SPEARBIT

Redacted Cartel SEAL Security Review

Auditors

Optimum, Lead Security Researcher

0x52, Lead Security Researcher

High Byte, Security Researcher

Sabnock, Junior Security Researcher

Maxime Viard, Junior Security Researcher

David Chaparro, Junior Security Researcher

Report prepared by: Pablo Misirov

July 5, 2023

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Critical Risk . . . 4
5.1.1 `RewardHarvester` - Tokens allocated as fees might be locked in the contract . . . 4
5.2 High Risk . . . 4
5.2.1 Centralization risks . . . 4
5.2.2 `BribeVault.transferBribes` - rewards might be locked in the contract in case `transferBribes` was called in the wrong timing . . . 5
5.2.3 `RewardHarvester.swapAndDepositReward` - Contract's operator can steal all contracts' tokens by utilizing arbitrary external calls . . . 5
5.3 Low Risk . . . 6
5.3.1 `updateRewardsMetadata` allows overwritting existings rewards . . . 6
5.3.2 Some checks are not enforced in `depositBribes` & `transferBribes` . . . 6
5.3.3 Usage of Maximum Allowance in `approve` Function of `ERC20Utils` Contract . . . 7
5.3.4 `Calls` to arbitrary destinations with bigger amount of gas than needed can lead to unexpected behaviors . . . 7
5.3.5 Single-step ownership change introduces risks . . . 7
5.3.6 Potential Mispricing Issue Due to Fee Calculation in Lower Claim Amounts . . . 8
5.3.7 Potential Overflow Vulnerability Due to Downcasting from `uint256` to `uint160` Without Sanity Checks . . . 8
5.3.8 Inconsistent fee denominators across contracts . . . 9
5.3.9 Vetting process for external (potentially untrusted) tokens used in the system . . . 9
5.4 Gas Optimization . . . 9
5.4.1 Redundant Event Emission in `grantDepositorRole` and `revokeDepositorRole` functions . . . 9
5.4.2 Array length read in each iteration of the loop wastes gas . . . 10
5.4.3 Loops counters can be gas optimized in some places . . . 10
5.4.4 Optimization of Off-Chain Data Handling in Smart Contract Code . . . 10
5.5 Informational . . . 11
5.5.1 `Swapper._externalCall()` can be simplified and calldata reduced . . . 11
5.5.2 Redundant logic can be simplified and improve gas cost . . . 11
5.5.3 `verifyCalldata` instead of `verify` for `MerkleProof.sol` . . . 11
5.5.4 Unnecessary checks for zero-length arrays . . . 12
5.5.5 Safety Checks Missing in `VaultOperator` Constructor . . . 12
5.5.6 Optimizing Joining/Leaving Logic and Events in `RewardHarvester` Contract . . . 12
5.5.7 Potential Incompatibility with `Fee-On-Transfer` or `Rebasing ERC20` Tokens in `swapAndDepositReward` Function . . . 13
5.5.8 Missing Inheritance of `IRewardDistributor`and `IBribeVault` Interfaces in Corresponding Contracts . . . 13
5.5.9 Unnecessary Check for briber Address in depositBribe Function of BribeVault Contract . . . 14
5.5.10 Set to default values is redundant . . . 14
5.5.11 Lack of variables at errors affects debugging and off-chain monitoring . . . 14
5.5.12 Unlocked pragma version may lead to unexpected behaviors . . . 15
5.5.13 Visibility `public` can be restricted when getters are used . . . 15
5.5.14 Code duplication affects maintainability and efficiency . . . 15
5.5.15 Unused named returns affects readability . . . 16

5.5.16 Missing events affect transparency and monitoring 16
5.5.17 Use of `immutable` Instead of `constant` for `FEE_BASIS` in `RewardHarvester` Contract 17
5.5.18 Potential Unchecked Array Lengths in `_swap` Function of Swapper Contract 17
5.5.19 Draft version is deprecated 17
5.5.20 Wrong or incomplete comments affects readability and maintenance 18
5.5.21 `RewardHarvester.depositReward`, `BribeVault.transferBribes` - events might contain the wrong value transferred 18
5.5.22 Unused code 19
5.5.23 Events without indexed event parameters make it harder/inefficient for off-chain tools 19
5.5.24 The system is heavily dependent on off-chain components 19

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Hidden Hand allows protocols to enable more efficient governance processes and to engage their voters, while users can earn extra yield on their favorite vote escrowed governance tokens through bribes.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of hidden-hand according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 10 days in total, Redacted Cartel engaged with Spearbit to review the hidden-hand protocol. In this period of time a total of **41** issues were found.

Summary

Project Name	Redacted Cartel
Repository	hidden-hand
Commit	30ce53...204b
Type of Project	Bribes Marketplace, DeFi
Audit Timeline	May 22 - June 2
Two week fix period	June 2 - June 16

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	1	1	0
High Risk	3	2	1
Medium Risk	0	0	0
Low Risk	9	4	5
Gas Optimizations	4	3	1
Informational	24	13	11
Total	**41**	**23**	**18**

5 Findings

5.1 Critical Risk

5.1.1 `RewardHarvester` - Tokens allocated as fees might be locked in the contract

Severity: Critical Risk

Context: RewardHarvester.sol#L131

Description: `RewardHarvester.claim` charges a fee for each user claim. This fee is allocated in the claimed token (denoted as `_token`) but is being accounted as `defaultToken` which may cause some of the fees to be locked in the probable case of multiple bribe tokens.

Recommendation: Consider replacing the line as showed in the snippet below

```
- feesCollected[defaultToken] += feeAmount;
+ feesCollected[_token] += feeAmount;
```

Redacted: Fixed. Commit c32e04.

Spearbit: Verified.

5.2 High Risk

5.2.1 Centralization risks

Severity: High Risk

Context: Hidden Hand Protocol

Description: During the course of the security review, we identified a critical concern regarding the centralization of the protocol. As an example, the function `emergencyWithdraw` allows a user with the `DEFAULT_ADMIN_ROLE` to withdraw any amount of any token from the contract at any time. This provides the admin with an alarming amount of control and power over the funds within the contract, significantly centralizing the protocol.

Other centralization issues detected:

- Centralization issue regarding fee changes dramatically (i.e: from 5% to 49%), so it affects all voters: setFee.
- Owner of `RewardHarvesterClaim` can front run claims / change fees to bigger values: changeFee.
- Operator has complete control of Merkle root and can maliciously set root to drain all funds in contract: updateRewardsMetadata.
- Operator can pause or resume rewards distribution at their discretion. A compromised or malicious operator can interrupt rewards distribution, particularly during volatile market conditions, negatively affecting the expected value of claims: setPauseState.

Recommendation: With a focus on decentralization and fair control over funds, there could be a great benefit in exploring various mechanisms to distribute the protocol's governance more evenly. This might include concepts such as role diversification, multi-signature permissions, time-locks, or other mechanisms that align with the spirit of decentralization and security. We advise careful exploration of these and other solutions that could serve to reduce the risk of single-point failures or misuse of power.

Redacted: Acknowledged. Partial changes were performed, specifically for `updateRewardsMetadata` where metadata can be readily updated. To minimize the risk of the compromised operator and/or invalid data, an internal timer is now set whenever a metadata update is performed (ie. new metadata is no longer instantly effective, and rewards claiming will be paused until after the set timestamp by the contract) so that the dao/multi-sig have the chance to react (ie. revoking the operator, pause claiming, and then update the metadata appropriately). For other owner-only methods, like `setPauseState`, no action will be taken as it's meant to be effective immediately.

Spearbit: Acknowledged.

5.2.2 `BribeVault.transferBribes` - rewards might be locked in the contract in case `transferBribes` was called in the wrong timing

Severity: High Risk

Context: BribeMarket.sol#L377, BribeMarket.sol#L405, BribeVault.sol#L280, BribeVault.sol#L387

Description: Anyone can deposit bribes to the `BribeVault` by calling `BribeMarket.depositBribe`. Each bribe is associated with a proposal, periods, token, and more parameters chosen by the user and the contract itself, for example, the proposal deadline which marks the last date to deposit bribes to a specific proposal. After the deposit phase was ended, the admin of `BribeVault` should call `transferBribes` which will transfer a specific set of bribes to the `RewardDistributor` contract (after fee deduction). The issue is that in case `transferBribes` was called before the defined deadline was reached, any future bribe deposits will be locked in the contract since `transferBribes` will revert with the `BribeAlreadyTransferred` error message. These funds might only be released by the admin centralized `emergencyWithdraw` function, which is definitely not desired.

A full example scenario:

1. Alice calls `depositBribe` for proposal A with token A with 100 tokens.
2. `transferBribes` was called before the deadline (could be maliciously or by accident).
3. Bob calls `depositBribe` (before the deadline) for proposal A with token A with 200 tokens. Both Alice and Bob will share the same `bribeIdentifier`, and also the same object in `bribes`, and the same reference in `rewardToBribes`.

The next time `transferBribes` will be called (for Bob), the transaction will revert with the `BribeAlreadyTransferred` error, and Bob's funds will be locked in the contract.

Recommendation: `transferBribes` should revert for calls made before the proposal deadline had arrived.

Redacted: Fixed. Commit fc8d1c.

Spearbit: Fixed by allowing multiple transfers of bribes (`transferBribes`) when there are new deposits that were not transferred yet.

5.2.3 `RewardHarvester.swapAndDepositReward` - Contract's operator can steal all contracts' tokens by utilizing arbitrary external calls

Severity: High Risk

Context: RewardHarvester.sol#L175, Swapper.sol#L18

Description: The function `RewardHarvester.swapAndDepositReward` provides the operator with the capability to deposit tokens different from the `defaultToken` into the `RewardHarvester` contract and subsequently perform a swap to the `defaultToken` within the same transaction. This functionality is facilitated by the `_swap` function, which allows the caller to invoke any desired contract with arbitrary data(function name and arguments), apart from `transferFrom`. The requirement is that by the end of the transaction, the balance of the default token should increase by the value specified in `_swapData.toAmount`, which is determined by the operator (the caller). However, it's important to note that this implementation has a potential security vulnerability. A malicious operator could exploit the arbitrary calls to drain all tokens or ether remaining in the contract by invoking `token.transfer` or `token.approve`, or swapping them for worthless tokens, among other possibilities. The only constraint is that one of the calls must actually perform a swap to the `defaultToken` with a small amount of tokens. In the case of the `RewardHarvester`, this issue becomes especially critical since it allows for the draining of all different tokens collected from the `RewardDistributor` contract.

Although we classified this issue as "High" rather than "Critical" due to an assumption that operators are trusted to some extent by the system, it's important to highlight that the impact of this vulnerability is significantly greater than that of other functions accessible only by the operator. This is mainly because once the attack occurs, it is irreversible, unlike the `updateRewardsMetadata` function, where a malicious operator who fails to invoke the function or provides incorrect arguments can be replaced by the contract owner.

Recommendation: The main purpose of the described function is to pull tokens from the operator, swap them to the default token and keep them in the `RewardHarvester` contract as a bribe. In the current code, the operator can spend any other tokens that reside in the contract, and thus isolation between the two is needed. We propose introducing a new utility contract that may implement a function very similar to `swapAndDepositReward` which will be only callable by the operator of `RewardHarvester` (can be achieved by storing a reference to the `RewardHarvester` contract). This function will be used for any arbitrary external calls as long as the swapped default tokens will be sent to the `RewardHarvester` contract by calling `RewardHarvester.depositReward` by the end of the function.

Redacted: Fixed, additional changes were applied, where an intermediary contract is used as the recipient of the standard rewards, which then proceed to perform the swaps+deposits into the harvester. Commit d3258b.

Spearbit: Fixed by removing `RewardHarvester.swapAndDepositReward` and introducing `RewardSwapper`; an intermediary contract to facilitate the isolated swaps.

5.3 Low Risk

5.3.1 `updateRewardsMetadata` allows overwritting existings rewards

Severity: Low Risk

Context: RewardDistributor.sol#L82

Description: The `updateRewardsMetadata` function currently allows `DEFAULT_ADMIN_ROLE` to overwrite existing reward distributions without any checks to confirm if a distribution already exists for a given identifier. `proofs`, `tokens`, or `merkleroot` itself can be overwritten. This could lead to the loss of critical data.

Recommendation: Before overwriting any reward data, add a check to verify if a reward distribution already exists for the identifier. If so, revert with a custom error.

Redacted: It is by design that the metadata can be overwritten whenever there's a new reward update for the specified identifier. Please note that the identifier is fixed for a protocol+token pair hash. Therefore, the metadata will always reflect the latest reward data for all users (cumulatively).

Spearbit: Acknowledged.

5.3.2 Some checks are not enforced in `depositBribes` & `transferBribes`

Severity: Low Risk

Context: BribeMarket.sol#L512, BribeVault.sol#L280, BribeVault.sol#L387

Description: The code should enforce that deadlines and other conditions are followed properly in all the pieces of the code regarding bribes. While we can see at _depositBribe have multiple checks about the proposal, token to be whitelist, periods... Those checks are not found at depositBribe, therefore, someone with the `DEPOSITOR_ROLE` can deposit bribes bypassing 3 checks: `if (proposalDeadline < block.timestamp) revert Errors.DeadlinePassed();` `if (_periods == 0 || _periods > maxPeriods) revert Errors.InvalidPeriod();` `if (!isWhitelistedToken(_token)) revert Errors.TokenNotWhitelisted();`

Additionally, `transferBribes` it's there to make the bribes flow work, this function is only callable by `DEFAULT_ADMIN_ROLE` and it's not enforced that `transferBribes` it is called on time, there isn't any type of check regarding the `proposalDeadline` to be finished.

Recommendation: Enforce the conditions performed at `depositBribe` and `transferBribes`.

Redacted: Acknowledged and partially addressed (for `transferBribes` indirectly, which now resolves the issue if the funds are transferred earlier than they should be).

Spearbit: Acknowledged.

5.3.3 Usage of Maximum Allowance in `approve` Function of `ERC20Utils` Contract

Severity: Low Risk

Context: ERC20Utils.sol#L84

Description: The `approve` function of the `ERC20Utils` contract sets the maximum possible allowance when the current allowance is less than the required amount. This usage of maximum allowance is generally not recommended as it could potentially amplify the impact of exploits.

Recommendation: Replace the usage of `MAX_UINT` with a more secure approach to increase the allowance. This could involve increasing the allowance only by the amount needed (i.e., `_amount`) to perform the transaction.

Redacted: Fixed. `ERC20Utils` is removed. Commit c32e04.

Spearbit: Verified.

5.3.4 `Call`s to arbitrary destinations with bigger amount of gas than needed can lead to unexpected behaviors

Severity: Low Risk

Context: ERC20Utils.sol#L89-L113

Description: At `transferTokens()`, a `call` is made to `_destination` address with a fixed gas limit of `10000`. The use of a hardcoded gas amount can present issues depending on the receiving contract's `fallback` function. If the receiving contract's logic requires more than `10000` gas to execute, the transaction will fail due to out-of-gas errors. This can restrict the usability of this function, particularly when interacting with complex contracts. If the function is just to do basic transfer, `2300` would probably be enough.

However, there is no clear use case for this function, therefore it's not even used. Precautions should be taken when using and setting gas values in a hardcoded way.

Recommendation: Document clearly what's the use case and which are the expected contract/functioning on the fallback so it doesn't use much more amount than needed. If indeed is just ETH transfers, lower gas (actually 2300 is used typically) can be set.

Redacted: Acknowledged. `ERC20Utils` is removed as most of it is now not in use with the latest refactoring.

Spearbit: Acknowledged.

5.3.5 Single-step ownership change introduces risks

Severity: Low Risk

Context: BribeFactory.sol#L10,RewardHarvester.sol#L13, RewardHarvesterClaim.sol#L9, VaultOperator.sol#L10

Description: Single-step ownership transfers add the risk of setting an unwanted owner by accident, if the ownership transfer is not done with excessive care it can be lost forever. Also, Open Zeppelin's Ownable library includes `renounceOwnership`, what it's unnecessary in most cases (such as this one) and discouraged to be included without being overwritten, as if called, ownership would be lost.

Recommendation: Employing 2 step ownership transfer mechanisms for this critical ownership change, such as Open Zeppelin's `Ownable2Step` or Synthetic's `Owned`, would reduce the risk significantly.

Redacted: Fixed. Using `Ownable2Step` from OpenZeppelin. Commit 71782e.

Spearbit: Fixed.

5.3.6 Potential Mispricing Issue Due to Fee Calculation in Lower Claim Amounts

Severity: Low Risk

Context: RewardHarvester.sol#L129

Description: In the `claim` function, there's a potential mispricing issue concerning fee calculation for claim amounts lower than `FEE_BASIS` (10,000). The following line of code :

```
uint256 feeAmount = (_amount * _fee) / FEE_BASIS;
```

could result in zero fees for any `_amount * _fee` less than the `FEE_BASIS`. Given that `_fee` is capped at 1,000 in `RewardHarvesterClaim`, all claim amounts from 0 to 999 (inclusive) would result in no fee, contrary to the intended functionality.

Recommendation: To address this potential issue, consider modifying the fee calculation to ensure that small claim amounts are not exempt from fees. This could involve altering the order of operations or using a different fee calculation model that scales more effectively with smaller amounts. If it's acceptable to have a small rounding discrepancy, we can alter the order of operations to ensure that fees are not waived for small claim amounts. Here is a suggestion for a new calculation:

```
uint256 feeAmount = (_amount * _fee + FEE_BASIS - 1) / FEE_BASIS;
```

The `+ FEE_BASIS - 1` term ensures that we round up instead of down when calculating the fee amount. It ensures that for all valid `_amount * _fee`, the fee is not zero. However, be aware that this modification slightly alters the fee calculation, possibly leading to slightly higher fees due to rounding up.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.3.7 Potential Overflow Vulnerability Due to Downcasting from `uint256` to `uint160` Without Sanity Checks

Severity: Low Risk

Context: BribeVault.sol#L264

Description: The `_transferWithPermit` function appears to have a potential vulnerability due to the downcasting of a `uint256` to a `uint160` without any prior sanity check. This flaw could allow a depositor to input a value that is greater than `type(uint160).max`. The contract would then register this deposit as the `uint160` representation of `_dp.amount`, which could be significantly different from the intended deposit value.

Moreover, by inputting a value equal to `type(uint160).max + 1`, the depositor could effectively bypass the zero-amount check

```
if (_dp.amount == 0) revert Errors.InvalidAmount();
```

because the overflow would cause the value to loop back to zero, hence avoiding the `InvalidAmount` error.

Recommendation: To mitigate this vulnerability, the code should include a sanity check before performing the downcasting operation. This check would ensure that `_dp.amount` is not greater than the maximum allowable value for `uint160`.

The check could be as follows:

```
if (_dp.amount > type(uint160).max) revert Errors.AmountOverflow();
```

Redacted: Fixed. Using `SafeCast` library for `uint256`. Commit 3b2473.

Spearbit: Verified.

5.3.8 Inconsistent fee denominators across contracts

Severity: Low Risk

Context: RewardHarvester.sol#L22 BribeVault.sol#L28

Description: Inconsistent fee denominators increase the operational risk of updating fees incorrectly due to mixing up fee denominators across contracts. As an example, a fee of `5_000` would be 0.5% with a denominator of `1_-000_000` while it would be 50% with a denominator of `10_000`, which can lead to a severe impact should it occur.

Recommendation: Use a consistent fee denominator across all system contracts

Redacted: Fixed. Using `1_000_000` for both fee denominators. Commit c32e04.

Spearbit: Fixed.

5.3.9 Vetting process for external (potentially untrusted) tokens used in the system

Severity: Low Risk

Description: The system relies on multiple external token contracts, which may be considered untrusted. It is essential to thoroughly evaluate these contracts for compatibility with the protocol. Bugs, operator privileges, non-standard implementations, and features such as blacklisting can significantly affect the protocol, leading to the loss or immobilization of funds.

Recommendation: Implement a robust vetting process to thoroughly review the potential risks associated with each external token contract. This should include the identification and evaluation of potential bugs, operator privileges, non-standard implementations, and features such as blacklisting. It would be prudent to tie this vetting process with governance decisions for improved risk mitigation.

Redacted: We currently have a standard governance process (also tied with the voting mechanic) before adding new tokens as rewards/bribes and thus would act as the vetting process.

Spearbit: Acknowledged.

5.4 Gas Optimization

5.4.1 Redundant Event Emission in `grantDepositorRole` and `revokeDepositorRole` functions

Severity: Gas Optimization

Context: BribeVault.sol#L109, BribeVault.sol#L122

Description: In the `grantDepositorRole` and `revokeDepositorRole` functions, there appear to be redundant event emissions. When a depositor role is granted or revoked, an event is emitted via the `_grantRole` or `_-revokeRole` methods. Subsequently, another event `GrantDepositorRole` or `RevokeDepositorRole` is emitted as well. Emitting two events for the same action not only makes the contract more gas-costly but can also lead to potential confusion or misinterpretation during event handling or log reading.

Recommendation: To improve gas efficiency and reduce potential confusion, consider removing the secondary `GrantDepositorRole` and `RevokeDepositorRole` events from `grantDepositorRole` and `revokeDepositorRole` functions.

Redacted: Fixed. Removed the `GrantDepositorRole` and `RevokeDepositorRole` events. Commit a828ad.

Spearbit: Verified.

5.4.2 Array length read in each iteration of the loop wastes gas

Severity: Gas Optimization

Context: RewardDistributor.sol#L60, RewardDistributor.sol#L79, BribeMarket.sol#L271, BribeMarket.sol#L295, BribeMarket.sol#L323, BribeMarket.sol#L348, BribeVault.sol#L375, BribeVault.sol#L393

Description: If not cached, the Solidity compiler will always read the length of the array during each iteration. For a memory array, it implies extra `MLOAD` operations (3 additional gas for each iteration beyond the first). There is one instance of the array length being cached (Swapper.sol#L36).

Recommendation: Cache the array length outside of the loop and use that variable in the loop.

```
+ uint256 _claimsLength = _claims.length;
- for (uint256 i; i < _claims.length;) {
+ for (uint256 i; i < _claimsLength;) {
...
 }
```

Redacted: Fixed. Applied recommendation on all our for loops. Commit 222abe.

Spearbit: Fixed.

5.4.3 Loops counters can be gas optimized in some places

Severity: Gas Optimization

Context: RewardDistributor.sol#L60, RewardDistributor.sol#L79, BribeMarket.sol#L271, BribeMarket.sol#L295, BribeMarket.sol#L348, BribeVault.sol#L375, BribeVault.sol#L393, BribeMarket.sol#L145, BribeMarket.sol#L173, BribeMarket.sol#L202

Description: Loop counters are optimized in many parts of the code by using an `unchecked {++i}` (unchecked + prefix increment). However, this is not done in some places where it is safe to do so.

Recommendation: Use the following style for loop counter increments in all places consistently:

```
for (uint256 i; i < cachedArrayLength;) {
    ...
    unchecked {
        ++i;
    }
}
```

Redacted: Fixed. Applied recommendation. Commit 094fa9e and 222abe.

Spearbit: Fixed.

5.4.4 Optimization of Off-Chain Data Handling in Smart Contract Code

Severity: Gas Optimization

Context: RewardHarvester.sol#L215, BribeVault.sol#L421, RewardHarvester.sol#L91-L104

Description: The smart contract code under review contains several instances where state variables are being used primarily for off-chain purposes. These variables include:

- `reward.hashedData`
- `r.distributorAmountReceived`
- `isMember`

State variables are stored on-chain and are thus more expensive to use in terms of gas costs.

Recommendation: It is recommended to replace the use of state variables for off-chain purposes with events only. Events are cheaper in terms of gas costs and are specifically designed for listening by the front-end of a

DApp or other off-chain services. They provide a way to trigger functionality outside the Ethereum blockchain and can be used to log data from smart contracts. This change will not only optimize the contract in terms of gas costs but also improve the overall readability and maintainability of the code.

Redacted: Acknowledged. The data above were used as an on-chain copy for verification purposes without relying on events logging, thus will be kept as it is.

Spearbit: Acknowledged

5.5 Informational

5.5.1 `Swapper._externalCall()` can be simplified and calldata reduced

Severity: Informational

Context: Swapper.sol#L116

Description: `Swapper._swap()`'s `_swapData` param accepts indices which are then used to calculate lengths of data chunks which is overly complex because it has to be +1 larger in size. `_swap()` then has a loop that calls `_externalCall()` which uses `add(d, _dataOffset)` for `argsOffset` to the `call` operation.

Recommendation: If it is assumed that the data starts at a fixed offset, lengths can be passed instead of indices thus simplifying this calculation. This would also eliminate the need for the `_swapData.startIndexes[i + 1] - (_swapData.startIndexes[i])` calculation.

Redacted: Fixed. The recommendation was applied.

Spearbit: Verified.

5.5.2 Redundant logic can be simplified and improve gas cost

Severity: Informational

Context: BribeMarket.sol#L92-L120, BribeFactory.sol#L49-L55, BribeFactory.sol#L98-L111

Description: The `initialize` function in BribeMarket.sol checks if `_bribeVault` is equal to `address(0)` and reverts if so. However, this check is redundant because the `_bribeVault` value should already be validated in the Factory contract, which is responsible for calling this `initialize` function. Specifically, `_setBribeVault` in BribeFactory.sol checks the `codeSize` of `_bribeVault` and will revert if it is zero (i.e., it's not a valid contract). Additionally, any calls not originating from the Factory contract would not be compliant with the intended operational flow and could present a front-running risk when initializing.

Recommendation: Considering that the `_bribeVault` is already checked in the Factory contract, the check in `initialize` seems unnecessary and could be removed to reduce gas cost and simplify the code.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.3 `verifyCalldata` instead of `verify` for `MerkleProof.sol`

Severity: Informational

Context: RewardDistributor.sol#L130

Description: This call in `RewardDistributor.claim` uses the `verify` function from `MerkleProof.sol` in `openzeppelin-contracts` however since version `4.7.0`, the function `verifyCalldata` has been available for evaluating calldata arguments.

Recommendation: May consider upgrading the `openzeppelin-contracts` dependency to at least version `4.7.0` to have access to this more appropriate function.

Redacted: Fixed. Now using `verifyCalldata` instead of `verify` commit 8236f.

Spearbit: Fixed.

5.5.4 Unnecessary checks for zero-length arrays

Severity: Informational

Context: RewardDistributor.sol#L58, RewardDistributor.sol#L77

Description: Calls to `RewardDistributor.claim` and `RewardDistributor.updateRewardsMetadata` with empty calldata would simply result in no-ops making these checks unnecessary.

Recommendation: Remove these checks unless it's important these calls revert.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.5 Safety Checks Missing in `VaultOperator` Constructor

Severity: Informational

Context: VaultOperator.sol#L19-L25

Description: In the `VaultOperator` contract, the constructor sets the `bribeVault` and `rewardDistributor` addresses without checking whether these addresses are non-zero, or if they represent deployed contracts. This contrasts with the approach in other contracts where such safety checks are implemented.

Ignoring these checks could lead to serious issues.

Recommendation: We recommend enforcing these safety checks within the constructor for `VaultOperator` to ensure both the `_bribeVault` and `_rewardDistributor` are valid addresses and represent deployed contracts. This can be done as follows:

```
constructor(address _bribeVault, address _rewardDistributor) {
    require(Address.isContract(_bribeVault), "BribeVault address must be a contract");
    require(Address.isContract(_rewardDistributor), "RewardDistributor address must be a contract");

    bribeVault = IBribeVault(_bribeVault);
    rewardDistributor = IRewardDistributor(_rewardDistributor);

    // Default to the deployer
    operator = msg.sender;
}
```

Redacted: Fixed. Added `address(0)`checks to `_bribeVault` and `_rewardDistributor` commit 7f728b.

Spearbit: Verified.

5.5.6 Optimizing Joining/Leaving Logic and Events in `RewardHarvester` Contract

Severity: Informational

Context: RewardHarvester.sol#L91-L105

Description: The `join` and `leave` functions in the `RewardHarvester` contract allow users to join and leave a reward pool, with each function emitting a corresponding event. However, there is no idempotent check to prevent users from calling the `join` function when they are already in, or the `leave` function when they are already out, which could lead to confusion and unnecessary gas consumption.

Furthermore, considering that joining and leaving can be seen as switching the member status, these two actions could be refactored into a single function and event, enhancing efficiency and code readability.

Recommendation: Introduce an idempotent check in the `join` and `leave` functions so they revert if users try to join when they are already in, or leave when they are already out.

Moreover, you may consider refactoring these two functions into a single function, such as `joinLeave()`, that toggles the membership status of the user and emits a single event detailing the change. This would streamline the contract's operations, reduce gas costs, and enhance clarity. Here's a possible implementation:

```
/**
    @notice Toggle membership status
 */
function joinLeave() external {
    bool _newStatus = !isMember[msg.sender];
    isMember[msg.sender] = _newStatus;

    emit MemberStatusChanged(msg.sender, _newStatus);
}
```

In this approach, the `MemberStatusChanged` event would replace the `MemberJoined` and `MemberLeft` events, and provide a unified way to track membership changes.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.7 Potential Incompatibility with `Fee-On-Transfer` or `Rebasing ERC20` Tokens in `swapAndDepositReward` Function

Severity: Informational

Context: RewardHarvester.sol#L192

Description: The `swapAndDepositReward` function in the given contract assumes that the token being used adheres to the standard `ERC20` token behavior. However, this function may break when interacting with certain kinds of `ERC20` tokens. Tokens that have fees on transfer or tokens that use rebasing mechanisms will have balance changes that are not directly related to the actual transfer of tokens. As such, the calculation of the amount returned might be incorrect when interacting with these types of tokens.

Recommendation: Consider explicitly documenting in the contract or the function comments that this function may not work correctly with fee-on-transfer or rebasing tokens. If you want to support these types of tokens, it would be necessary to adjust the function logic to accommodate their unique behaviors.

Redacted: Acknowledged with no action taken on the codebase, but it will be taken into consideration when vetting for new tokens.

Spearbit: Acknowledged.

5.5.8 Missing Inheritance of `IRewardDistributor`and `IBribeVault` Interfaces in Corresponding Contracts

Severity: Informational

Context: RewardDistributor.sol#L13-L146, BribeVault.sol#L12-L450

Description: Both `RewardDistributor` and `BribeVault` contracts are using the `IRewardDistributor` and `IBribeVault` interfaces respectively, but do not explicitly inherit from them in their contract declarations. This can lead to confusion and potential discrepancies between the actual contracts and their intended interfaces.

Recommendation: Explicitly declare `IRewardDistributor` and `IBribeVault` in `RewardDistributor` and `BribeVault` contracts respectively. This enforces the adherence of these contracts to the structure and functions defined in their respective interfaces, enhancing code readability and maintainability.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.9 Unnecessary Check for briber Address in depositBribe Function of BribeVault Contract

Severity: Informational

Context: BribeVault.sol#L285

Description: In the `depositBribe` function of the `BribeVault` contract, there is a check for whether `_dp.briber` is the zero address. However, this condition cannot be true as this function is exclusively called by the `depositBribe` function of the `BribeMarket` contract. In that function, `msg.sender` is passed as the briber, which, by definition, cannot be the zero address. Therefore, this check is unnecessary and can be removed.

Recommendation: Consider removing the unnecessary check for `_dp.briber == address(0)` from the `depositBribe` function in the BribeVault contract. This removal could marginally reduce the gas cost of the function and simplify the code.

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.10 Set to default values is redundant

Severity: Informational

Context: Swapper.sol#L106

Description: By default, Solidity initializes all variables to their default value, which for boolean is `false`. Over the code, explicit initialization is not used, as it costs a little extra gas (with the optimizer off).

Recommendation: Keep consistency and don't initialize to default.

Redacted: Fixed. Commit e3f088.

Spearbit: Verified.

5.5.11 Lack of variables at errors affects debugging and off-chain monitoring

Severity: Informational

Context: Errors.sol

Description: The `Errors.sol` library defines custom errors that can be thrown in the contract. Each error provides a brief message explaining the cause of the error. While this is a great practice in terms of clarity and legibility, the current errors do not provide any additional information about the specific state or variable values that led to the error. This can make debugging and off-chain monitoring more difficult because it requires tracing back through the code to identify the exact state that caused the error.

Recommendation: Consider adding parameters that provide additional context about the error state. This can greatly improve efficiency and save time at debugging and off-chain error monitoring. Example:

```diff
- error InvalidAmount();
+ error InvalidAmount(uint256 amount);
```

Redacted: Acknowledged.

Spearbit: Acknowledged.

5.5.12 Unlocked pragma version may lead to unexpected behaviors

Severity: Informational

Context: Common.sol

Description: `Common.sol` has its Solidity version pragma set as ˆ0.8.0. This unlocked version indicates that the contract could be compiled with any compiler version that is not less than 0.8.0 and less than 0.9.0.

While this flexibility can be advantageous in some cases, it may lead to unintended behaviors. In this case, it looks pretty safe as if this library only includes structs definition, however, if later changed/added some function, this may result in unexpected behaviors, and therefore, the general recommendation and best practice, is to lock the version in all contracts.

Recommendation: Lock version to pragma solidity `0.8.12` to keep consistency and follow best practices

Redacted: Fixed. Now using `0.8.12` on the `Common.sol` contract. Commit 4aca58.

Spearbit: Fixed.

5.5.13 Visibility `public` can be restricted when getters are used

Severity: Informational

Context: BribeMarket.sol#L30-L34, BribeMarket.sol#L425-L437

Description: In the `BribeMarket` contract, `allWhitelistedTokens` and `allBlacklistedVoters` arrays have been declared as `public`. This visibility automatically generates getter functions. As there are additional getter functions `getWhitelistedTokens()` and `getBlacklistedVoters()` defined for these arrays, both variables should be restricted in visibility to avoid duplicates.

Recommendation: Restrict the visibility of `allWhitelistedTokens` and `allBlacklistedVoters` arrays to `private` or `internal` and only use the `getWhitelistedTokens()` and `getBlacklistedVoters()` functions to retrieve the data. This change can help encapsulate the data and ensure it is only accessible through specified functions.

Redacted: Fixed. Made `allWhitelistedTokens` and `allBlacklistedVoters` private. Commit efee72.

Spearbit: Fixed.

5.5.14 Code duplication affects maintainability and efficiency

Severity: Informational

Context: BribeFactory.sol#L89-L111

Description: The functions `_setBribeMarketImplementation` and `_setBribeVault` are both, simple setters, with a condition. They contain code duplication regarding checking if an address has `extcodesize` or not. This is done to ensure that the address parameter is a contract and in that way avoid setting addresses that are not contracts (i.e.: `address(0)` will be avoided).

```
function _setBribeMarketImplementation(address _implementation) internal {
    uint256 codeSize; //@audit from here
    assembly {
        codeSize := extcodesize(_implementation)
    }
    if (codeSize == 0) revert Errors.NotAContract(); //@audit to here, are duplicated

    bribeMarketImplementation = _implementation;
}
```

Note: This condition regarding if an address is not a contract is true except during the constructor time of a contract, where its codesize will be 0.

Recommendation: Consider moving this piece of code to an `internal` function or modifier. For example:

```
modifier isNotContract(address _a) {
  uint256 codeSize;
  assembly {
    codeSize:= extcodesize(_a)
  }
  if (codeSize == 0) revert Errors.NotAContract();
  _;
}
```

Redacted: Fixed. Created and added a `isContract` modifier to both functions. Commit 6dcde2.

Spearbit: Verified.

5.5.15 Unused named returns affects readability

Severity: Informational

Context: BribeVault.sol#L213

Description: In the `getBribe` function, the returned named variables token and amount are not being used within the code. This may be confusing and affect readability. Unutilized variables can make the function appear more complex than it actually is.

Recommendation: Removed the named variables to improve the function's readability and ensure efficiency.

```
/**
    @notice Get bribe information based on the specified identifier
    @param  bribeIdentifier  bytes32  The specified bribe identifier
    */
 function getBribe(
    bytes32 bribeIdentifier
- ) external view returns (address token, uint256 amount)
+ ) external view returns (address, uint256) {
    Bribe memory b = bribes[bribeIdentifier];
    return (b.token, b.amount);
 }
```

Redacted: The method is used as a getter from `BribeMarket` as a convenient method to get bribe info without first getting the hashed identifier, where named return values were used.

Spearbit: Acknowledged.

5.5.16 Missing events affect transparency and monitoring

Severity: Informational

Context: VaultOperator.sol#L39

Description: Missing events in critical functions, especially privileged ones, reduce transparency and ease of monitoring. Users may be surprised at changes affected by such functions without being able to observe related events.

In this case, users may get surprised to know who is the operator at `VaultOperator.sol` after he interacts with some privileged functions.

Recommendation: Add appropriate events to emit in critical/privileged functions.

Redacted: Fixed. Added the `SetOperator` event. Commit b88454.

Spearbit: Fixed.

5.5.17 Use of `immutable` Instead of `constant` for `FEE_BASIS` in `RewardHarvester` Contract

Severity: Informational

Context: RewardHarvester.sol#L22

Description: In the `RewardHarvester` contract, the `FEE_BASIS` state variable is declared as `immutable` and initialized directly in its declaration. The `immutable` keyword in Solidity is used for variables that can be assigned during contract creation (in the constructor), and their value remains constant from that point forward. However, in this case, `FEE_BASIS` is assigned at the point of the declaration itself and does not change its value later, even in the constructor. Therefore, the use of `immutable` could potentially be replaced with `constant` to more accurately reflect the variable's nature and use.

Recommendation: Consider changing the declaration of `FEE_BASIS` from `immutable` to `constant`. This will accurately reflect the variable's usage and might also bring minor gas savings as constant variables do not occupy storage space and their values are inserted directly into the bytecode.

Redacted: Fixed. Commit 2613ec.

Spearbit: Verified.

5.5.18 Potential Unchecked Array Lengths in `_swap` Function of Swapper Contract

Severity: Informational

Context: Swapper.sol#L36

Description: In the `_swap` function of the abstract `Swapper` contract, a loop iterates over `calleesLength` and accesses the elements of several arrays at the index `i`. Currently, there is no explicit check in `_swap` function to ensure that the lengths of these arrays are at least `calleesLength`, which could potentially lead to an out-of-bounds error. Although the `_swap` function, is used solely in the `swapAndDepositReward` function of the `RewardHarvester` contract and a `_canSwap` function ensures valid array lengths before `_swap` is called, it's a best practice to make `_swap` function independently secure and robust, since `Swapper` is an abstract contract and could be used elsewhere with different preconditions.

Recommendation: Move the length check inside the `_swap` function to ensure the lengths of values and `startIndexes` arrays are at least `calleesLength`. This approach makes the function more robust, even if it is used in different contexts or in subclasses of `Swapper`.

Redacted: Fixed. Commit 5ef9af.

Spearbit: Fixed.

5.5.19 Draft version is deprecated

Severity: Informational

Context: ERC20Utils.sol

Description: Utilization of draft versions of contracts can introduce issues into a production environment. Draft contracts are typically preliminary and non-finalized forms of the contract. Using them might cause potential risks and maintenance problems due to potential bugs and a lack of finalized features. In this case, the draft version is just deprecated, and should be replaced with the release version.

Recommendation: Change the deprecated draft version into the actual version. This change would help maintain the codebase's reliability and simplify future maintenance.

Redacted: Acknowledged. `ERC20Utils` is removed as most of it is now not in use with the latest refactoring.

Spearbit: Acknowledged.

5.5.20 Wrong or incomplete comments affects readability and maintenance

Severity: Informational

Context: IBribeMarket.sol#L4, IBribeVault.sol#L6, IRewardDistributor.sol#L6, IvlAura.sol#L12, IPermit2.sol#L21, IPermit2.sol#L23, RewardDistributor.sol#L29, ERC20Utils.sol#L48

Description: Comments are key to understanding the codebase logic. In particular, Natspec comments provide rich documentation for functions, return variables and more. This documentation aids users, developers and auditors in understanding what the functions within the contract are meant to do.

All the interfaces are missing Natspec comments, which can be later inherited with @inheritdoc in the contracts that implement it:

- IBribeMarket.sol#L4, IBribeVault.sol#L6, IRewardDistributor.sol#L6, IvlAura.sol#L12

Also, some typos are done through the code: Typo: `allownce` -> Corrected: `allowance`

- IPermit2.sol#L21, IPermit2.sol#L23

Typo: `addrews` -> Corrected: `address`

- ERC20Utils.sol#L48

Typo: `identifier` -> Corrected: `identifiers`

- RewardDistributor.sol#L29

Recommendation: Add and fix comments where needed. For Natspec comments, include the missing parameters, returns and descriptions where needed.

Redacted: Fixed. Commit 1023d2.

Spearbit: Fixed.

5.5.21 `RewardHarvester.depositReward`, `BribeVault.transferBribes` - events might contain the wrong value transferred

Severity: Informational

Context: RewardHarvester.sol#L168, BribeVault.sol#L429

Description: `RewardHarvester.depositReward` facilitates the deposit of rewards to the contract. As part of the transaction, the `BribeTransferred` event is emitted. The event uses `_amount`, which is the amount of default tokens provided by the depositor to be deposited. However, in case the default token is a "fee-on-transfer" token, the logged amount won't necessarily reflect the true value that was deposited into the contract. The same issue applies to the `TransferBribe` event in `BribeVault.transferBribes` which uses `distributorAmount` instead of `distributorAmountReceived`.

Recommendation: Consider implementing the before and after balance querying as implemented in BribeVault.sol#L288-L296.

Redacted: Fixed. Applied the recommendation. Commit da7c84 and commit c32e04.

Spearbit: Verified.

5.5.22 Unused code

Severity: Informational

Context: RewardHarvester.sol#L254-L260, RewardHarvester.sol#L266-L272, RewardHarvester.sol#L33, ERC20Utils.sol#L51-L53, ERC20Utils.sol#L59-L61, ERC20Utils.sol#L95-L113, ERC20Utils.sol#L69-L87

Description: Unused code may pose significant risks, including potential security vulnerabilities and increased complexity, which in turn escalates maintenance costs. Additionally, such code can result in higher gas consumption during contract deployment, leading to additional expenses.

During the security assessment, we were able to identify the following unused code sections:

1. `RewardHarveser` - The feature to support approved routers is not used and should be entirely removed.
2. `ERC20Utils` - `approve`, `transferTokens`, `maxUint`, `ethAddress` functions are not used.

Redacted: Fixed. Removed all reported unused code. Commit ae28ce and commit f1a44ce.

Spearbit: Fixed.

5.5.23 Events without indexed event parameters make it harder/inefficient for off-chain tools

Severity: Informational

Context: BribeMarket.sol#L63-L68, BribeMarket.sol#L47-L53, BribeVault.sol#L50-L54, RewardHarvester.sol#L44-L53, RewardHarvesterClaim.sol#L22

Description: Indexed event fields make them quickly accessible to off-chain tools that parse events. However, note that each indexed field costs extra gas during emission; so it's not necessarily best to index the maximum allowed per event (three fields).

Recommendation: Consider which event parameters could be particularly useful for off-chain tools and index them.

Redacted: Acknowledged. Events that require indexing for efficient off-chain calculation were already indexed, and as such the rest were deemed not required for indexing.

Spearbit: Acknowledged.

5.5.24 The system is heavily dependent on off-chain components

Severity: Informational

Context: Hidden Hand Protocol

Description: In the course of our security audit, we have identified a significant dependency on off-chain components. Our ability to cover and foresee every potential bug in this engagement is limited due to the fact that many processes are implemented off-chain. This dependency on off-chain systems could potentially expose the protocol to unforeseen vulnerabilities and risks.

Recommendation: Given the heavy reliance on off-chain systems, we strongly recommend that the team undergo an off-chain audit. This will help to identify and mitigate potential vulnerabilities that may not be apparent in an on-chain audit. It is crucial to ensure that all aspects of the system, both on-chain, and off-chain, are thoroughly audited to maintain the highest level of security and reliability.

Redacted: Acknowledged.

Spearbit: Acknowledged.