

MixBytes()

# Velodrome Pool Launcher Security Audit Report

NOVEMBER 10, 2025

# Table of Contents

<b>1. Introduction</b>	<b>2</b>
1.1 Disclaimer	2
1.2 Executive Summary	2
1.3 Project Overview	3
1.4 Security Assessment Methodology	5
1.5 Risk Classification	7
1.6 Summary of Findings	8
<b>2. Findings Report</b>	<b>10</b>
2.1 Critical	10
2.2 High	10
2.3 Medium	10
M-1 Unclaimed Fees Become Inaccessible After V2Locker.unlock()	10
2.4 Low	11
L-1 Unbounded lockUntil Parameter Allows Permanent Token Lock	11
L-2 Missing Pool Validation Lets Anyone Create Lockers for Arbitrary Tokens	12
L-3 Potential Spam Lock Creation Without Minimum Liquidity Requirements	13
L-4 Staked Lockers Cannot Be Unlocked Without Prior Unstaking	14
L-5 Left-Over Token Dilution During V2 Playground Migration	15
L-6 Pre-Transferring Tokens Enables Front-Running Attack	16
L-7 Fees Remain on Locker When bribeAmount Is Zero	17
L-8 Rounding Error Accumulates Against Beneficiary When Deducting Share	18
L-9 Unused and Redundant Imports	19
L-10 Potential Revert on Zero Bribe Amount	20
L-11 Unnecessary Zero Token Transfers in Fee Claims	21
L-12 Missing Reentrancy Protection in CLLocker.increaseLiquidity()	22
<b>3. About MixBytes</b>	<b>23</b>

# 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

Velodrome Pool Launcher is a liquidity – bootstrapping extension that creates, manages, and eventually unlocks time-locked liquidity positions called lockers. Users create dedicated lockers, deposit their LP tokens or LP-NFTs into them, and optionally stake the locked assets in a gauge. While the assets remain locked, users collect trading fees directly from the pool and, when staked, accrue staking rewards distributed by the gauge.

The codebase was audited over 6 days by a team of 3 auditors, using a combination of manual review and automated tooling.

During the audit, in addition to verifying standard attack vectors and our internal checklist, we conducted an in-depth review of the following areas:

- **Approval Management Security.** We confirmed that tokens are approved only to protocol-controlled addresses added by admins (e.g., pools, nfpManager), and approvals are revoked after each interaction with external contracts.
- **Post-Unlock Quiescence.** Once `unlock()` executes, `lockedUntil` is cleared, all `onlyLocked` or staked gates block further actions, and the locker can no longer interact with the pool or gauge – eliminating any subsequent effect on protocol state.
- **Assets Return Safely on Unlock.** We verified that throughout the lock's lifecycle the LP tokens or NFT stay under the locker's (or its gauge's) control, and once unlocking is allowed the assets are transferred back in full to the owner – no residual balances remain.
- **Locker Lifecycle Consistency.** The locker goes through a lifecycle that includes locked/unlocked and staked/unstaked states. We checked that the locker's state changes correctly and cannot be accidentally or deliberately violated.
- **Stranded Balances, Rewards, and Fees.** We checked whether funds that should have been delivered to the user can become stuck on the locker contract.

The codebase is well-structured, consistently written, and adheres to established development practices. During the course of our review, no significant security vulnerabilities or design flaws were identified. The findings observed during the audit, along with detailed explanations and recommendations, are presented in the **Findings Report** section below.

## 1.3 Project Overview

### Summary

Title	Description
Client Name	Velodrome
Project Name	Pool Launcher
Type	Solidity
Platform	EVM
Timeline	12.09.2025 – 03.10.2025

### Scope of Audit

File	Link
src/LockerFactory.sol	<a href="#">LockerFactory.sol</a>
src/Playground.sol	<a href="#">Playground.sol</a>
src/Locker.sol	<a href="#">Locker.sol</a>
src/libraries/CreateXLibrary.sol	<a href="#">CreateXLibrary.sol</a>
src/libraries/TickMath.sol	<a href="#">TickMath.sol</a>
src/extensions/v2/V2Locker.sol	<a href="#">V2Locker.sol</a>
src/extensions/v2/V2Playground.sol	<a href="#">V2Playground.sol</a>
src/extensions/v2/V2LockerFactory.sol	<a href="#">V2LockerFactory.sol</a>
src/extensions/cl/CLLocker.sol	<a href="#">CLLocker.sol</a>
src/extensions/cl/CLLockerFactory.sol	<a href="#">CLLockerFactory.sol</a>
src/extensions/cl/CLPlayground.sol	<a href="#">CLPlayground.sol</a>
src/PoolLauncher.sol	<a href="#">PoolLauncher.sol</a>
src/extensions/cl/CLPoolLauncher.sol	<a href="#">CLPoolLauncher.sol</a>

File	Link
<code>src/extensions/cl/V2PoolLauncher.sol</code>	<a href="#">V2PoolLauncher.sol</a>

## Versions Log

Date	Commit Hash	Note
12.09.2025	c0cb381bc33c9ecf17f6a929c4241b5a9b3cacd8	Initial commit
03.10.2025	9b340a16356d40e08845f1e66de1ad09e72d5251	Re-audit commit
10.11.2025	e4bc22e8f86d6e9c68b1067199d98551b1283f22	Deployed Commit

## Mainnet Deployments

File	Address	Blockchain
<code>V2Locker.sol</code>	<a href="#">0x11bc9051A4EE340FD182A6cA72d18eC93d92E60d</a>	Base
<code>V2LockerFactory.sol</code>	<a href="#">0x067b028C66f61466F66864cc01F92Afc7D99e530</a>	Base
<code>V2PoolLauncher.sol</code>	<a href="#">0xA81eEbdEb3129bf5B89AEd89EDe9eC5fB6FDE3B3</a>	Base
<code>CLLocker.sol</code>	<a href="#">0xE5123AC41655b1A5CA09B3A6BE6c7151E67C00E0</a>	Base
<code>CLLockerFactory.sol</code>	<a href="#">0x8BF02b8da7a6091Ac1326d6db2ed25214D812219</a>	Base
<code>CLPoolLauncher.sol</code>	<a href="#">0xb9A1094D614c70B94C2CD7b4efc3A6adC6e6F4d3</a>	Base
<code>V2Locker.sol</code>	<a href="#">0x11bc9051A4EE340FD182A6cA72d18eC93d92E60d</a>	Optimism
<code>V2LockerFactory.sol</code>	<a href="#">0x067b028C66f61466F66864cc01F92Afc7D99e530</a>	Optimism
<code>V2PoolLauncher.sol</code>	<a href="#">0xA81eEbdEb3129bf5B89AEd89EDe9eC5fB6FDE3B3</a>	Optimism
<code>CLLocker.sol</code>	<a href="#">0xE5123AC41655b1A5CA09B3A6BE6c7151E67C00E0</a>	Optimism
<code>CLLockerFactory.sol</code>	<a href="#">0x8BF02b8da7a6091Ac1326d6db2ed25214D812219</a>	Optimism
<code>CLPoolLauncher.sol</code>	<a href="#">0xb9A1094D614c70B94C2CD7b4efc3A6adC6e6F4d3</a>	Optimism

## 1.4 Security Assessment Methodology

### Project Flow

Stage	Scope of Work
Interim audit	<b>Project Architecture Review:</b> <ul style="list-style-type: none"><li>• Review project documentation</li><li>• Conduct a general code review</li><li>• Perform reverse engineering to analyze the project's architecture based solely on the source code</li><li>• Develop an independent perspective on the project's architecture</li><li>• Identify any logical flaws in the design</li></ul> <p><b>OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.</b></p>
	<b>Code Review with a Hacker Mindset:</b> <ul style="list-style-type: none"><li>• Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities.</li><li>• Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead.</li><li>• Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws.</li><li>• Review test cases and in-code comments to identify potential weaknesses.</li></ul> <p><b>OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.</b></p>
	<b>Code Review with a Nerd Mindset:</b> <ul style="list-style-type: none"><li>• Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits.</li><li>• Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors.</li></ul> <p><b>OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.</b></p>

Stage	Scope of Work
	<p><b>Consolidation of Auditors' Reports:</b></p> <ul style="list-style-type: none"> <li>• Cross-check findings among auditors</li> <li>• Discuss identified issues</li> <li>• Issue an interim audit report for client review</li> </ul> <p><b>OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.</b></p>
Re-audit	<p><b>Bug Fixing &amp; Re-Audit:</b></p> <ul style="list-style-type: none"> <li>• The client addresses the identified issues and provides feedback</li> <li>• Auditors verify the fixes and update their statuses with supporting evidence</li> <li>• A re-audit report is generated and shared with the client</li> </ul> <p><b>OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.</b></p>
Final audit	<p><b>Final Code Verification &amp; Public Audit Report:</b></p> <ul style="list-style-type: none"> <li>• Verify the final code version against recommendations and their statuses</li> <li>• Check deployed contracts for correct initialization parameters</li> <li>• Confirm that the deployed code matches the audited version</li> <li>• Issue a public audit report, published on our official GitHub repository</li> <li>• Announce the successful audit on our official X account</li> </ul> <p><b>OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.</b></p>

## 1.5 Risk Classification

### Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: 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	1
Low	12

### Findings Statuses

ID	Finding	Severity	Status
M-1	Unclaimed Fees Become Inaccessible After <code>V2Locker.unlock()</code>	Medium	Fixed
L-1	Unbounded <code>lockUntil</code> Parameter Allows Permanent Token Lock	Low	Acknowledged
L-2	Missing Pool Validation Lets Anyone Create Lockers for Arbitrary Tokens	Low	Fixed
L-3	Potential Spam Lock Creation Without Minimum Liquidity Requirements	Low	Acknowledged
L-4	Staked Lockers Cannot Be Unlocked Without Prior Unstaking	Low	Fixed
L-5	Left-Over Token Dilution During V2 Playground Migration	Low	Fixed
L-6	Pre-Transferring Tokens Enables Front-Running Attack	Low	Fixed
L-7	Fees Remain on Locker When <code>bribeAmount</code> Is Zero	Low	Fixed
L-8	Rounding Error Accumulates Against Beneficiary When Deducting Share	Low	Acknowledged
L-9	Unused and Redundant Imports	Low	Fixed

<b>L-10</b>	Potential Revert on Zero Bribe Amount	Low	Fixed
<b>L-11</b>	Unnecessary Zero Token Transfers in Fee Claims	Low	Fixed
<b>L-12</b>	Missing Reentrancy Protection in CLLocker.increaseLiquidity()	Low	Fixed

# 2. Findings Report

## 2.1 Critical

Not Found

## 2.2 High

Not Found

## 2.3 Medium

M-1	Unclaimed Fees Become Inaccessible After <code>V2Locker.unlock()</code>		
Severity	Medium	Status	Fixed in 9b340a16

### Description

`Locker.claimFees()` is protected by the `onlyLocked` modifier, which requires `lockedUntil != 0`. Inside `V2Locker.unlock()`, the contract sets `lockedUntil = 0` before transferring the LP tokens to the recipient. During the transfer, the underlying pool credits all accumulated fees to the locker contract. Once the LP tokens are transferred out, the locker can no longer call `claimFees()` because it is no longer locked, leaving those fees permanently stranded. A user who forgets to call `claimFees()` before unlocking will lose the entire fee balance earned up to that moment.

### Recommendation

We recommend claiming swap fees within the `V2Locker.unlock()` function.

### Client's Commentary

Fixed in 7bea6181

## 2.4 Low

L-1	Unbounded <code>lockUntil</code> Parameter Allows Permanent Token Lock		
Severity	Low	Status	Acknowledged

### Description

Both `CLLockerFactory.lock()` and `V2LockerFactory.lock()` accept a `uint32 _lockUntil` argument but do not validate it against a maximum lock duration. Since `uint32` can represent values up to `type(uint32).max`, a user may inadvertently set the unlock time far in the future. Once the locker is created, the protocol provides no mechanism to accelerate or cancel the unlock. Any LP tokens deposited into such a locker become effectively irretrievable for a very long time.

### Recommendation

We recommend introducing a constant (e.g., `MAX_LOCK_DURATION = 4 * 365 days`) and enforcing the following check:

```
require(
    _lockUntil <= block.timestamp + MAX_LOCK_DURATION ||
    _lockUntil == type(uint32).max,
    InvalidLockDuration()
);
```

This check reduces the chance of user errors and forces users who want to lock their liquidity far in the future to explicitly choose the permanent lock option.

### Client's Commentary

*The finding has been acknowledged and will not be fixed. A maximum lock duration can be enforced through the UI, preserving flexibility at the contract level.*

L-2	Missing Pool Validation Lets Anyone Create Lockers for Arbitrary Tokens		
Severity	Low	Status	Fixed in 9b340a16

#### Description

`V2LockerFactory.lock()` accepts a raw `address _pool` parameter and never verifies that the address is an authentic Velodrome V2 pool produced by the authorised `v2Factory`. A user can therefore pass any ERC-20 contract (or even a non-ERC-20 contract) as `_pool` together with an arbitrary `_lp` amount.

Although no vulnerabilities have been detected, validating user supplied addresses remains best practice and reduces the chance of hidden issues surfacing later.

#### Recommendation

We recommend validating that `_pool` is registered in the `v2Factory` during locker creation in `V2LockerFactory.lock()` function.

#### Client's Commentary

Fixed in e151d010

L-3	Potential Spam Lock Creation Without Minimum Liquidity Requirements		
Severity	Low	Status	Acknowledged

#### Description

Both `CLLockerFactory.lock()` and `V2LockerFactory.lock()` functions allow creating lockers without enforcing minimum liquidity requirements. Users can lock positions for other owners with minimal or zero liquidity amount. This enables malicious actors to spam the system by creating numerous lockers with dust amounts, potentially overwhelming user interfaces, degrading performance of off-chain indexing systems, and making it difficult for legitimate users to navigate through meaningful positions. The attack cost is minimal as it only requires small amounts of liquidity and gas fees.

#### Recommendation

We recommend implementing minimum liquidity thresholds in both factory contracts to prevent spam attacks.

#### Client's Commentary

*The finding has been acknowledged and will not be fixed. The view functions in the `LockerFactory` support pagination to prevent degraded performance.*

L-4	Staked Lockers Cannot Be Unlocked Without Prior Unstaking		
Severity	Low	Status	Fixed in 9b340a16

### Description

When a locker has deposited its LP tokens into the gauge, all LP balance sits inside the gauge contract, while the locker's own balance is zero. If the owner tries to unlock the locker the function executes:

```
IERC20(pool).safeTransfer({ to: _recipient, value: _lp });
```

Because the locker has no tokens, the transfer fails and the whole tx reverts. The user is left with a cryptic error and may not understand that they must first call `un stake()`.

The same flaw is present in `CLLocker.unlock()`, where the contract calls `nfpManager.transferFrom()` to move the NFT position while the liquidity remains staked in the gauge. With the position unavailable inside the locker, this transfer likewise reverts and obscures the real cause to the user.

### Recommendation

We recommend adding an explicit guard at the top of `unlock()`:

```
if (staked) revert LockerStaked();
```

or automatically withdrawing from the gauge before transferring:

```
if (staked) {
    gauge.withdraw({ lp: _lp });
    delete staked;
}
```

Either approach prevents the unexpected revert and makes the flow self-explanatory.

### Client's Commentary

Fixed in 7bea6181

L-5	Left-Over Token Dilution During V2 Playground Migration		
Severity	Low	Status	Fixed in 9b340a16

### Description

When `V2LockerFactory.migrate()` is called from the Playground, the `_unlockLP()` function sends both the LP tokens **and** any leftover `token0` / `token1` balances from the locker directly to the pool contract:

```
_unlockLP({
  _locker: _locker,
  _pool: pool,
  _recipient: pool, // leftovers forwarded to the pool
  _owner: Ownable(_locker).owner()
});
```

The pool then executes:

```
(uint256 amount0, uint256 amount1) = IV2Pool(pool).burn({
  to: playground
});
```

Because `balance0` and `balance1` inside the pool now include the leftover tokens, the exiting LP holder receives only a pro-rata share of those leftovers, the remainder is permanently absorbed into the pool and benefits all other LP holders.

```
// Using full balances ensures pro-rata distribution
amount0 = (_liquidity * _balance0) / _totalSupply;
amount1 = (_liquidity * _balance1) / _totalSupply;
```

### Recommendation

We recommend refunding the entire leftover `token0` and `token1` amounts to the locker owner during migration instead of forwarding them to the pool.

### Client's Commentary

Fixed in 691716e3

<b>L-6</b>	Pre-Transferring Tokens Enables Front-Running Attack		
<b>Severity</b>	Low	<b>Status</b>	Fixed in 9b340a16

#### Description

The protocol assumes wallets will interact via EIP-7702, wrapping `approve()`, `safeTransferFrom()`, and the `lock()` call into one atomic transaction. If a user does not use this flow (e.g., they pre-transfer LP tokens to the factory and only then call `lock()`), a front-runner can observe the pending tx, call `lock()` first, and capture the tokens. The victim's transaction then reverts or creates an empty locker while their funds are stolen.

#### Recommendation

We recommend implementing a pull-based pattern where `lock()` performs the `safeTransferFrom()` itself – allowing users to batch `approve() + lock()` in a single wallet action and eliminating the vulnerable intermediate state.

#### Client's Commentary

Fixed in the following commits:

- [33682424](#)
- [25d13814](#)

<b>L-7</b>	Fees Remain on Locker When <code>bribeAmount</code> Is Zero		
<b>Severity</b>	Low	<b>Status</b>	Fixed in 9b340a16

### Description

Inside `Locker.bribe()`, the unstaked branch executes:

```
(uint256 claimed0, uint256 claimed1) = _collectFees();
uint256 bribeAmount0 = _calculatePercentage(claimed0, _percentage);
uint256 bribeAmount1 = _calculatePercentage(claimed1, _percentage);

if (bribeAmount0 > 0) {
    _bribe(...);
    IERC20(token0).safeTransfer(msg.sender, claimed0 - bribeAmount0);
}
if (bribeAmount1 > 0) {
    _bribe(...);
    IERC20(token1).safeTransfer(msg.sender, claimed1 - bribeAmount1);
}
```

When `bribeAmount0 == 0` and / or `bribeAmount1 == 0`, `_bribe` is correctly skipped, but the corresponding `safeTransfer()` is also skipped, leaving the claimed fees in the locker until unlock.

### Recommendation

We recommend transferring the `claimed` amounts to the owner, regardless of the bribe amount:

```
IERC20(token0).safeTransfer(msg.sender, claimed0 - bribeAmount0);
```

should be moved outside the `if (bribeAmount0 > 0)` block (same for token1).

### Client's Commentary

Fixed in [ec29a0b2](#)

L-8	Rounding Error Accumulates Against Beneficiary When Deducting Share		
Severity	Low	Status	Acknowledged

### Description

In `CLLocker._collectRewards()` and `CLLocker._collectFees()`:

- `CLLocker.sol#L147`
- `CLLocker.sol#L131-L132`

as well as `V2Locker._collectRewards()` and `V2Locker._collectFees()`:

- `V2Locker.sol#L122-L123`
- `V2Locker.sol#L138`

the beneficiary share is computed via integer division in `Locker._deductShare()`, which rounds down:

```
share = (_amount * beneficiaryShare) / MAX_BPS;
```

This truncation causes small residual amounts to be systematically retained by the locker and not sent to the beneficiary, accumulating error over time in favor of the locker and against the beneficiary.

In the extreme, frequent calls on small amounts can result in persistent rounding to zero, effectively starving the beneficiary and yielding no rewards to them.

### Recommendation

We recommend tracking and carrying forward rounding remainders per token to ensure the beneficiary eventually receives the exact pro-rata amount. For example, accumulate the remainder `( _amount * beneficiaryShare ) % MAX_BPS` in a per-token variable and add it to the next calculation, paying out when it crosses 1 unit.

### Client's Commentary

*The issue has been acknowledged and will not be fixed as the rounding error is negligible.*

L-9	Unused and Redundant Imports		
Severity	Low	Status	Fixed in 9b340a16

#### Description

Multiple files include unused imports or redundant `using` directives, which increase bytecode size marginally (when linked libraries are pulled in) and reduce readability/maintainability. Examples:

- `src/LockerFactory.sol`: unused `SafeERC20` and `IERC20` imports
- `src/extensions/v2/V2Locker.sol`: unused `IV2LockerFactory` import
- `src/extensions/v2/V2LockerFactory.sol`: unused `EnumerableSet` import and redundant `using` directive
- `src/extensions/cl/CLLockerFactory.sol`: unused `EnumerableSet` import and redundant `using` directive

#### Recommendation

We recommend removing unused imports and redundant `using` directives in the listed files to improve clarity and avoid accidental coupling.

#### Client's Commentary

Fixed in c994c4aa

<b>L-10</b>	Potential Revert on Zero Bribe Amount		
<b>Severity</b>	Low	<b>Status</b>	Fixed in 9b340a16

### Description

In `Locker.bribe()`, when the locker is staked, the function computes `bribeAmount` and calls `Locker._bribe()` unconditionally. If `bribeAmount` equals zero (for example, when `_percentage` is zero, rounding down to zero, or no rewards were claimed), the downstream `IBribeVotingReward.notifyRewardAmount()` call reverts because it rejects zero amounts.

```
uint256 claimed = _collectRewards();

uint256 bribeAmount = _calculatePercentage({
    _amount: claimed,
    _percentage: _percentage
});

_bribe({
    _briber: briber,
    _token: rewardToken,
    _amount: bribeAmount
});
```

### Recommendation

We recommend avoiding calls to `_bribe()` with a zero amount. This can be achieved by either:

- Adding a guard: `if (bribeAmount > 0) { _bribe(...); }` in the staked branch; or
- Making `_bribe()` a no-op for `_amount == 0` (early return in `_bribe()` function).

### Client's Commentary

Fixed in [ec29a0b2](#)

<b>L-11</b>	Unnecessary Zero Token Transfers in Fee Claims		
<b>Severity</b>	Low	<b>Status</b>	Fixed in 9b340a16

#### Description

In the `Locker._claimFees()` function, the contract performs token transfers even when the claimed amounts are zero. This results in unnecessary gas consumption and emits transfer events for zero amounts.

#### Recommendation

We recommend adding zero checks before performing transfers to avoid unnecessary operations.

#### Client's Commentary

Fixed in the following commits:

- [ec29a0b2](#)
- [9b340a16](#)

<b>L-12</b>	Missing Reentrancy Protection in CLLocker.increaseLiquidity()		
<b>Severity</b>	Low	<b>Status</b>	Fixed in 9b340a16

#### Description

The `CLLocker.increaseLiquidity()` function lacks the `nonReentrant` modifier, while other critical functions in the contract properly implement reentrancy protection. This function performs multiple external calls including token transfers via `_fundLocker()`, interactions with the gauge contract for unstaking/restaking, and calls to the NFT position manager for increasing liquidity. The absence of reentrancy protection could allow malicious ERC-20 contracts to re-enter the function during token transfers, potentially leading to unexpected state changes or exploitation of the staking/unstaking logic.

#### Recommendation

We recommend adding the `nonReentrant` modifier to the `increaseLiquidity()` function to maintain consistency with other state-changing functions and prevent potential reentrancy attacks.

#### Client's Commentary

Fixed in f37b0756

# 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](https://github.com/mixbytes/audits_public)



[hello@mixbytes.io](mailto:hello@mixbytes.io)



<https://x.com/mixbytes>