Security Assessment

Final Report

LP Oracles

August 2025

Prepared for Balancer

certora Balancer

Table of content

certora Balancer

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Balancer	balancer-v3	d32067a7	EVM

Project Overview

This document describes the verification of **LP Oracles** using manual code review. The work was undertaken from **11 August** to **19 August**.

The following contract list is included in our scope:

```
contracts/LPOracleBase.sol
contracts/LPOracleFactoryBase.sol
contracts/StableLPOracle.sol
contracts/StableLPOracleFactory.sol
contracts/WeightedLPOracleBase.sol
contracts/WeightedLPOracleFactory.sol
```

Protocol Overview

The LP oracle contracts provide functionality to create price oracles tailored for Balancer LP tokens by aggregating Chainlink price feeds from the underlying pool assets and applying pool-specific math (weighted or stable) to compute an LP price per token based total value locked for the pool.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	2	2	1
Low	5	5	4
Informational	0	0	0
Total	7	7	5

Severity Matrix

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

Detailed Findings

ID	Title	Severity	Status
M-01	Rate provider staleness cannot be validated	Medium	Acknowledged
M-02	Price freshness cannot be validated for low heartbeat price feeds	Medium	Fixed
L-01	Lack of price validation can corrupt weightedOracle TVL calculation	Low	Fixed
L-02	Lack of Access Control can lead to state pollution	Low	Acknowledged
L-03	StableOracle math breaks on extreme price drops	Low	Fixed
L-04	Missing sequencer uptime check for L2 deployment	Low	Fixed
L-05	Missing order checks can lead to faulty oracle	Low	Fixed

Medium Severity Issues

M-01 Rate provider staleness cannot be validated		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: LPOracle#188	Status: Acknowledged	

Description:

Per Balancer's documentation, there are two types of rate providers that can be used:

- Direct Balance Query - querying the current balance of both assets in the corresponding vault
- Oracles - using Chainlink's oracle to fetch the rate.

The following implementation for an oracle rate provider is provided in the **Example** section:

```
JavaScript
 function getRate() external view override returns (uint256) {
 (, int256 price, , , ) = _feed.latestRoundData(); require(price > 0, "Invalid price rate
response"); return uint256(price) * _scalingFactor;
}
```

Here, there is no validation of the `updatedAt` parameter. Furthermore, it is never returned by the `getRate` function.

Currently, the `LPOracleBase` will leave it to the integrating contracts to validate the `updatedAt` timestamp by returning the oldest update and letting them perform a standard staleness check:

```
JavaScript
function latestRoundData() external view returns (uint80 roundId, int256 answer, uint256
startedAt, uint256 updatedAt, uint80 answeredInRound) {
```

Internally, `calculateTVL()` in both `WeightedLPOracle` and `StableLPOracle` calls `getPoolTokenInfo()`, which invokes `_loadPoolData()`:

```
JavaScript
 function _loadPoolData(address pool, Rounding roundingDirection) internal view returns
(PoolData memory poolData) { poolData.load( _poolTokenBalances[pool], _poolConfigBits[pool],
_poolTokenInfo[pool], _poolTokens[pool], roundingDirection ); }
```

This function will rely on the `rateProvider.getRate()` to fetch the `balancesLiveScaled18`.

However, the updatedAt timestamp from the rate provider is never included in the final minimum updatedAt timestamp retrieval, preventing integrating protocols from validating it.

Recommendations: Consider returning the `updatedAt` variable from the rate provider and including it in the conservative timestamp check.

Alternatively, due to the low likelihood, this issue can be acknowledged and the behaviour can be documented.

Customer's response: This will be analysed on a case by case basis. Changing the rate provider API is probably not an option in this case, and rate providers that already use chainlink market price feeds are probably not going to be a priority use case.

Fix Review: Acknowledged.

M-02 Price freshness cannot be validated for low heartbeat price feeds

Severity: **Medium**	Impact: **Medium**	Likelihood: **High**
Files: LPOracleBase.sol#188	Status: Fixed	

Description:

In LPOracleBase.getFeedData(), the updatedAt value returned is set to the minimum timestamp among all Chainlink feeds:

```rust
updatedAt = updatedAt < feedUpdatedAt ? updatedAt : feedUpdatedAt;
```

This approach is not suitable for integrating protocols as assets have different volatility and frequency of updates.

For example, in one of the pools, the underlying assets are ETH and USDC. While ETH is updated frequently due to its volatility, USDC may update every 24 hours, so they would both require different staleness periods. However, the latestRoundData() will only return the oldest updatedAt, which cannot be used for validation of both assets.

Exploit Scenario:

- Feed 1: 1 hour heartbeat
- Feed 2: 24 hour heartbeat

When queried, the getFeedData() will, during normal operations, return an updatedAt value between 0-24 hours. Which is the normal price update timing for feed 2. If an integrating protocol wants to check the freshness of the prices for Feed 1, setting a threshold of 1 hour will effectively break the oracle since the updatedAt would be the one for Feed 2 and would thus revert almost every time.

Recommendations: In order to give protocols more flexibility, consider returning an array of all the updatedAt timestamps.

Customer's response: Resolved in PR1498

Fix Review: The getFeedData function has been updated to now return an array with all the updatedAt values from all the feeds. This allows any user to implement staleness checks for each feed. However, this array is not used in the latestRoundData() function which only returns the minUpdatedAt renamed to updatedAt. Additional explanation was provided in natSpec in PR1548 to avoid user confusion.

Low Severity Issues

L-01 Lack of price validation can corrupt weightedOracle TVL calculation		
Severity: **Low**	Impact: **High**	Likelihood: **Low**
Files: WeightedLPOracle#L119	Status: Fixed	

Description:

The `weightedOracle` consists of up to 8 price feeds linked to a pool. However, there is no price validation whatsoever on any of the price feeds. This can be problematic since a malfunctioning price feed can return a `0` value and in the case of RWA price feeds (futures), a negative value can be returned.

If we examine `calculateTVL` function in `WeightedOracle`, we find:

```
Rust
for (uint256 i = 0; i < _totalTokens; i++) {
     tvl = tvl.mulDown(prices[i].toUint256().divDown(weights[i]).powDown(weights[i]));
}
```

- If a `0` value is returned by any of the price feeds, the `tvl` returned will be `0`.
- If a `negative` value is returned, the `tvl` will revert due to casting to uint.

Recommendations:

We recommend including an answer `> 0` check for every price feed in a WeightedOracle.

Customer's response: Resolved in PR1509

Fix Review: Any price<=0 will now revert, which resolves the issue.

L-02 Lack of Access Control can lead to state pollution		
Severity: **Low**	Impact: **High**	Likelihood: **Low**
Files: LPOracleFactoryBase#43	Status: Acknowledged	

Description:

The create function in LPOracleFactoryBase has zero access control, which can allow for state pollution by griefers.

When an integrating protocol decides to deploy a stable oracle for its users. In doing so, it will also deploy an oracle factory, which can then be used by anyone to create any legitimate or erroneous oracle they want.

It is not possible to block the deployment of correct oracles, but it is possible to spam and fill up the mapping with incorrect oracles.

This can lead to confusion for users utilising the integrating protocol since the malicious oracles are also deployed from the official factory contract of the integrating protocol.

Recommendations:

Minimal access control should be implemented so that the deployer of the factory contract can decide who can deploy oracles.

Customer's response: Acknowledged.

Fix Review: Acknowledged.

L-03 StableOracle math breaks on extreme price drops		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: StableLPOracle#57	Status: Fixed	

Description:

The `StableOracle` uses complex math to calculate the `TVL`. If a price feed reports extremely low prices, these math operations become dysfunctional.

- When `price < 1e8`, the calculations will revert due to division by zero.
- When `1e8 < price < 1e12`, the calculations will revert due to K non-convergence.

However, in such a case the actual `stablePool` is likely also dysfunctional due to the extreme value difference in tokens within the pool.

Recommendations:

It is possible to include a price check for every feed which enforces that the price is higher than 1e12 or it will revert with “Oracle Failure”.

Alternatively, if the gas cost for this is deemed too high, it could also be a good option to clearly document the cases in which the StableOracle is expected to revert.

Customer’s response: Resolved in PR1530

Fix Review: The client added graceful failures for values that would produce issues and added normalisation of prices which further reduces chances of the K calculations going awry.

L-04 Missing sequencer uptime check for L2 deployment		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: LPOracleBase#176	Status: Fixed	

Description:

Balancer is deployed on multiple L2s, which means that the oracles will also be used on those chains. However, the getFeedData is missing a check for sequencer uptime check:

```
Rust
 function getFeedData() public view returns (int256[] memory prices, uint256 updatedAt) {
        uint256 totalTokens = _totalTokens;
        AggregatorV3Interface[] memory feeds = _getFeeds(totalTokens);
        uint256[] memory feedDecimalScalingFactors =
_getFeedTokenDecimalScalingFactors(totalTokens);

        prices = new int256[](totalTokens);

        updatedAt = type(uint256).max;
        for (uint256 i = 0; i < totalTokens; i++) {
            (, int256 answer,, uint256 feedUpdatedAt,) = feeds[i].latestRoundData();
```

`LPOracleBase.latestRoundData()` calls external Chainlink price feeds but does not verify that the L2 sequencer is operational.

On L2 rollups, Chainlink recommends integrating the **Sequencer Uptime Feed** to detect:

- When the sequencer is offline (status = 1)
- The initial "grace period" after recovery, when prices can still be stale or inconsistent due to delayed state sync

Without this, the oracle may return **incorrect LP prices** during sequencer downtime or immediately after recovery, potentially causing issues.

Recommendation: It is recommended to implement a sequencer uptime check in the LPOracleBase contract for L2 deployments by adding a function to verify the sequencer status.

Customer's response: Resolved in PR1521

Fix Review: The sequencer uptime check was correctly introduced in all oracles

L-05 Missing order checks can lead to faulty oracle		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: LPOracleBase#54	Status: Fixed	

Description:

When an Oracle is deployed, an array of AggregatorV3Interface is given to correspond to the tokens in the tokenPool. We can see in the constructor that there is a check to ensure that the length or the feeds array is the same as the totalTokens.

However, there is no check to ensure that the AggregatorV3 feed and the token actually correspond.

Assume we have:

- tokenPool: USDC, USDT, DAI
- feeds: USDT, USDC, DAI

The code will incorrectly map the price feeds with the wrong tokens. The deployed Oracle WILL function, there is no revert or error anywhere due to the mismatch, but the returned TVL values will be wildly incorrect.

Recommendation:
Mitigation is non-trivial since the price feeds themselves do not have the token information. This would most likely require a centralised feed registry where feeds are linked to tokens and then the feeds can be set automatically.

It could also be an option to clearly document this behavior in NatSpec and protocol docs to ensure that users are aware of the risks.

Customer's response: Resolved in PR1508

Fix Review: Natspec was added which clearly stresses the importance of correct ordering.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.