Electisec Pessimistic Velodrome LP Oracle Review

Review Resources:

- Repository README

Auditors:

- Fedebianu
- Adriro

Table of Contents

Review Summary

Oracle

The Pessimistic Velodrome LP Oracle can be used to price Velodrome Liquidity Pool (LP) tokens, optionally wrapped in a Yearn V2 or V3 vault. It adapts the concept of *fair reserves*, developed by Alpha Homora, to price both volatile and stable pools, combined with the idea of *pessimistic* pricing implemented in the Inverse Finance FiRM oracle.

The contracts of the Pessimistic Velodrome LP Oracle repository were reviewed over three days. Two auditors performed the code review between April 30th and May 2nd, 2025. The review was limited to the latest commit at the start of the engagement, which is revision `575ac4cd226fae22a69bddb945fb45700c68ee83`.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
contracts
├── FixedPointMathLib.sol
├── PessimisticVeloSingleOracle.sol
└── ShareValueHelper.sol
```

After the findings were presented to the HAI team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

Electisec and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. Electisec and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By

deploying or using the code, HAI and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control rules are in place.
Mathematics	Good	Mathematical operations are correctly implemented.
Complexity	Good	The codebase is straightforward and easy to follow.
Libraries	Good	The implementation uses the OpenZeppelin library and a copy of the fixed point utilities from Solmate.
Decentralization	Average	While the state is mostly immutable, the contract relies on an operator to record daily lows.
Code stability	Good	The codebase remained stable throughout the review.
Documentation	Good	The *readme* includes a good description of the design and mathematical concepts applied in the Oracle, with proper references to its sources.
Monitoring	Good	Appropriate events are emitted for off-chain monitoring.
Testing and verification	Average	While the original implementation has an extensive set of tests, these were not ported to the new ”single” variant of the Oracle.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.

- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Low Findings

1. Low - Missing grace period check-in Chainlink L2 Sequencer

Chainlink recommends implementing a grace period after the sequencer becomes available to allow a time margin to process queued messages.

Technical Details

The implementation of `getChainlinkPrice()` checks if the L2 sequencer is up, but doesn't validate if it has been recently started.

Impact

Low.

Recommendation

Consider implementing the check as recommended by Chainlink.

Developer Response

Fixed in 5a5492802dcf9ea2791e7e10335a14eae16297aa and acbd4abdfff2646fc80eb117611ad733c1a79386.

2. Low - Incorrect check in the constructor

The `PessimisticVeloSingleOracle` constructor incorrectly checks an uninitialized state variable instead of the constructor parameter, which prevents setting an Oracle feed for the second token.

Technical Details

In the `PessimisticVeloSingleOracle` constructor, there is an incorrect check in the `else if` condition:

```
} else if (token1Feed != address(0)) {  // @audit using uninitialized state variable
    token1Feed = _token1Feed;
    token1Heartbeat = _token1Heartbeat;
    if (IChainLinkOracle(_token1Feed).decimals() != 8) {
        revert("Must be 8 decimals");
    }
}
```

The code checks `token1Feed` instead of `_token1Feed`. Since `token1Feed` is a state variable, it will be `address(0)` at deployment time, causing the condition to evaluate to `false` continually, and the deployment will always revert.

Impact

Low.

Recommendation

Replace the incorrect check with the constructor parameter.

Developer Response

Fixed in 4ef1e0feb8dda7f9778b8b203d90e2a5e81e2c46.

Gas Saving Findings

1. Gas - Use custom errors instead of error strings

Technical Details

`PessimisticVeloSingleOracle.sol` use error strings for reverts alternating both `require` and `revert`.

This approach is less gas efficient compared to using custom errors.

Impact

Gas savings.

Recommendation

Implement custom errors for all revert cases to save gas and have a cleaner code.

Developer Response

Fixed in 3a6f820a791480b0c008572fe93d9278142a69c1 and 3f13aaf31d265bb91141a4eb3c4d4604f33f9159.

2. Gas - Move pool data as immutable variables

Different configuration values from the pool can be stored as immutable variables to save gas.

Technical Details

The following properties from the pool can be stored as immutable variables during contract construction:

- `token0`
- `token1`
- `decimals0`
- `decimals1`
- `stable` flag

This would allow to remove the call to `poolContract.metadata()` in `getTokenPrices()` and also change this call in `_getFairReservesPricing()` to `poolContract.getReserves()`.

Impact

Gas savings.

Recommendation

Move the listed pool config values to immutable variables. Note that `token0` and `token1` are already present as immutable variables but are not initialized nor used.

Developer Response

Fixed in da664deda8235ebfb74d53a7f54584ff0039ba92.

3. Gas - LP decimals check can be moved to the constructor

The implementation of `_getFairReservesPricing()` checks that the LP decimals are 18 every time this function is called.

Technical Details

PessimisticVeloSingleOracle.sol#L424-L431

```
424:     function _getFairReservesPricing(
425:         address _pool
426:     ) internal view returns (uint256 fairReservesPricing) {
427:         // get what we need to calculate our reserves and pricing
428:         IVeloPool poolContract = IVeloPool(_pool);
429:         if (poolContract.decimals() != 18) {
430:             revert("Lp token must have 18 decimals");
431:         }
```

Impact

Gas savings.

Recommendation

Move this check to the contract constructor.

Developer Response

Fixed in 3a6f820a791480b0c008572fe93d9278142a69c1.

4. Gas - Simplify TWAP calculation

The math to get the TWAP price based on the other token's price can be simplified.

Technical Details

In `getTokenPrices()`, TWAPs are calculated as following:

```
341:                 // get twap price for token1. this is the amount of token1 we
would get from 1 token0
342:                 price1 =
343:                     ((decimals1 * decimals1) / 100) /
344:                     getTwapPrice(_token0, decimals0 / 100); // returned in
decimals1
345:                 price1 = (price0 * price1) / (decimals1);
```

The expanded calculation is:

```
price1 = price0 * (((decimals1 * decimals1) / 100) / getTwapPrice(_token0, decimals0
/ 100)) / decimals1
```

Which can be simplified as:

```
price1 = price0 * decimals1 / getTwapPrice(_token0, decimals0)
```

Impact

Gas savings.

Recommendation

TWAP calculations can be simplified using the following expressions:

```
price0 = price1 * decimals0 / getTwapPrice(_token1, decimals1)
price1 = price0 * decimals1 / getTwapPrice(_token0, decimals0)
```

If lower precision in TWAP is needed, then this could be simplified as:

```
price0 = price1 * decimals0 / (getTwapPrice(_token1, decimals1 / 100) * 100)
price1 = price0 * decimals1 / (getTwapPrice(_token0, decimals0 / 100) * 100)
```

Developer Response

Fixed in ef3b300953a0021aa479168df803e6d2570d5a71.

Informational Findings

1. Informational - Flag `useChainlinkOnly` can be removed

The `useChainlinkOnly` flag, which checks the presence of both Chainlink feeds, can be removed in the single-pool version of the Oracle.

Technical Details

The `useChainlinkOnly` flag is not needed as the contract will be initialized with one or both Chainlink feeds, and further checks are not required.

Impact

Informational.

Recommendation

Remove the `_useChainlinkOnly` constructor parameter and the `useChainlinkOnly` immutable variable. Additionally, the initialization of the contract can be significantly simplified by just assigning the variables and then checking for the presence of at least one Chainlink Oracle.

```
require(_token0Feed != address(0) || _token1Feed != address(0), "At least one
Chainlink Oracle is required");

if (_token0Feed != address(0) && IChainLinkOracle(_token0Feed).decimals() != 8) {
    revert("Must be 8 decimals");
}

if (_token1Feed != address(0) && IChainLinkOracle(_token1Feed).decimals() != 8) {
    revert("Must be 8 decimals");
}

token0Feed = _token0Feed;
token0Heartbeat = _token0Heartbeat;
token1Feed = _token1Feed;
token1Heartbeat = _token1Heartbeat;
```

Developer Response

Fixed in b35e176cd38930db678b476f7f2b2ece006b3dcf.

Final remarks

The Pessimistic Velodrome LP Oracle successfully combines Alpha Homora's *fair reserves* calculations with Inverse Finance's *pessimistic* pricing to address flash loan manipulation vulnerabilities in LP token pricing. The implementation demonstrates solid mathematical foundations and proper Chainlink integration with appropriate security checks. The fair reserves formula is correctly applied to both volatile and stable pools, adapting the code to account for the different invariants used by each pool type. No major security issues were found during the review.