Zellic

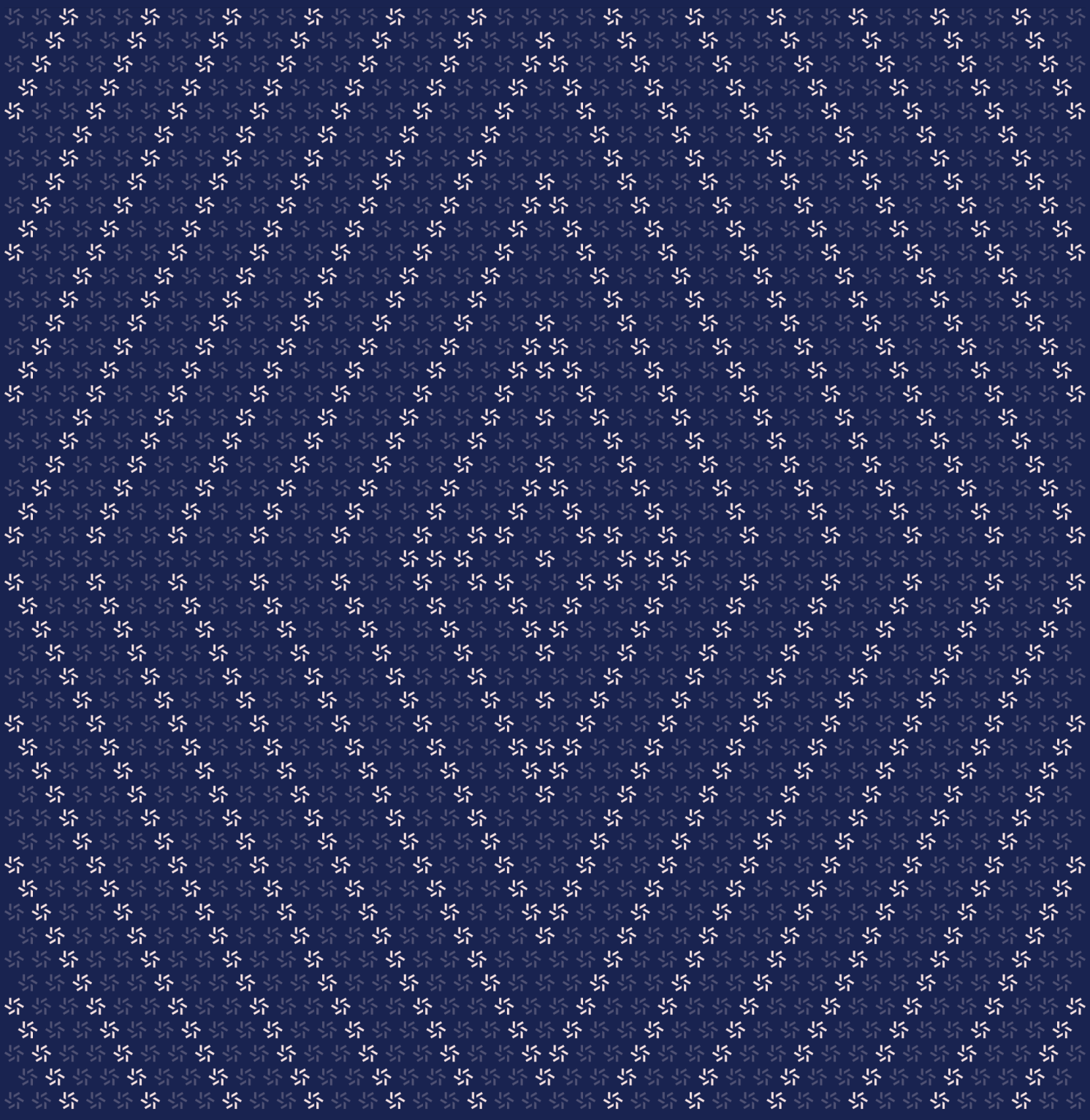
**Prepared for**
0xbuzzlightyear
0xbugsy
Maia DAO

**Prepared by**
Katerina Belotskaia
Ulrich Myhre
Zellic

**March 25, 2025**

# Partner Vault

## 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 Maia DAO from March 12th to March 19th, 2025. During this engagement, Zellic reviewed Partner Vault'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:

- Are there any vulnerabilities that could result in Partner Manager not being able to apply or clear any amount of utility tokens?
- Are access controls implemented effectively to prevent unauthorized operations?
- Is `_ENUMERABLE_ADDRESS_UINT92_MAP_SLOT_SEED` an appropriate value to avoid storage collisions?

## 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

The PartnerVault contract does not interact with users directly — all interactions happen through the PartnerManager contract, which is out of scope.

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

## 1.4. Results

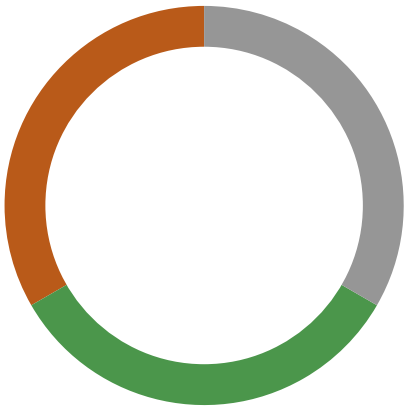
During our assessment on the scoped Partner Vault contracts, we discovered three findings. No critical issues were found. One finding was of high impact, one was of low impact, and the remaining finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Maia

DAO in the Discussion section (4. ↗).

### Breakdown of Finding Impacts

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



# 2. Introduction

## 2.1. About Maia DAO and Partner Vault

Maia DAO contributed the following description of Maia DAO:

> Maia DAO is an Omnichain Liquidity Rental platform that helps DeFi projects reduce incentive costs by providing a single, chain-agnostic venue for renting native liquidity, where projects only pay for the liquidity their users can use.

Maia DAO contributed the following description of Partner Vault:

> Smart contract vault that manages gauge voting, boost lending, and governance power for Burnt Hermes utility tokens held by a Hermes Partner Manager contract.

More information can be found here ↗.

## 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:

### Partner Vault Contracts

| | |
|---|---|
| **Type** | Solidity |
| **Platform** | EVM-compatible |
| **Target** | partner-vault |
| **Repository** | https://github.com/Maia-DAO/partner-vault ↗ |
| **Version** | `6d31e703332a87094cf71be24023a1e92c3dc99d` |
| **Programs** | `libs/EnumerableAddressToUint92MapLib.sol`<br>`libs/GaugeWeightVotingLib.sol`<br>`libs/UtilityVaultLib.sol`<br>`PartnerVault.sol` |

## 2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.8 person-weeks. The assessment was conducted by two consultants over the course of 1.2 calendar weeks.

### 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

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.

| | |
|---|---|
| **March 12, 2025** | Kick-off call |
| **March 12, 2025** | Start of primary review period |
| **March 19, 2025** | End of primary review period |

# 3. Detailed Findings

## 3.1. Value updates have no effect

| | | | |
|---|---|---|---|
| **Target** | EnumerableAddressToUint92MapLib.sol | | |
| **Category** | Coding Mistakes | **Severity** | High |
| **Likelihood** | High | **Impact** | High |

### Description

The EnumerableAddressToUint92MapLib library offers functions for retrieving, adding, updating, and removing key-value pairs from a set. Adding and updating has the combined function `addOrUpdate(bytes32 keyValuePair)` that will either add a key and value to the set or update the value if the key already exists. When an update happens, the previous value is also returned in the final result.

A small snippet of the function looks like this:

```
function addOrUpdate(bytes32 keyValuePair) internal returns (bytes32 result) {
    /// @solidity memory-safe-assembly
    assembly {
        let rootSlot := _ENUMERABLE_ADDRESS_UINT92_MAP_SLOT_SEED
        let key := shr(96, keyValuePair)
        if iszero(key) {
            mstore(0x00, 0xd5a0335e) // `KeyIsZero()`.
            revert(0x1c, 0x04)
        }
        for { let n := sload(not(rootSlot)) } 1 {} {
            mstore(0x20, rootSlot)
            if iszero(n) {
                result := sload(rootSlot)
                let v0 := shr(96, result)
                if iszero(v0) { // @audit: Wrong
                    sstore(rootSlot, keyValuePair)
                    break
                }
// ...
```

Here the function is reading from storage and checking if the `rootSlot` is zero, meaning that there are three or less values in the map. There are multiple branches in the code, depending on the number of existing elements in the set, and this is the special case where there are three values or fewer, and it has to check all the entries in order.

The mistake is that it only checks if the key in storage is zero (i.e., that it is empty and ready to insert

a value). It does not check if the stored key is equal to the key it is trying to update the value of.

## Impact

When attempting to update the value of an existing key, the value is effectively not updated. Calls to `manageVotingStrategies()` for updating existing gauges is noneffective. Additionally, the function for recalculating `totalShares` is marked as `unchecked` and could underflow.

```
function _addOrUpdateVotingStrategy(address gauge, uint256 share) private {
    uint256 prevShare = EnumerableAddressToUint92MapLib.getValue(
        EnumerableAddressToUint92MapLib.addOrUpdate(
            GaugeWeightVotingLib.joinKeyValue(gauge, share)
        )
    );
    /// @dev If value/share is the same, there is no change to be made.
    if (share == prevShare) revert GaugeAlreadyAddedWithSameShare();
    unchecked {
        /// @dev Each share is capped at 1 ether, so totalShares can't
    overflow.
        totalShares = totalShares - prevShare + share;
    }
}
```

If a new gauge is added with 100 shares, `totalShares` will increase by 100 as `prevShare` is 0. From this point on, the `prevShare` for this gauge will always be 100, despite updating the value. If the same gauge is added again with 10 shares, `totalShares` will be set to `100 - 100 + 10 = 10`. But if a third value 1 is set, `totalShares` will be set to `10 - 100 + 1 = -89`. This underflows and becomes a number close to $2^{256}$.

The latter problem is unlikely to occur by mistake, as it involves setting a gauge value multiple times in the same call.

## Recommendations

There is a workaround for the problem where the value can be removed and then added again. Due to this, the contract is not completely bricked and will suffer from much higher gas costs for normal operations. Our recommendation is to change the assembly code to check if the key is zero *or* equal to the insertion key.

Alternatively, using an existing library for this and not rolling a custom one would mean knowing about changes and fixes in that library as time goes on.

### Remediation

This issue has been acknowledged by Maia DAO, and a fix was implemented in commit a910b6ab ↗.

## 3.2. Missing bounds checks

| | | | |
|---|---|---|---|
| **Target** | EnumerableAddressToUint92MapLib.sol | | |
| **Category** | Coding Mistakes | **Severity** | Low |
| **Likelihood** | Low | **Impact** | Low |

### Description

Values in the EnumerableAddressToUint92MapLib library cannot exceed $2^{96}$, and doing so would lead to corrupting the key-value pair, because these are ORed together.

```
function joinKeyValue(address key, uint256 value)
    internal pure returns (bytes32 keyValuePair) {
     /// @solidity memory-safe-assembly
     assembly {
         if lt(value, MIN_SHARES_PER_GAUGE) {
             mstore(0x00, 0x630da3c4) // `BellowMinShares()`.
             revert(0x1c, 0x04)
         }
         if gt(value, MAX_SHARES_PER_GAUGE) {
             mstore(0x00, 0x509c2511) // `ExceedsMaxShares()`.
             revert(0x1c, 0x04)
         }

         keyValuePair := or(shl(96, key), value)
     }
 }
```

Inside the library, the values are shifted around such that only 96 bits remain, but there is a comment stating that zero values lead to undefined behavior. Values above $2^{96}$ could be equal to zero modulo $2^{96}$.

There are currently no explicit checks to limit the size of the incoming values. Instead, these are limited to be within `MIN_SHARES_PER_GAUGE` and `MAX_SHARES_PER_GAUGE`, which are currently set to 100 and $10^{18}$. While these values fall within the 96-bits-maximum limit, there are no comments, documentation, tests, or other indicators that these constants must be within a certain range. Looking at EnumerableAddressToUint92MapLib as a generic library, this could be a problem in the future if it is reused with different constants and the deployer is not aware of this requirement.

### Impact

If a value is added with a value larger than $2^{96}$, `joinKeyValue()` relies on the value of `MAX_SHARES_PER_GAUGE` being less than 96 bits; otherwise, the value can be corrupted.

### Recommendations

Document and/or create a test for the invalid behavior, making sure that the constants are never accidentally changed in a way that allows too large values.

### Remediation

This issue has been acknowledged by Maia DAO, and a fix was implemented in commit `047328bf` ↗.

### 3.3. Multiple spelling mistakes

| | | | |
|---|---|---|---|
| **Target** | GaugeWeightVotingLib.sol | | |
| **Category** | Coding Mistakes | **Severity** | Informational |
| **Likelihood** | N/A | **Impact** | Informational |

#### Description

Spelling mistakes in constants make functionality difficult to understand and also affect the hashes of errors and storage locations forever, so they should be changed early. We found several:

1. EnumerableAddressToUint92MapLib should be named EnumerableAddressToUint96MapLib, and its constant should be `_ENUMERABLE_ADDRESS_UINT96_MAP_SLOT_SEED`, changing the value of the constant. All comments and mentions of `uint92` should be updated.
2. The error `BellowMinShares` should be `BelowMinShares`, changing the `0x630da3c4` constant in `joinKeyValue`.

#### Impact

Spelling mistakes in constants are permanent and, for libraries especially, confuse users when the internal data type does not match the outside name.

#### Recommendations

Rename and regenerate the constants after fixing the spelling mistakes.

#### Remediation

This issue has been acknowledged by Maia DAO, and a fix was implemented in commit 28c3adde ↗.

# 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. Voting-strategy–management behavior

During the execution of the `manageVotingStrategies` function — which is used to update the percentage of votes distributed among specified gauges — the internal functions `_removeVotingStrategy` and `_addOrUpdateVotingStrategy` are called. Access to `manageVotingStrategies` is restricted to the contract owner and accounts with the `_ROLE_MANAGE_VOTING` role.

```
function _removeVotingStrategy(address gauge) private {
    uint256 share = EnumerableAddressToUint92MapLib.getValue(
            [...]
    );
    /// @dev If value/share is zero, then the gauge was not added.
    if (share == 0) revert GaugeNotAdded();
    [...]
}

function _addOrUpdateVotingStrategy(address gauge, uint256 share) private {
    uint256 prevShare = EnumerableAddressToUint92MapLib.getValue(
    [...]
    );
    /// @dev If value/share is the same, there is no change to be made.
    if (share == prevShare) revert GaugeAlreadyAddedWithSameShare();
    [...]
}
```

The `_removeVotingStrategy` and `_addOrUpdateVotingStrategy` functions have verifications that can cause the `manageVotingStrategies` function call to revert in specific cases:

- If a gauge is provided for removal but does not currently exist
- If a gauge is being updated with a shares value equal to the existing value

This behavior introduces a requirement for the caller to have precise knowledge of the current state of all gauges before preparing the input data for the `manageVotingStrategies` function.

As a result, while this strict behavior helps prevent unintended or redundant changes, this also leads to several usability concerns. Requiring the caller to perfectly know the current state makes integration harder. Also, if one proposal changes the gauge set, any subsequent proposal that does not perfectly account for the new state will revert.

We recommend silently skipping changes where the share value is unchanged or the gauge has

already been removed, while still processing the rest of the list. Additionally, events could be emitted to track all changes to the gauges list.

This issue has been acknowledged by Maia DAO, and a fix was implemented in commit 64189e7a ↗.

## 4.2. Test suite

When building a complex contract ecosystem with multiple moving parts and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

The test coverage for this project should be expanded to include all contracts, not just surface-level functions. More specifically, the following tests are currently missing and should be implemented to ensure complete coverage and correctness of critical functionalities:

- A test to ensure that the function `applyAll` works as expected
- Tests for partial weight clearing (`clearWeight`), ensuring that the new weights for gauges are calculated as expected, including all relevant verifications
- A test to ensure that the function `decrementGaugesBoostIndexed` works as expected
- Tests that verify the update of gauge shares when voting strategies are modified (`manageVotingStrategies`)
- Tests to ensure that all sensitive functions are properly protected and only callable by authorized roles

Therefore, we recommend building a rigorous test suite that includes all contracts to ensure that the system operates securely and as intended.

Good test coverage has multiple effects.

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).
- It displays code maturity.
- It bolsters customer trust in your product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point seems contradictory, given the time investment to create and maintain tests. To expand upon that, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks something if there are no tests. This is especially true for new developers or those returning to the code after a prolonged absence. Tests have your back here. They are an indicator that the existing functionality *most likely* was not broken by your change to the code.

# 5. 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.

## 5.1. Module: PartnerVault.sol

### Function: `allowAddressBoostAggregator(address user)`

This function allows the contract owner or an account with the `_ROLE_MANAGE_BOOST_ALLOWLIST` role to execute the `addAllowlistedAddress` function of the `boostAggregator` contract. To perform this operation, the current contract must be the owner of the `boostAggregator`.

#### Inputs

- `user`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The `allowlistedAddresses` flag for the specified user is set to `true`. The user will be permitted to transfer NFTs to the `boostAggregator`.

#### Branches and code coverage

**Intended branches**

- `allowlistedAddresses` is set to `true` for the specified address.
  - ☑ Test coverage

**Negative behavior**

- The caller is not the contract owner.
  - ☐ Negative test
- The caller is not an account with the `_ROLE_MANAGE_BOOST_ALLOWLIST` role.
  - ☐ Negative test

#### Function call analysis

- `this.boostAggregator.addAllowlistedAddress(user)`

- **What is controllable?** `user`.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** The function does not revert if `allowlistedAddresses` is already `true`, meaning redundant calls will have no effect but will still execute successfully.

### Function: `applyAll()`

This function applies all available gauge boosts, gauge weights, and governance tokens. It can be called by any user. The PartnerVault contract invokes this function during migration to a new PartnerVault contract to reapply all previously cleared tokens.

All gauge boosts are transferred from the `partnerManager` to the `boostAggregator`. All gauge weights are transferred from the `partnerManager` to this contract and delegated to itself. The weight for the current gauges is updated based on the `shares` state. Governance tokens are transferred from the `partnerManager` to this contract and delegated to the `governanceVault`.

### Branches and code coverage

**Intended branches**

- The boost apply has been completed as expected.
  - ☐ Test coverage
- The weight apply has been completed as expected.
  - ☐ Test coverage
- The governance apply has been completed as expected.
  - ☐ Test coverage

### Function: `applyBoost()`

This function allows any caller to transfer all available `gaugeBoost` tokens from the `partnerManager` contract to the `boostAggregator`.

### Branches and code coverage

**Intended branches**

- `gaugeBoost` tokens have been applied successfully.
  - ☑ Test coverage

**Negative behavior**

- There are no available tokens to apply.

  ☑ Negative test

### Function call analysis

- `this._applyBoost() -> UtilityVaultLib.applyUtilityTransfer(address(this.boostAggregator), address(this.partnerManager), address(this.gaugeBoost)) -> SafeTransferLib.safeTransferAllFrom(utilityToken, partner, vault);`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because the trusted `gaugeBoost` contract is called.

## Function: `applyGovernance()`

This function allows applying all available `governance` tokens. The `governance` tokens from the `partnerManager` contract will be transferred to the current contract and delegated to the `governanceVault`.

### Branches and code coverage

**Intended branches**

- The expected amount of `governance` tokens has been delegated to the `governanceVault`.

  ☑ Test coverage

**Negative behavior**

- The `partnerManager` does not have any available `governance` tokens in its balance.

  ☑ Negative test

### Function call analysis

- `this._applyGovernance() -> UtilityVaultLib.applyUtilityDelegation(address(this.governanceVault), address(this.partnerManager), address(this.governance)) -> SafeTransferLib.safeTransferAllFrom(utilityToken, partner, address(this))`

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because this function transfers the full balance instead of accepting a specific amount and the called `governance` contract is trusted.

- `this._applyGovernance() -> UtilityVaultLib.applyUtilityDelegation(address(this.governanceVault), address(this.partnerManager), address(this.governance)) -> ERC20MultiVotes(utilityToken).freeVotes(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the free `governance` tokens that can be delegated to the `governanceVault`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here since this is a view function.
- `this._applyGovernance() -> UtilityVaultLib.applyUtilityDelegation(address(this.governanceVault), address(this.partnerManager), address(this.governance)) -> ERC20MultiVotes(utilityToken).incrementDelegation(vault, utilityAvailable)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `maxDelegates` has been reached and `canContractExceedMaxDelegates` is not enabled for this contract.

### Function: `applyVotingStrategies()`

This function allows delegating and updating all currently available weights. If shares have not been provided yet or the current weight is zero, the function does nothing and simply returns without any state changes. If fee votes are available, they will be delegated to this contract itself, and the weights for the current gauge list will be recalculated using the new weight.

## Branches and code coverage

**Intended branches**

- `totalShares` is zero.
  - ☐ Test coverage

- `totalWeight` is zero.
  - ☐ Test coverage
- `freeWeight` is greater than zero and has been delegated successfully.
  - ☐ Test coverage
- Weights have been updated as expected.
  - ☐ Test coverage

### Function call analysis

- `SafeTransferLib.balanceOf(address(this.gaugeWeight), address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the full current `gaugeWeight` balance of the contract.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems because the `applyVotingStrategies` function has a `nonReentrant` modifier and the called `balanceOf` function is a view function.
- `this.gaugeWeight.freeVotes(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current contract balance of votes, excluding delegated votes.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because the `applyVotingStrategies` function has a `nonReentrant` modifier and the called `freeVotes` function is a view function.
- `this.gaugeWeight.incrementDelegation(address(this), freeWeight)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** It reverts if the number of delegates for this delegator exceeds `maxDelegates`, but this is not possible in this case since the contract has only one delegatee — this contract itself.
- `GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight)-> gaugeWeight.getUserGaugeWeight(address(this), gauge)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current weight for the `gauge`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because this call simply retrieves the amount

from a mapping.

- `GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight) -> gaugeWeight.decrementGauges(gaugesToDecrement, amountsToDecrement)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `flywheelBooster.accrueBribesNegativeDelta` function can revert if the `flywheelBooster` address has been updated before this call, but this depends on the `flywheelBooster` implementation.
- `GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight) -> gaugeWeight.incrementGauges(gaugesToIncrement, amountsToIncrement)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** It reverts if the contract reaches the gauge limit and `canContractExceedMaxGauges` is not enabled for this contract.

### Function: `applyWeight()`

This function allows applying free weight to the gauges. If `shares` have not been provided using `manageVotingStrategies` by the owner or `_ROLE_MANAGE_VOTING` yet, it is not possible to apply the free weight, and the function simply returns.

## Branches and code coverage

**Intended branches**

- `freeWeight` is zero.
  - ☑ Test coverage
- `totalShares` is zero.
  - ☑ Test coverage
- If `freeWeight` and `totalShares` are not zero, the weight for the gauges is applied as expected.
  - ☑ Test coverage

**Negative behavior**

- The current `freeWeight` amount is too small to be distributed among the sensors.

- ☐ Negative test

### Function call analysis

- `this._applyWeight() -> UtilityVaultLib.applyUtilityDelegation(address(this), address(this.partnerManager), address(this.gaugeWeight)) -> SafeTransferLib.safeTransferAllFrom(utilityToken, partner, address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because this function transfers the full balance instead of accepting a specific amount and the `applyWeight` function has a `nonReentrant` modifier.
- `this._applyWeight() -> UtilityVaultLib.applyUtilityDelegation(address(this), address(this.partnerManager), address(this.gaugeWeight)) -> ERC20MultiVotes(utilityToken).freeVotes(address(this));`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** The returned value is indirectly controlled by the user, as it depends on the number of tokens the user transfers to the `partnerManager` contract through the `forfeitWeight` function. Consequently, the weight applied to the gauges is influenced by how many tokens are transferred from the `partnerManager` to this contract.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because the called `freeVotes` function is a view function and the `applyWeight` function has a `nonReentrant` modifier.
- `this._applyWeight() -> UtilityVaultLib.applyUtilityDelegation(address(this), address(this.partnerManager), address(this.gaugeWeight)) -> ERC20MultiVotes(utilityToken).incrementDelegation(vault, utilityAvailable)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** The function reverts if the number of delegates for this delegator exceeds `maxDelegates`, but this is not a concern in this case since the contract has only one delegatee — this contract itself.
- `this._applyWeight() -> GaugeWeightVotingLib.increaseWeights(this.gaugeWeight, this.totalShares,`

```
freeWeight) -> gaugeWeight.incrementGauges(gaugeList,
newWeightAllocations)
```

- **What is controllable?** N/A.
- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** The function reverts if this contract reaches the gauge limit and is not `canContractExceedMaxGauges`, but in this case, no new gauges are added.

### Function: `clearAll()`

This function allows clearing all previously applied `gaugeBoost`, `gaugeWeight`, and `governance` tokens. Only the `partnerManager` can initiate this process, typically during migration to a new PartnerVault contract.

To execute this,

- all `gaugeBoost` tokens will be withdrawn from the `boostAggregator` and transferred to the `partnerManager`,
- all `gaugeWeight` tokens will be undelegated from the contract itself and transferred to the `partnerManager`, and
- all `governance` tokens will be undelegated from the `governanceVault` and transferred to the `partnerManager`.

### Branches and code coverage

**Intended branches**

- The `boost` tokens have been cleared as expected.
  - ☑ Test coverage
- The `weight` tokens have been cleared as expected.
  - ☑ Test coverage
- The `governance` tokens have been cleared as expected.
  - ☐ Test coverage

**Negative behavior**

- The caller is not `partnerManager`.
  - ☐ Negative test

### Function: `clearGovernance(uint256 amount)`

This function allows the caller to undelegate `governanceVault` tokens from the `governanceVault` and transfer them to the `partnerManager`. This function is available only for the PartnerManager contract. If the current amount of `freeVotes` is sufficient, tokens will be transferred without executing the `undelegate` function.

### Inputs

- `amount`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The specified amount of governance tokens will be transferred to the `partnerManager`.

### Branches and code coverage

#### Intended branches

- `freeVotes` > `amount`.
  - ☐ Test coverage
- `freeVotes` < `amount`.
  - ☑ Test coverage

#### Negative behavior

- The caller is not a `partnerManager`.
  - ☐ Negative test
- The requested `amount` exceeds the available token balance.
  - ☐ Negative test

### Function call analysis

- `this._clearGovernance(amount) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> ERC20MultiVotes(utilityToken).freeVotes(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current contract balance of votes, excluding delegated votes.

    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because `freeVotes` is a view function.
- `this._clearGovernance(amount) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> ERC20MultiVotes(utilityToken).undelegate(vault, amountToFree)`
    - **What is controllable?** `amountToFree` is partly controlled.
    - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `userUnusedVotes` is less than `amount`, meaning the requested amount of votes cannot be undelegated.
- `this._clearGovernance(amount) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> SafeTransferLib.safeTransfer(utilityToken, partner, amount);`
    - **What is controllable?** `amount` is partly controlled.
    - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the contract does not have enough tokens to transfer.

### Function: `clearWeight(uint256 amount)`

This function can only be called by the `partnerManager` contract. A user who wants to initiate the claiming of `gaugeWeight` tokens should call the `partnerManager.claimWeight()` function with the desired amount of `gaugeWeight` to be claimed.

If the available `gaugeWeight` is insufficient, the `partnerManager` calls this `clearWeight` function with the missing amount. If the amount of free votes is less than the provided `amount`, the remaining votes will be undelegated, and the full `amount` will be transferred to the `partnerManager`. After this, the `gaugeWeight` tokens will be transferred from the `partnerManager` to the user.

### Inputs

- `amount`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: The amount of `gaugeWeight` tokens to clear.

## Branches and code coverage

**Intended branches**

- Weights have been updated correctly in the case where the full weight has been cleared.
  - ☑ Test coverage
- Weights have been updated correctly in the case where the weight has been cleared partly.
  - ☐ Test coverage
- The necessary amount of votes have been undelegated.
  - ☑ Test coverage
- `freeVotes` > `amount`.
  - ☐ Test coverage
- `freeVotes` < `amount`.
  - ☑ Test coverage

**Negative behavior**

- The caller is not a `partnerManager`.
  - ☐ Negative test
- The `amount` exceeds the available token balance.
  - ☐ Negative test

## Function call analysis

- `this._clearWeight(amount) -> this.gaugeWeight.freeVotes(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current contract balance of votes, excluding delegated votes.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because `freeVotes` is a view function.
- `this._clearWeight(amount) -> GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight) -> gaugeWeight.getUserGaugeWeight(address(this), gauge)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current weight for the `gauge`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because this function simply retrieves values from a mapping.

- `this._clearWeight(amount) -> GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight) -> gaugeWeight.decrementGauges(gaugesToDecrement, amountsToDecrement)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** The `flywheelBooster.accrueBribesNegativeDelta` function may revert if the `flywheelBooster` address was updated before this call. This behavior depends on the `flywheelBooster` implementation.
- `this._clearWeight(amount) -> GaugeWeightVotingLib.updateWeights(this.gaugeWeight, this.totalShares, totalWeight) -> gaugeWeight.incrementGauges(gaugesToIncrement, amountsToIncrement)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the contract reaches the gauge limit and `canContractExceedMaxGauges` is not enabled for this contract.
- `this._clearWeight(amount) -> this.gaugeWeight.undelegate(address(this), amountToFree)`
    - **What is controllable?** `amountToFree` is partly controlled.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `userUnusedVotes` is less than `amount`, meaning the requested votes cannot be undelegated.
- `this._clearWeight(amount) -> SafeTransferLib.safeTransferFrom(address(this.gaugeWeight), address(this), address(this.partnerManager), amount)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the contract does not have enough tokens to transfer.

### Function: `collectBoostFees()`

This function allows any caller to execute `boostAggregator.withdrawProtocolFees`. The received `hermes` tokens will be deposited to `bHermes` if the `hermes` token address is not zero.

## Branches and code coverage

**Intended branches**

- The fee has been withdrawn successfully and deposited to `bHermes` if `hermes` is not zero.
  - ☑ Test coverage
- The fee has been withdrawn successfully and remains in the contract balance if `hermes` is zero.
  - ☑ Test coverage

**Negative behavior**

- No available fee for withdrawal.
  - ☐ Negative test

## Function call analysis

- `this._withdrawBoostFees() -> this.boostAggregator.withdrawProtocolFees(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Executes `uniswapV3Staker.claimReward(to, fees)`, which claims the entire remaining fee and transfers it to the current contract address.
- `SafeTransferLib.balanceOf(this.hermes, address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current `hermes` balance for this contract.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because `balanceOf` is a view function.
- `this.bHermes.deposit(amount, address(this.partnerManager))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `previewDeposit` for the provided amount returns zero, but this is unlikely.

### Function: `collectERC20(address token)`

This function allows any caller to transfer tokens from this contract's balance to the owner. However, the token address cannot be equal to the `gaugeWeight`, `governance`, or `hermes` token addresses.

### Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: Cannot be `gaugeWeight`, `governance`, or `hermes`.
  - **Impact**: The specified token is transferred from this contract to the owner.

### Branches and code coverage

**Intended branches**

- Tokens have been successfully transferred to the owner of this contract.
  - ☑ Test coverage

**Negative behavior**

- The token is equal to the `gaugeWeight` address.
  - ☑ Negative test
- The token is equal to the `governance` address.
  - ☑ Negative test
- The token is equal to the `hermes` address.
  - ☑ Negative test

### Function: `decrementGaugesBoostIndexed(uint256 offset, uint256 num)`

This function allows the owner or an account with the `_ROLE_MANAGE_BOOST_AGGREGATOR` role to execute the `decrementGaugesBoostIndexed` function of the `boostAggregator` contract. The current contract must be the owner of the `boostAggregator` to execute this function.

### Branches and code coverage

**Negative behavior**

- The caller is an owner or `_ROLE_MANAGE_BOOST_AGGREGATOR`.
  - ☐ Negative test

### Function call analysis

- `SafeTransferLib.balanceOf(address(this.gaugeBoost), address(this.boostAggregator))`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current balance of `gaugeBoost` tokens for the `boostAggregator`.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here since this is a view function.
- `this.boostAggregator.decrementGaugesBoostIndexed(balance, offset, num)`
    - **What is controllable?** `offset`, `num`
    - **If the return value is controllable, how is it used and how can it go wrong?** There is no return value here.
    - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.

## Function: `denyAddressBoostAggregator(address user)`

This function allows the owner or an account with the `_ROLE_MANAGE_BOOST_ALLOWLIST` role to execute the `removeAllowlistedAddress` function of the `boostAggregator` contract. The current contract must be the owner of the `boostAggregator` to execute this function.

### Inputs

- `user`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: `allowlistedAddresses` for the provided user will be reset to `false`. This address will no longer be able to transfer NFTs to the `boostAggregator`.

### Branches and code coverage

**Intended branches**

- `allowlistedAddresses` has been successfully deleted.
    - ☑ Test coverage

**Negative behavior**

- The caller is neither the owner nor has the `_ROLE_MANAGE_BOOST_ALLOWLIST` role.
    - ☐ Negative test

### Function call analysis

- `this.boostAggregator.removeAllowlistedAddress(user)`
  - **What is controllable?** `user`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here, but the function will not revert even if `allowlistedAddresses` is already `false`.

### Function: `manageVotingStrategies(address[] gauges, uint256[] shares)`

This function allows the owner or an account with the `_ROLE_MANAGE_VOTING` role to specify the proportions in which weight will be distributed among the provided gauges.

### Inputs

- `gauges`
  - **Control**: Full control.
  - **Constraints**: Should not be empty.
  - **Impact**: Defines the list of gauges to be added, updated, or removed.
- `shares`
  - **Control**: Full control.
  - **Constraints**: The length of `shares` must match the length of `gauges`.
  - **Impact**: Determines the weight allocation for each gauge in the provided `gauges` list.

### Branches and code coverage

#### Intended branches

- `gauges` has been updated as expected.
  - ☐ Test coverage
- `gauges` has been deleted as expected.
  - ☑ Test coverage
- `gauges` has been added as expected.
  - ☑ Test coverage

#### Negative behavior

- `gauges.length != shares.length`.
  - ☑ Negative test
- The `share` amount is less than `MIN_SHARES_PER_GAUGE`.
  - ☑ Negative test
- The `share` amount is greater than `MAX_SHARES_PER_GAUGE`.
  - ☑ Negative test
- `shares.length == 0`.
  - ☑ Negative test
- `gauges.length == 0`.
  - ☑ Negative test
- `gauges` contains the duplicate addresses.
  - ☐ Negative test
- `gauges` contains the duplicate addresses with equal `shares` amounts.
  - ☑ Negative test
- Attempts to remove a gauge that has not been added before.
  - ☑ Negative test
- The caller is the owner nor has the `_ROLE_MANAGE_VOTING` role.
  - ☐ Negative test

### Function call analysis

- `this._removeVotingStrategy(gauges[i]) -> EnumerableAddressToUint92MapLib.getValue(EnumerableAddressToUint92MapLib.remove(gauge));`
  - **What is controllable?** `gauge`.
  - **If the return value is controllable, how is it used and how can it go wrong?** If the returned amount is zero, the function will revert.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the gauge is not added.
- `this._removeVotingStrategy(gauges[i]) -> gaugeWeight.getUserGaugeWeight(address(this), gauge)`
  - **What is controllable?** `gauge`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current weight for the specified gauge. If the weight is zero, it will not be decremented for this gauge.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here because this call simply retrieves the amount from a mapping.

- `this._removeVotingStrategy(gauges[i]) -> this.gaugeWeight.decrementGauge(gauge, weight)`
    - **What is controllable?** `gauge`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the gauge is not trusted. Calls `flywheelBooster.accrueBribesNegativeDelta(user, ERC20(gauge), weight)`, which could execute arbitrary logic. But the `flywheelBooster` address has been modified only by the owner and considered trusted.
- `this._addOrUpdateVotingStrategy(gauges[i], share) -> EnumerableAddressToUint92MapLib.getValue(EnumerableAddressToUint92MapLib.addOrUpdate())`
    - **What is controllable?** `gauge` and `share`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current share amount for the specified gauge. If the new share value is the same as the existing value, the function will revert.
    - **What happens if it reverts, reenters or does other unusual control flow?** This function does not correctly update elements.
- `this._addOrUpdateVotingStrategy(gauges[i], share) -> EnumerableAddressToUint92MapLib.addOrUpdate(GaugeWeightVotingLib.joinKeyValue(gauge, share))`
    - **What is controllable?** `gauge` and `share`.
    - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the key is zero.

### Function: `optInVoteStrategyBribe(address gauge, address flywheel)`

This function allows the owner or an account with the `_ROLE_MANAGE_BRIBES` role to `optIn` to a provided `flywheel` for a specified `gauge`. The `_MAX_BRIBES_PER_GAUGE` limit must not have been reached.

### Inputs

- `gauge`
    - **Control**: Full control.
    - **Constraints**: No constraints.
    - **Impact**: The strategy address to `optIn`.
- `flywheel`

- **Control**: Full control.
- **Constraints**: No constraints.
- **Impact**: The `flywheel` address to `optIn`.

### Branches and code coverage

**Negative behavior**

- The caller is not an owner or `_ROLE_MANAGE_BRIBES`.
  - ☐ Negative test
- The `optIn` function has already been executed for the `gauge` and `flywheel`.
  - ☐ Negative test
- The `_MAX_BRIBES_PER_GAUGE` has been reached.
  - ☑ Negative test

### Function call analysis

- `flywheelBooster.getUserGaugeFlywheels(address(this), ERC20(gauge)`
  - **What is controllable?** `gauge`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current number of `userGaugeFlywheels` for this contract and the provided `gauge`.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here.
- `flywheelBooster.optIn(ERC20(gauge), FlywheelCore(flywheel))`
  - **What is controllable?** `gauge` and `flywheel`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the `gauge` already has `optIn` for this `flywheel`. It also reverts if the `gauge` is not a trusted one added by the owner of `bHermesGauges` or if the `flywheel` is invalid for the `bribesFactory`.

### Function: `optOutVoteStrategyBribe(address gauge, address flywheel, bool accrue)`

This function allows the owner or an account with the `_ROLE_MANAGE_BRIBES` role to `optOut` of a specified `flywheel` for a given `gauge`.

### Inputs

- `gauge`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The strategy address to `optOut`.
- `flywheel`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: The `flywheel` address to `optOut`.
- `accrue`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: Whether or not to accrue rewards before opting out.

### Branches and code coverage

**Negative behavior**

- The caller is an owner or `_ROLE_MANAGE_BRIBES`.
  - ☐ Negative test

### Function call analysis

- `this.flywheelBooster.optOut(ERC20(gauge), FlywheelCore(flywheel), accrue)`
  - **What is controllable?** `gauge`, `flywheel`, and `accrue`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `optIn` has not been executed for the provided `flywheel` beforehand.

### Function: `setProtocolFeeBoostAggregator(uint256 protocolFee)`

This function allows the owner or an account with the `_ROLE_MANAGE_BOOST_FEE` role to update the `protocolFee` in the `boostAggregator` contract. The current contract must be the owner of the `boostAggregator` to execute this function.

### Inputs

- `protocolFee`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: This fee will be charged during the execution of the `unstakeAndWithdraw` function of the `boostAggregator` contract.

### Branches and code coverage

#### Intended branches

- The `protocolFee` has been updated successfully.
  - ☑ Test coverage

#### Negative behavior

- The caller is an owner or `_ROLE_MANAGE_BOOST_FEE`.
  - ☐ Negative test
- The `protocolFee` is greater than `maxFee`.
  - ☐ Negative test

### Function call analysis

- `this.boostAggregator.setProtocolFee(protocolFee)`
  - **What is controllable?** `protocolFee`.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the new `protocolFee` exceeds `maxFee`.

### Function: `updateBoostAggregator(BoostAggregator _boostAggregator)`

This function allows the owner or an account with the `_ROLE_MANAGE_BOOST_AGGREGATOR` role to update the `boostAggregator` contract address. Before updating the address, boost fees will be withdrawn from the old `boostAggregator` and will remain on the contract balance until the next `collectBoostFees` or `collectERC20` execution. And all gauge boosts will also be withdrawn and transferred to the `partnerManager`. Additionally, the ownership of the old `boostAggregator` will be transferred to the current owner of this contract.

### Inputs

- `_boostAggregator`
  - **Control**: Full control.
  - **Constraints**: Owner of the new `_boostAggregator` should be this contract.
  - **Impact**: New `boostAggregator` contract address.

## Branches and code coverage

#### Intended branches

- The `boostAggregator` address has been updated successfully.
  - ☑ Test coverage

#### Negative behavior

- The caller is an owner or `_ROLE_MANAGE_BOOST_AGGREGATOR`.
  - ☐ Negative test

## Function call analysis

- `this._withdrawBoostFees() -> this.boostAggregator.withdrawProtocolFees(address(this)) -> boostAggregator.withdrawProtocolFees(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Executes `uniswapV3Staker.claimReward(to, fees)` to claim the entire remaining fee, which will be transferred to the current contract address.
- `this._emptyBoostAggregator() -> this.boostAggregator.withdrawAllGaugeBoost(address(this.partnerManager))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** It can run out of gas if the `gaugeList` is too large, because there is a loop over all user gauges in the `decrementAllGaugesAllBoost` function of the HermesGaugeBoost contract.
- `this.boostAggregator.transferOwnership(this.owner())`
  - **What is controllable?** N/A.

- **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
- **What happens if it reverts, reenters or does other unusual control flow?** There is no problem here.

### Function: `updateGovernanceVault(address _governanceVault)`

This function allows the owner or an account with the `_ROLE_MANAGE_GOVERNANCE_VAULT` role to update the `governanceVault` address, either by setting a new address or resetting it to zero. Before the update, all governance tokens will be undelegated from the old `governanceVault`, and all tokens will be transferred to the `partnerManager`.

#### Inputs

- `_governanceVault`
  - **Control**: Full control.
  - **Constraints**: No constraints.
  - **Impact**: New `governanceVault` contract address.

#### Branches and code coverage

**Intended branches**

- The `governanceVault` has been updated successfully.
  - ☑ Test coverage
- Votes have been undelegated successfully from the old vault.
  - ☐ Test coverage

**Negative behavior**

- The caller is not an owner or `_ROLE_MANAGE_GOVERANCE_VAULT`.
  - ☐ Negative test

#### Function call analysis

- `this._emptyGovernanceVault() -> SafeTransferLib.balanceOf(address(this.governance), address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current `governance` balance for this contract.

  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here since this is a view function.
- `this._emptyGovernanceVault() -> this._clearGovernance(SafeTransferLib.balanceOf(address(this.governance), address(this))) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> ERC20MultiVotes(utilityToken).freeVotes(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the current contract balance of votes, excluding delegated votes. Even if a user provides additional tokens directly to this contract, they will simply be treated as free tokens without any impact.
  - **What happens if it reverts, reenters or does other unusual control flow?** There are no problems here, because this a view function.
- `this._emptyGovernanceVault() -> this._clearGovernance(SafeTransferLib.balanceOf(address(this.governance), address(this))) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> ERC20MultiVotes(utilityToken).undelegate(vault, amountToFree)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if `userUnusedVotes` is less than `amount`, meaning the requested amount of votes cannot be undelegated.
- `this._emptyGovernanceVault() -> this._clearGovernance(SafeTransferLib.balanceOf(address(this.governance), address(this))) -> UtilityVaultLib.clearUtilityDelegation(this.governanceVault, address(this.partnerManager), address(this.governance), amount) -> SafeTransferLib.safeTransfer(utilityToken, partner, amount)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** This function does not return a value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts if the contract does not have enough tokens to transfer; however, that is not the case here, because the provided amount is set to the full balance of this contract.

# 6. Assessment Results

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

Before proceeding with mainnet deployment, we recommend expanding the current test suite to ensure comprehensive coverage and reduce potential risks in production. In particular, we recommend the following:

- Add more unit tests for the EnumerableAddressToUint92MapLib library to verify correct behavior under different conditions, including boundary cases.
- Ensure all functions are covered, even if they appear simple or straightforward, as this helps guard against unexpected side effects or future changes.
- Include dedicated access-control tests to confirm that only authorized roles can invoke restricted functions.

## 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.