PUBLIC

Code Assessment

of the Bribe Platform Smart Contracts

Jan 17, 2023

Produced for

by

Contents

1 Executive Summary

Dear StakeDao team,

Thank you for trusting us to help StakeDao with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Bribe Platform according to Scope to support you in forming an opinion on their security risks.

StakeDao implements smart contracts allowing users to incentivize (or bribe) Curve token holders to vote for a specific Curve gauge.

The most critical subjects covered in our review are Adjusted Bias Measured Possibly Too Late and Queued Upgrade Still Taken in Account After Closing Bribe. Both issues open the possibility to drain funds. All critical and high issues raised have been corrected accordingly. Still, many issues were acknowledged or the risk is accepted.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	1
• Code Corrected	1
High-Severity Findings	1
• Code Corrected	1
Medium-Severity Findings	5
• Code Corrected	1
• Risk Accepted	4
Low-Severity Findings	7
• Code Partially Corrected	1
• Risk Accepted	3
• Acknowledged	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Bribe Platform repository based on written communication.

The scope consists of two solidity smart contracts:

1. ./src/Platform.sol
2. ./src/PlatformFactory.sol

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	22 Nov 2022	c0f573bd1a8f31dd1852250f498ec4294be64eb6	Initial Version
2	29 Nov 2022	7e9fe1582b7f1169bad12f51a707ffe25a88aa6c	Version 2

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

StakeDao offers a bribes system, that provides economic incentives for holders of `veCRV` tokens to vote on desirable Curve liquidity gauges, as part of the Curve CRV inflation distribution mechanism.

2.2.1 Curve CRV distribution

The Curve protocol defines an inflation schedule for its own CRV governance token. Every week, a preset amount of CRV is minted and distributed to Liquidity Providers (LPs) that stake their liquidity tokens in Curve Gauges. Holders of VeCRV, the vested CRV token, can vote on their preferred Gauges, and the amount of CRV allocated per Gauge is proportional to the outcome of the weekly vote. Curve's GaugeController holds a list of Gauges and their relative weight according to the user's votes. CRV rewards will then be distributed each `period`, in a one-week timeframe for which the votes are reevaluated and distribution changed consequently. Once CRV is locked in the Curve Voting Escrow as VeCRV, it cannot be transferred. StakeDao's Bribe system allows holders of VeCRV to profit from their voting power by allowing rewards to be allocated for votes cast toward specific Gauges.

2.2.2 StakeDAO bribes contract

StakeDao introduces the contract `Platform.sol` which will handle the rewards distribution, and the factory contract `PlatformFactory.sol` that is used to manage and deploy one `Platform` contract per `GaugeController` distinct address.

2.2.3 PlatformFactory

The `PlatformFactory` is used as a contract factory and the `Platform` manager, managing platforms is possible using a constructor-defined owner account. The most important feature is the `deploy` function. It is used to deploy a new `Platform` smart contract. The deployment address will directly depend on the `gaugeController` passed as a parameter. Note, that this function is accessible by anyone and that no checks concerning the validity of the `gaugeController` is done. Each platform can then be mapped with an owner-defined `platformFee` through the `setPlatformFee` function. The fees will be directly sent to the `feeCollector` address which is defined through an owner-permissioned setter function. A platform can also be "killed" by the owner through the factory, which will disable almost all actions on the specific `Platform` contract.

2.2.4 Platform

Bribing users will be done through the `Platform` smart contracts. Any account can bribe voters for a valid gauge and with any reward token. The creator of the bribe can choose certain periods into which the total reward amount will be split and then proportionally shared with voters of the gauge for each period. They can also specify a blacklist of users that should not be rewarded and not be taken into account for the total voting power computation for each period. Additionally, a manager for the bribe is defined at creation time. The previously explained logic is implemented in the `createBribe` function, which returns a new bribe id once executed. Later on, if the bribe has been made upgradable by its creator, the bribe manager can upgrade it and add a reward amount including additional periods to bribe voters. Voters can claim their rewards through the `claim` and `claimAll` functions. The `gaugeController` is queried to retrieve the voting power of the user and to compute the share that they should receive for the current period. Unclaimed rewards for each period will be accumulated and split across every period left. Leftovers can be claimed by the bribe manager once the bribe has ended or if the platform has been killed by the factory owner.

2.2.5 Trust model and assumptions

The ERC20 tokens used are assumed to not have fees and to not be rebasing. The factory owner is supposed to be trusted, he has the power to kill any deployed platform and to set a fee for each bribe creation of up to 99% of the reward amount. Bribe managers are assumed to not be trusted.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	4

- Unkill Function Allows Claiming After Closing Risk Accepted
- Bribe Manager Can Deny Bribe by Decreasing maxRewardPerVote Risk Accepted
- GaugeController Not Checkpointed Before First Period Update Risk Accepted
- Repeated Addresses in Bribe Blacklist Cause Total Bias Under Estimation Risk Accepted

Low-Severity Findings	7

- Error Messages and Event Usage Acknowledged
- Gas Optimizations Code Partially Corrected
- Incorrect User Bias Calculation Risk Accepted
- Missing Sanity Checks Risk Accepted
- Naming Issues, NatSpec Missings, Incorrect Comments, Typos Acknowledged
- Unused Imports Acknowledged
- safeTransfer Functions Do Not Check Contract Existence Risk Accepted

5.1 Unkill Function Allows Claiming After Closing

Security Medium Version 2 Risk Accepted

Function `unKill()`, only callable by the `PlatformFactory` contract owner, allows to reset `isKilled` to false. If a bribe manager calls `closeBribe()` while the Platform is killed, and the platform is then unkilled, the bribe becomes claimable again, even though the left over funds have been transferred by `closeBribe()`. Users can claim their bribes and the funds will be taken from other bribes sharing the same tokens.

Risk accepted

StakeDao accepts the risk but already fixed the issue by removing the unkill function in the latest code version that was not included in the audit.

5.2 Bribe Manager Can Deny Bribe by Decreasing maxRewardPerVote

Security Medium Version 1 Risk Accepted

The bribe manager can use `increaseBribeDuration()` to queue a decrease of `maxRewardPerVote` to close to 0 just before the start of a claiming period, to rug the expected bribe of users who have already voted.

Risk accepted

StakeDao states that they accept the risk.

5.3 GaugeController Not Checkpointed Before First Period Update

Correctness Medium Version 1 Risk Accepted

The `GaugeController` is not checkpointed in `_updateRewardPerToken()`. If no vote has been cast on the gauge before the first period, `_getAdjustedBias()` will return 0 instead of the actual value, or might revert if blacklisted users cause an underflow to happen. This can cause the reward to become unclaimable for some voters.

Risk accepted

StakeDao states:

```
While it would cause an issue for the first period with old vote users not
being able to claim their rewards, it would be solved by the next period
with the rolling over.
```

5.4 Repeated Addresses in Bribe Blacklist Cause Total Bias Under Estimation

Security Medium Version 1 Risk Accepted

Newly created bribes can have repeated addresses in the blacklist. If an address is present multiple times in the blacklist, its bias will be deducted multiple times from the total bias in `_getAdjustedBias()`, and the function might return a value smaller than the cumulative bias of potential claimers. `rewardPerToken` can therefore be manipulated upward by inserting repeated addresses in the blacklist.

Risk accepted

StakeDao states that they accept the risk.

5.5 Error Messages and Event Usage

Design Low Version 1 Acknowledged

1. In `PlatformFactory`, StakeDao might consider adding an event to the state change in `setFeeCollector`.
2. In `Platform`, the error messages `INVALID_GAUGE` is not used.

Acknowldeged

StakeDao acknowledges the issue. No actions are taken.

5.6 Gas Optimizations

Design Low Version 1 Code Partially Corrected

1. Double external call to `vote_user_slopes` in `_claim()`, at line 385 - 387
2. The function `getActivePeriod` is redundant since `activePeriod` is already public and will implicitly define an external getter
3. `_updateRewardPerToken` calls `getCurrentPeriod` which was in both execution flows called right before in the parent function
4. `getPeriodsLeft` and `getActivePeriodPerBribe` copy the entire `Bribe` struct from storage to memory, but only use 2 of the fields from the struct. This causes unnecessary `SLOAD` operations to be performed, at a cost that scales linearly with the size of the blacklist.

Code partially corrected

`getCurrentPeriod()` is now only called once. The two other potential optimizations were not applied.

5.7 Incorrect User Bias Calculation

Correctness Low Version 1 Risk Accepted

The internal `_getAddrBias()` function returns *0* if `currentPeriod + _WEEK >= endLockTime`. However, as long as `endLockTime` is bigger than `currentPeriod` the user has voting power. Indeed, in its time progression, a user bias will incorrectly go from `slope * 3 * WEEK` to `slope * 2 * WEEK` to `0` while skipping `slope * 1 * WEEK`.

Risk accepted

StakeDao is aware of the issue but decided to accept the risk and leave the code as it is.

5.8 Missing Sanity Checks

Design Low Version 1 Risk Accepted

The following arguments are not checked or are insufficiently checked if they make sense:

- In `Platform.createBribe` the variable `manager` (address zero check), `maxRewardPerVote` (zero check) and a check for `rewardPerPeriod` as it could be zero after the division with `numberOfPeriods`
- In `Platform.updateManager` there is no sanity check for address zero
- In `Platform._claim()` it is not checked that the bribe exists
- In `PlatformFactory` setting the fee collector and transferring the owner are not checked for address zero

---

Risk accepted

StakeDao states that they accept the risk.

5.9 Naming Issues, NatSpec Missings, Incorrect Comments, Typos

Design | Low | Version 1 | Acknowledged

In `Platform.sol`:

1. line 104, missing `@notice` for `Upgrade` struct
2. line 126, incorrect grammar: `Minimum duration a Bribe`
3. line 158, rewardPerToken naming is ambiguous, the variable value is better understood as the reward per vote not the reward per token.
4. line 254, `Target bias for the gauge`, incorrect NatSpec on parameter `maxRewardPerVote`
5. In `createBribe()` NatSpec, missing parameters `upgradeable` and `manager`.
6. line 503: comment says `called once per Bribe`, however the function is called multiple times on the first period, but the condition is only true on first call.
7. line 640: `_additionnalPeriods` declaration contains a typo
8. `getActivePeriod` and `getActivePeriodPerBribe` are named ambiguously, they do very different things but share almost the same name

---

Acknowldeged

StakeDao acknowledges the issue. No actions are taken.

5.10 Unused Imports

Design | Low | Version 1 | Acknowledged

The contract `PlatformFactory` imports `ERC20` but does not use it.

Acknowldeged

StakeDao acknowledges the issue. No actions are taken.

5.11 safeTransfer Functions Do Not Check Contract Existence

Security | Low | Version 1 | Risk Accepted

The `safeTransfer` and `safeTransferFrom` functions of solmate's safeTransferLib do not check that the `token` contract actually exists. If called with a token address that doesn't contain code, the calls will succeed even if no transfer is performed. This could be an issue when a token will be deployed at a predictable address.

Risk accepted

StakeDao states that they accept the risk.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	1

- Adjusted Bias Measured Possibly Too Late Code Corrected

High-Severity Findings	1

- Queued Upgrade Still Taken in Account After Closing Bribe Code Corrected

Medium-Severity Findings	1

- closeBribe Does Not Refund Tokens Added in Upgrade Code Corrected

Low-Severity Findings	0

6.1 Adjusted Bias Measured Possibly Too Late

Security Critical Version 1 Code Corrected

The amount of excluded votes belonging to the users in the `blacklist` are counted by the internal function `_getAdjustedBias` for the recently concluded period when `_updateBribePeriod()` is called. However, the period update only happens when users interact with the contract. Between the start of the new voting period (`timestamp / WEEK * WEEK`) and the time `_updateBribePeriod()` is called, a `blacklist` user can cast a new vote on the gauge, which is incorrectly counted by ``_getAdjustedBias()` as belonging to the previous period.

`rewardPerToken` at period `T` is computed as

```
rewardPerToken(T) = rewardPerPeriod / (total_bias(T) - omitted_reward(T_blacklisted_last_vote))
```

So the periods of `total_bias` and `omitted_reward` might not match.

Since the bribe creator has full control on who to include in the blacklist and what gauge to set the bribe on, they can make `rewardPerToken` as high as they desire by making the denominator arbitrarily small with a blacklisted user that they control. Since `_claim()` doesn't check that `bribe.totalRewardAmount` is not exceeded when distributing the reward, a dishonest bribe creator can use this bug to steal funds from other bribes.

Code corrected

A check has been added so that subtracting the bias of a blacklisted user is only performed if the blacklisted user has voted before the start of the period. Otherwise bias is not deducted and rewarded users get a bit less.

```
_lastVote = gaugeController.last_user_vote(_addressesBlacklisted[i], gauge);
if (period > _lastVote) {
    _bias = _getAddrBias(userSlope.slope, userSlope.end, period);
    gaugeBias -= _bias;
}
```

A check is also introduced so that the cumulative bribe payout never exceeds the `bribe.totalRewardAmount`.

6.2 Queued Upgrade Still Taken in Account After Closing Bribe

Security High Version 1 Code Corrected

A queued upgrade for a bribe can still be taken in account after the manager has closed the bribe with `closeBribe`. If it is the case, then part of the following rewards distributed are stolen from other bribes.

Once the bribe is closed by the manager, claiming again will update the bribe and reset the `endTimestamp` in the future, without taking in account the `totalRewardAmount - amountClaimed` amount withdrew by the manager.

Using this attack to steal all the funds of the contract is possible with no risks, but would necessitate at least the same amount of tokens that the attacker wants to steal and being able to lock them for multiple weeks.

Code corrected

The upgrade is now deleted from the queue when `closeBribe()` is called.

6.3 closeBribe Does Not Refund Tokens Added in Upgrade

Design Medium Version 1 Code Corrected

When a bribe is closed while an upgrade is queued, the unclaimed amount will be refunded to the bribe manager, but not the additional `_increasedAmount` added in the queued upgrade.

Code corrected

During the closing of a bribe, if there is an upgrade in the queue, instead of transferring back `total reward amount - amount claimed`, the total amount after the upgrade is used.

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Interface Definitions

Note Version 1

To indicate a file is an interface, the naming convention is to prepend an `I` to the file name. The interface `SmartWalletChecker` and `VeToken` is for test purposes only. Interfaces used only for test purposes are usually separated into test folders.

The following interfaces are defined but not used:

In `GaugeController`: `add_gauge` (only for tests), `WEIGHT_VOTE_DELAY`, `gauge_relative_weight_write`, `gauge_relative_weight`, `gauge_relative_weight`, `get_total_weight`, `get_gauge_weight`, `add_type` (only in tests), `admin`.

7.2 `claimable()` Might Return Incorrect Values

Note Version 1

Due to the nature of view functions not being able to change state, the `claimable()` view function doesn't checkpoint the gauge nor it updates the period, so the value it returns could be invalid. This should be made clear in the natspec.