PUBLIC

Code Assessment

of the Sink Gauge v2

Smart Contracts

Feb 14, 2025

Produced for

velodrome

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Velodrome team,

Thank you for trusting us to help Velodrome with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Sink Gauge v2 according to Scope to support you in forming an opinion on their security risks.

Velodrome implements a setup to lower the need of newly minted Velo which consequently slows down or could even stop the inflation rate. This is achieved by using a special gauge contract. If users vote for this gauge, the corresponding rewards will be sent directly from the voter to the minter contract. There is also a pool without functionality and the factory contracts to correctly integrate the setup in the overall ecosystem

We could not identify any relevant issue in the code base. In summary, we find that the codebase provides a high 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	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

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 four contracts needed for the sink setup inside the Sink Gauge v2 repository.

- `SinkGaugeFactory`
- `SinkPoolFactory`
- `SinkPool`
- `SinkGauge`

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

V	Date	Commit Hash	Note
1	08 Feb 2025	0dd950938a43cb69d98fcd00768b8de1a5b87c52	Initial Version

For the solidity smart contracts, the compiler version `0.8.25` was chosen.

2.1.1 *Excluded from scope*

All files not explicitly mentioned in scope are out of scope. Third-party libraries such as OpenZeppelin libraries are out-of-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.

Velodrome's sink v2 contracts allow users to allocate their votes to the sink gauge. The sink gauges redirect the Velo rewards back to the minter. Consequently, it is a way to reduce the need for new Velo tokens to be minted by the minter each epoch and lower the token inflation rate. To allow users to vote for the single gauge and to integrate the contracts correctly, the sink gauge has a pool associated with it. However, the pool is not functional and has an empty contract. Both contracts have the associated factory contracts, too. All four contracts have minimal functionality to integrate with the system but do not allow users normal pool operations. Additionally, the usual reward contracts are associated with the sink setup such that it fully integrates into voting, bribes, and rewards.

The pool is an empty contract and just has the address needed to set up the sink contracts in the system. The factory is not a real factory that allows to deploy of multiple contracts but simply deploys the empty pool contract and returns the pool address if queried. Additionally, it has two view functions `isPair` and `isPool` that both simply return false.

The gauge factory, like the pool factory, implies deploying the gauge in its constructor but cannot deploy other gauges. No other gauges can be created via the `createGauge` function which is the only function that is present. The function simply returns the address of the gauge when called. The gauge has three functions that need to be at least present to not make the relevant calls revert. `left()` returns always `0` such that it is always possible to send funds through the gauge. `getReward()` is an empty implementation to avoid reverts in the call path needed. `notifyRewardAmount()` is the only function that implements meaningful logic. It can only be called by the Voter contract with a non-zero amount. It tracks the total rewards that were sent over the gauge and the rewards per epoch. Besides tracking it simply redirects the funds from `msg.sender` (the Voter contract) directly to the Minter contract with a `transferFrom()`. Hence, the gauge will not even hold or really "touch" the rewards re-directed over it. Finally, a `NotifyReward` and `ClaimRewards` event is emitted.

The contracts are fully unpermissioned and the only state-changing function can be called by the Voter contract only.

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 Open Findings

In this section, we describe our findings. The findings are split into these different categories:

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	0
Low-Severity Findings	0

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 No Emergency Recovery

Informational **Version 1** **Acknowledged**

CS-VELOG2-001

In case funds are accidentally send to the contract or Ether forced into the contract, none of the contracts has a recovery/sweep function.