, where `flywheel` is the address of the last address inside `userGaugeFlywheels[msg.sender][strategy]`, continues to store the index of the last element and is not updated to the new index of the moved address of the `flywheel`.

```
function optOut(ERC20 strategy, FlywheelCore flywheel) external {
    FlywheelCore[] storage bribeFlywheels
    = userGaugeFlywheels[msg.sender][strategy];

    uint256 userFlywheelId
    = userGaugeflywheelId[msg.sender][strategy][flywheel];

    if (userFlywheelId == 0) revert NotOptedIn();

    flywheel.accrue(strategy, msg.sender);

    flywheelStrategyGaugeWeight[strategy][flywheel]
    = flywheelStrategyGaugeWeight[strategy][flywheel]
```

```
        - bHermesGauges(owner()).getUserGaugeWeight(msg.sender,
    address(strategy));

    uint256 length = bribeFlywheels.length;
    if (length != userFlywheelId) bribeFlywheels[userFlywheelId - 1]
    = bribeFlywheels[bribeFlywheels.length - 1];

    bribeFlywheels.pop();
    userGaugeflywheelId[msg.sender][strategy][flywheel] = 0;
}
```

Impact

The function could experience a malfunction. And if invoked by a user in the future, it might result in the function being reverted, preventing the user from stopping the accrual of their tokens.

Recommendations

We recommend adding code that will update the index value to a new one.

```
    function optOut(ERC20 strategy, FlywheelCore flywheel) external {
    ...
        uint256 length = bribeFlywheels.length;
        if (length != userFlywheelId) bribeFlywheels[userFlywheelId - 1]
    = bribeFlywheels[bribeFlywheels.length - 1];
        userGaugeflywheelId[msg.sender][strategy][bribeFlywheels[length
            - 1]] = userFlywheelId;
    ...
    }
```

Remediation

This issue has been acknowledged by Maia DAO, and a fix was implemented in commit 607537d2.

4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1 Component: Composable Stable Pool Wrapper

Module flow

The module is a thin wrapper around the functionality that converts between shares and assets, using the Balancer pool's stable math logic. The module also gives protection against reentrancy while inside the view context, where state cannot be modified. The state being unmodifiable is accomplished by VaultReentrancyLib.sol, which tries to purposefully trigger the actual reentrancy guard, expecting it to immediately revert when the reentrancy flag is mutated in a (read-only) view context. From the length of the revert data, it is possible to differentiate between a revert due to actual reentrancy or an attempt to modify state in a view context.

4.2 Module: FlywheelBoosterGaugeWeight.sol

Function: `optIn(ERC20 strategy, FlywheelCore flywheel)`

1. Performs the check that `userGaugeflywheelId` is not already set for `msg.sender` and corresponds to the `strategy` and `flywheel` addresses.
2. Performs the check that the `strategy` address is trusted gauge.
3. Performs the check that the Flywheel contract was actually deployed over the `bribesFactory`.
4. Accrues rewards for the `msg.sender` on a strategy.
5. Increases the whole amount of `flywheelStrategyGaugeWeight[strategy][flywheel]` by the current user balance allocated to the strategy.
6. Adds the `flywheel` address to the array `userGaugeFlywheels[msg.sender][strategy]`.

7. Adds the index of `flywheel` from `userGaugeFlywheels` to the `userGaugeflywheelId[msg.sender][strategy][flywheel]`.

Inputs

- `strategy`
  - **Constraints**: The address should be a trusted gauge.
  - **Impact**: For this strategy address, the `boostedTotalSupply` value will be increased; further, this value will be used to accumulate global rewards on a strategy.
- `flywheel`
  - **Constraints**: The contract should be deployed over the `bribesFactory`.
  - **Impact**: The contract that manages token rewards. It distributes reward streams across various `strategies` and distributes them among the `users` of these strategies.

Branches and code coverage (including function calls)

Intended branches

- The `userGaugeflywheelId` != 0 after the call.
  ☑ Test coverage
- The `flywheelStrategyGaugeWeight` incremented.
  ☐ Test coverage

Negative behavior

- Double `optIn` for the same `strategy` and `flywheel`.
  ☑ Negative test
- The strategy is not a gauge.
  ☑ Negative test
- The untrusted Flywheel contract.
  ☑ Negative test

Function call analysis

- `flywheel.accrue(strategy, msg.sender)`
  - **What is controllable?** `flywheel` and `strategy`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If reverted, the user will not be able to `optIn` and the user's balance will not be able to take into account total supply.
- `bHermesGauges(owner()).getUserGaugeWeight(msg.sender, address(strategy))`

- **What is controllable?** `strategy`.
- **If return value controllable, how is it used and how can it go wrong?** Return the user's allocated weight to that gauge (strategy).
- **What happens if it reverts, reenters, or does other unusual control flow?** No problem — just view function.

Function: `optOut(ERC20 strategy, FlywheelCore flywheel)`

1. Performs the check that the `strategy` and `flywheel` addresses were `optIn` by `msg.sender`.

(b) Accrues rewards for the `msg.sender` on a strategy.

(c) Decreases the whole amount of `flywheelStrategyGaugeWeight[strategy][flywheel]` by the current user balance allocated to the strategy.

(d) Deletes the `flywheel` address from `userGaugeFlywheels[msg.sender][strategy]`.

(e) Deletes the index of the `flywheel` address from `userGaugeflywheelId[msg.sender][strategy][flywheel]`.

Inputs

- `strategy`
  - **Constraints**: The `userFlywheelId` should not be zero for the provided `strategy` and `flywheel`.
  - **Impact**: The strategy address for which the user will `optOut`, but only after the `optIn` call.
- `flywheel`
  - **Constraints**: The `userFlywheelId` should not be zero for the provided `strategy` and `flywheel`.
  - **Impact**: The `flywheel` address for which the user will `optOut`, but only after the `optIn` call.

Branches and code coverage (including function calls)

Intended branches

- The `userGaugeflywheelId == 0` after the call.
  - ☑ Test coverage
- The `flywheelStrategyGaugeWeight` decremented.
  - ☐ Test coverage

Negative behavior

- `msg.sender` did not `optIn` before for `strategy` and `flywheel`.
  ☑ Negative test
- `msg.sender` did not `optIn` before for `strategy`.
  ☑ Negative test
- `msg.sender` did not `optIn` before for `flywheel`.
  ☑ Negative test
- The case when `length ≠ userFlywheelId`.
  ☐ Negative test

Function call analysis

- `flywheel.accrue(strategy, msg.sender)`
  - **What is controllable?** `flywheel` and `strategy`.
  - **If return value controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If reverted, the user will not be able to `optIn` and the user's balance will not be able to take into account total supply.
- `bHermesGauges(owner()).getUserGaugeWeight(msg.sender, address(strategy))`
  - **What is controllable?** `strategy`.
  - **If return value controllable, how is it used and how can it go wrong?** Return the user's allocated weight to that gauge (strategy).
  - **What happens if it reverts, reenters, or does other unusual control flow?** No problem — just view function.

5 Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Maia DAO V2 Ecosystem contracts, we discovered one finding, which was of high impact. Maia DAO acknowledged the finding and implemented a fix.

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.