Security Assessment
Final Report

Liquidity Bootstrap Pool

February 2025

Prepared for Balancer Labs

certora Balancer

Table of content

certora Balancer

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Balancer V3	balancer-v3-monorepo	12f2c3d	EVM

Project Overview

This document describes the verification of **Liquidity Bootstrap Pool** using manual code review. The work was undertaken from **10 February 2025** to **17 February 2025**.

The following contract list is included in our scope:

```
pkg/pool-weighted/contracts/lbp/LBPool.sol
pkg/pool-weighted/contracts/lbp/LBPoolFactory.sol
pkg/pool-weighted/contracts/lib/LBPoolLib.sol
pkg/pool-weighted/contracts/lib/GradualValueChange.sol
```

During the manual audit, the Certora team discovered issues in the Solidity contracts code, as listed on the following page.

Protocol Overview

The Liquidity Bootstrap Pool enables Balancer pools to bootstrap liquidity for token sales. The protocol offers the ability to create pools in a permissionless manner, enabling pool creators to configure the duration of the sale and weights to suit their needs. Liquidity for the project token is raised in a blue-chip reserve token to ensure reliable price discovery based on the start and end weights defined by the pool owner.

Findings Summary

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

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	-	-	-
Low	-	-	-
Informational	4		
Total	4		

Severity Matrix

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

Detailed Findings

Informational Severity Issues

I-01. Event GradualWeightUpdateScheduled emits incorrect startTime

Description: In the constructor of the `LBPool` contract, the start time of the bootstrap event is retrieved from the `lbpParams` parameter, which is utilized for the emission of the `GradualWeightUpdateScheduled` event.

```
Unset
emit GradualWeightUpdateScheduled(lbpParams.startTime, lbpParams.endTime, startWeights,
endWeights);
```

However, the actual start time of the event can be different from `lbpParams.startTime` since function `resolveStartTime()` updates the `startTime` to `block.timestamp` if the `startTime` is in the past.

```
Unset
function resolveStartTime(uint256 startTime, uint256 endTime) internal view returns (uint256
resolvedStartTime) {
        // If the start time is in the past, "fast forward" to start now
        // This avoids discontinuities in the value curve. Otherwise, if you set the
start/end times with
        // only 10% of the period in the future, the value would immediately jump 90%
        resolvedStartTime = Math.max(block.timestamp, startTime);

        if (resolvedStartTime > endTime) {
            revert GradualUpdateTimeTravel(resolvedStartTime, endTime);
        }
    }
```

This would lead to the event `GradualWeightUpdateScheduled` emitting incorrect data, potentially misleading off-chain consumers (such as frontends) and the `LBPool` owner as to what the actual start time for the bootstrap event is.

Recommendation: Use the state variable `_startTime` in the event emission instead of `lbpParams.startTime`.

Customer's response: Fixed in 02de03d

I-02. Incorrect validation for startTime/endTime as per specification

Description: The natspec comment from `LBPoolLib.sol` mentions that the `endTime` should be strictly greater than the `actualStartTime`:

```
Unset
/**
 * @dev Normalize `startTime` to block.now (`actualStartTime`) if it's in the past, and
verify that
 * `endTime` > `actualStartTime` as well as token weights.
 */
```

However, function `resolveStartTime()` in `GradualValueChange.sol` does not implement this strict validation. This means it is possible for the `endTime` to be equal to the `startTime`.

```
Unset
function resolveStartTime(uint256 startTime, uint256 endTime) internal view returns (uint256
resolvedStartTime) {
    // If the start time is in the past, "fast forward" to start now
    // This avoids discontinuities in the value curve. Otherwise, if you set the start/end
times with
    // only 10% of the period in the future, the value would immediately jump 90%
    resolvedStartTime = Math.max(block.timestamp, startTime);

    if (resolvedStartTime > endTime) {
        revert GradualUpdateTimeTravel(resolvedStartTime, endTime);
    }
}
```

While this does not pose a risk in `LBPool.sol`, we should stick to the specification to avoid any issues in the future with the use of the libraries `LBPoolLib.sol` and `GradualValueChange.sol`.

Recommendation: Use `resolvedStartTime >= endTime` instead of `resolvedStartTime > endTime`.

Customer's response: Fixed in 02de03d

I-03. _projectTokenIndex and _reserveTokenIndex are not exposed

Description: Some external viewer functions, such as `getGradualWeightUpdateParams()` and `getLBPoolImmutableData()`, use `_projectTokenIndex` and `_reserveTokenIndex` to represent the corresponding token indexes in the returned start and end weight arrays. However, these indexes are never exposed (both of them are defined as `uint256 private immutable`).

This makes the contract ABI a bit unfriendly to use/integrate with since integrators will need to calculate the indexes themselves.

Recommendation: Expose the `_projectTokenIndex` and `_reserveTokenIndex` via external viewer functions.

Customer's response: Fixed in 51c5291.

I-04. Optimize redundant boolean comparison in function onSwap()

Description: Function `onSwap()` in `LBPool.sol` ensures that swaps can only occur between the `startTime` and `endTime` of the bootstrap event. The if condition at the start of the function ensures this.

While there is no logical issue with the current condition, it can be optimized by simply negating the returned boolean value from the `_isSwapEnabled()` function call. This would save gas on swaps since comparing to a constant (`true` or `false`) is a bit more expensive than directly checking the returned boolean value.

```
Unset
function onSwap(
```

```
    PoolSwapParams memory request
) public view override(IBasePool, WeightedPool) onlyVault returns (uint256) {
    // Block if the sale has not started or has ended.
    if (_isSwapEnabled() == false) {
        revert SwapsDisabled();
    }
}
```

Recommendation: Optimize `if (_isSwapEnabled() == false)` to `if (!_isSwapEnabled()).`

Customer's response: Acknowledged. This is by design.

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.