MixBytes()

Velodrome CLGaugeFactory Security Audit Report

NOVEMBER 05, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

The audited scope encompasses the gauge and factory infrastructure for a concentrated liquidity AMM protocol. The CLFactory and CLGaugeFactory deploy pools and their corresponding gauge contracts, which manage staking of NFT liquidity positions and distribute emission rewards based on veToken voting. The CLGauge contracts handle the staking lifecycle and reward distribution, while the Redistributor implements emission caps by collecting excess rewards from over-allocated gauges and redistributing them proportionally to other active gauges.

This security audit was conducted as a differential audit from commit 4513a630c3b06def2bacb212e5aa2e2acb03b67e. The audit focused on the new Redistributor contract as a complete implementation review, while CLGauge, CLGaugeFactory, and CLFactory were analyzed specifically for changes and additions from the base commit.

Several contracts were briefly reviewed as off-scope dependencies to understand their interfaces and interaction patterns with the in-scope contracts. The Voter contract was reviewed for its distribution mechanism and gauge registration logic, which directly interacts with the emission system. The Minter contract was examined to understand weekly emission calculations and redistribution mechanisms. The FeesVotingReward contract was reviewed to verify fee distribution flows from gauges. The VotingEscrow contract was analyzed for its team role management and permission structure used by the Redistributor. The NonFungiblePositionManager was examined to understand position ownership verification and the collect mechanism used during gauge deposits. Finally, CLPool was reviewed more thoroughly despite being off-scope due to its critical role in reward synchronization and staking operations with the gauge system.

During the audit, potential vulnerabilities were systematically evaluated against an internal security checklist covering common smart contract risks. This comprehensive review included verification of arithmetic safety in Solidity 0.7.6 contexts, reentrancy protection mechanisms, access control enforcement, state consistency across contract interactions, and proper handling of edge cases in reward calculations. In addition to these foundational checks, the following specific attack vectors and vulnerability patterns were examined in depth:

1. We confirmed that users cannot double-claim rewards as the NFT's reward growth index checkpoint is properly updated after each claim, ensuring only the delta between checkpoints is claimable.

2. We verified that `rewardRate` is correctly updated when `notifyRewardWithoutClaim` is called from the redistributor, as it properly handles the `else` block calculation `(_amount + _leftover) / timeUntilNext` to account for additional rewards within the same epoch.
3. We verified that the architectural changes (adding the redistributor) will not block reward accrual.
4. We reviewed the proposed migration flow from v1 to v2 pools and confirmed that malicious actors cannot block pool creation or deploy a malicious pool that could block or steal liquidity.

In conclusion, the project demonstrates a high level of security, operational integrity, and good overall code quality. The contracts in scope are well-written and follow established best practices. During the audit, several low severity issues were identified that would enhance the contracts robustness if addressed.

1.3 Project Overview

Summary

Title	Description
Client Name	Velodrome
Project Name	CLGaugeFactory
Type	Solidity
Platform	EVM
Timeline	21.10.2025 - 28.10.2025

Scope of Audit

File	Link
contracts/core/CLFactory.sol	CLFactory.sol
contracts/gauge/CLGauge.sol	CLGauge.sol
contracts/gauge/CLGaugeFactory.sol	CLGaugeFactory.sol
contracts/gauge/Redistributor.sol	Redistributor.sol

Versions Log

Date	Commit Hash	Note
21.10.2025	0bfd7445fb8f0ffd5886d0242cbb2243cd005ae9	Initial Commit
28.10.2025	8bc4bce185f0cd32a7426998a2d2c0bec6e3c335	Re-audit Commit

Mainnet Deployments

File	Address	Blockchain
Redistributor.sol	0x11a53f31Bf406de59fCf9613E1922bd3E283A4B4	Base

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	7

Findings Statuses

ID	Finding	Severity	Status
L-1	Missing Upper Bound Validation for `defaultCap` During `CLGaugeFactory` Deployment	Low	Fixed
L-2	Validation Missing for `_end` Exceeding `_start` in `_redistribute()`	Low	Fixed
L-3	Streamline Epoch Timing Logic in `_notifyRewardAmount()`	Low	Fixed
L-4	Unnecessary `receive()` in `CLGauge` (ETH should never be sent to gauge)	Low	Fixed
L-5	Inefficient Token Transfer Pattern in Emission Cap Handling	Low	Fixed
L-6	Emission Cap Bypass Through Rollover Accumulation	Low	Acknowledged
L-7	Single-Step Admin Transfer Risks Permanent Access Loss	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Missing Upper Bound Validation for defaultCap During CLGaugeFactory Deployment		
Severity	Low	**Status**	Fixed in 8bc4bce1

Description
During the deployment of the CLGaugeFactory contract, there is no upper bound validation for defaultCap in the constructor. This validation is, however, enforced in the CLGaugeFactory.setDefaultCap function. This inconsistency could lead to deployments with unreasonably high emission caps, which contradict the intended emission distribution mechanism and bypass the safety checks implemented in the setter function.

Recommendation
We recommend adding upper bound validation in the constructor to ensure that defaultCap does not exceed MAX_BPS.

Client's Commentary
Fixed in 381f27c3

L-2	Validation Missing for `_end` Exceeding `_start` in `_redistribute()`		
Severity	Low	**Status**	Fixed in 8bc4bce1

Description

The `Redistributor._redistribute()` function lacks validation to ensure that `_end` is greater than `_start`, which could lead to an underflow in the loop condition and subsequent out-of-bounds array access. If `_start > _end`, the expression `_end - _start` will underflow in Solidity 0.7.6, causing the transaction to revert with an unclear error message.
Although this is a rare case since the function is restricted to `onlyUpkeepOrKeeper`, it is still better to provide a clear error message rather than relying on panic errors.

Recommendation

We recommend adding a validation check to ensure that `_end > _start`.

Client's Commentary

Fixed in 7e3da162

L-3	Streamline Epoch Timing Logic in _notifyRewardAmount()		
Severity	Low	**Status**	Fixed in 8bc4bce1

Description

In the CLGauge._notifyRewardAmount() function, the following code could be optimized:

```
uint256 timestamp = block.timestamp;
uint256 timeUntilNext = ProtocolTimeLibrary.epochNext(timestamp) - timestamp;
pool.updateRewardsGrowthGlobal();
uint256 nextPeriodFinish = timestamp + timeUntilNext;
```

Recommendation

We recommend changing the order so that nextPeriodFinish is computed first via epochNext(timestamp), and timeUntilNext is then derived as the difference.

```
uint256 timestamp = block.timestamp;
uint256 nextPeriodFinish = ProtocolTimeLibrary.epochNext(timestamp);
uint256 timeUntilNext = nextPeriodFinish - timestamp;
pool.updateRewardsGrowthGlobal();
```

Client's Commentary

Fixed in ab3ad7a1

L-4	Unnecessary receive() in CLGauge (ETH should never be sent to gauge)		
Severity	Low	**Status**	Fixed in 8bc4bce1

Description

In the CLGauge contract, the receive() function is defined to accept ETH only from the NFT position manager (require(msg.sender == address(nft))). Under the current design, the gauge is not intended to be an ETH recipient.

Recommendation

We recommend removing the receive() function or make it revert unconditionally to forbid inbound ETH.

Client's Commentary

Fixed in 8bc4bce1

L-5	Inefficient Token Transfer Pattern in Emission Cap Handling		
Severity	Low	**Status**	Fixed in 8bc4bce1

Description

In `CLGauge.notifyRewardAmount()`, when a gauge exceeds its emission cap, the function performs two separate token transfers from the voter instead of one. First, it transfers the excess amount, then transfers the capped amount via `_notifyRewardAmount()`. This increases gas costs for every capped distribution.

Recommendation

We recommend moving the initial `safeTransferFrom()` to transfer all tokens before the cap check, eliminating the duplicate transfer. Then only approve and forward the excess to the redistributor. Update `_notifyRewardAmount()` to skip the transfer when called from `notifyRewardAmount()`.

Client's Commentary

Fixed in 0f36abdb

L-6	Emission Cap Bypass Through Rollover Accumulation		
Severity	Low	**Status**	Acknowledged

Description

In `CLGauge._notifyRewardAmount()`, rollover tokens are added after the emission cap check has been performed. The cap is enforced in `notifyRewardAmount()`, but rollover is added afterward in the internal function. This allows gauges to receive more than their configured cap when rollover accumulates.

For gauges with no staked liquidity, all emissions become rollover. Over multiple epochs, this compounds, allowing a gauge to accumulate rollover well beyond its cap. When liquidity is added, the gauge distributes accumulated rollover plus new capped emissions, bypassing the cap. The tracked amount in `rewardsByEpoch[epochStart]` includes rollover, which can cause issues in the redistributor's cap validation logic.

Recommendation

We recommend retrieving `pool.rollover()` before the emission cap check in `notifyRewardAmount()`, add it to the incoming amount, then apply the cap to the total.

Client's Commentary

The finding has been acknowledged and will not be fixed. Rollover after `distribute` and before `redistribute` should be negligible so it should be okay.

L-7	Single-Step Admin Transfer Risks Permanent Access Loss		
Severity	Low	**Status**	Acknowledged

Description

In `CLGaugeFactory.setEmissionAdmin()` and `setNotifyAdmin()`, admin roles are transferred immediately without confirmation from the recipient. A mistyped address permanently locks access to critical functions. Loss of these roles requires redeploying the factory and all gauges, as there is no recovery mechanism.

Recommendation

We recommend implementing a two-step transfer process:

1. Current admin proposes a new admin via `transferAdmin()`, storing the address in a `pendingAdmin` variable;
2. The pending admin calls `acceptAdmin()` to complete the transfer.

Client's Commentary

The finding has been acknowledged and will not be fixed. The risk is accepted, and appropriate measures will be taken when transferring admin permissions.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes