yAudit Euler Price Oracles Update Review

Review Resources:

- Balancer Rate Providers documentation

Auditors:

- HHK
- Adriro

Table of Contents

Review Summary

Euler Price Oracles

Euler Price Oracles (EPO) is a library of modular oracle adapters and components that implement IPriceOracle, an opinionated quote-based interface.

The current review focuses on three new adapters: LidoFundamentalOracle.sol, FixedRateOracle.sol, and RateProviderOracle.sol.

The contracts of the EPO repository were reviewed over two days. Two auditors performed the code review between September 5th and September 6th, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start. This was commit 8c4aeedd12b987543498b772a081e8cd261072be for the EPO repository.

Scope

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

```
src/adapter/lido/
└── LidoFundamentalOracle.sol
src/adapter/fixed
└── FixedRateOracle.sol
src/adapter/rate
├── IRateProvider.sol
└── RateProviderOracle.sol
```

After the findings were presented to the Euler 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.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit 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, Euler and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access control is not required for the adapters.
Mathematics	Average	A decimal handling issue could lead to incorrect oracle results.
Complexity	Good	The contracts for the adapters are well-structured, straightforward, and easy to read.
Libraries	Good	The reviewed adapters do not depend on external libraries.
Decentralization	Good	Adapters are final; their configurations are immutable.
Code stability	Good	The codebase remained stable during the review.
Documentation	Average	There were some areas where NatSpec comments could have been improved.
Monitoring	Good	Monitoring is provided by the parent Router.
Testing and verification	Average	Despite having multiple tests, the decimal handling issue in the rate provider oracle was missed.

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.

Critical Findings

None.

High Findings

1. High - Scale is incorrectly calculated in RateProviderOracle.sol

Decimals are incorrectly handled when calculating the scale between the assets and the rate provider.

Technical Details

The RateProviderOracle.sol contract uses Balancer's rate providers to convert between the base and quote assets. These rate providers return results in 18 decimals because Balancer natively uses 18 decimals for internal accounting.

The adapter implementation is aware of this, but it builds the scale using 18 as the base asset's decimals, ignoring the actual number of decimals for the base asset.

```
34:         // Since Balancer uses 18 decimals for internal accounting, we override base
decimals to 18.
35:         scale = ScaleUtils.calcScale(18, quoteDecimals, quoteDecimals);
```

The arguments to `calcScale()` should be base decimals, quote decimals, and feed decimals, the latter being used to interpret the provider's conversion rate.

For example, the following test in the codebase expects to convert between XAUt, which has six decimals, to USD.

```
183:     function test_GetQuote_XAUT() public {
184:         RateProviderOracle oracle = new RateProviderOracle(XAUT, USD,
BALANCER_XAUT_RATE_PROVIDER);
185:         uint256 rate = 2522e18;
186:
187:         uint256 outAmount = oracle.getQuote(1e18, XAUT, USD);
188:         uint256 outAmount1000 = oracle.getQuote(1000e18, XAUT, USD);
189:         assertApproxEqRel(outAmount, rate, REL_PRECISION);
190:         assertEq(outAmount1000, outAmount * 1000);
191:
192:         uint256 outAmountInv = oracle.getQuote(outAmount, USD, XAUT);
193:         assertEq(outAmountInv, 1e18);
194:         uint256 outAmountInv1000 = oracle.getQuote(outAmount1000, USD, XAUT);
195:         assertEq(outAmountInv1000, 1000e18);
196:     }
```

We can see that the input amounts are incorrectly expressed using 18 decimals. Lines 187 and 188 define inputs using a scale of `10**18` when the XAUt unit is `10**6`. The test priced `1_000_000_000_000` units of XAUt at the current price of gold ($2522).

Impact

High. The oracle will return incorrect prices due to decimal mishandling.

Recommendation

When calculating the scale via 'calcScale ()', the rate provider precision should be used as the feed decimals.

```
    constructor(address _base, address _quote, address _rateProvider) {
        base = _base;
        quote = _quote;
        rateProvider = _rateProvider;
+       uint8 baseDecimals = _getDecimals(base);
        uint8 quoteDecimals = _getDecimals(quote);
-       // Since Balancer uses 18 decimals for internal accounting we override base
decimals to 18.
-       scale = ScaleUtils.calcScale(18, quoteDecimals, quoteDecimals);
+       // Since Balancer uses 18 decimals for internal accounting we override feed
decimals to 18.
+       scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, 18);
    }
```

Developer Response

Fixed in https://github.com/euler-xyz/euler-price-oracle/pull/61.

Medium Findings

None.

Low Findings

1. Low - Delayed Lido Oracle update might result in inaccurate pricing

Technical Details

The Lido Oracle is in charge of reporting the Consensus Layer ether balance update every 24 hours. If the balance increases, the steth will rebase positively, while if it decreases because of a slashing, it will rebase negatively.

The Lido oracle documentation describes cases in which the Oracle could fail to update or be delayed.

In such case, the LidoFundamentalOracle may fail to report accurate pricing as the `_getQuote()` will return the last rate.

In the past, there have been events where the Oracle was delayed by 6 hours. There have also been cases of slashing, but they haven't been significant enough to lead to negative rebase.

While it is very unlikely that both happen at the same time, they could lead to less accurate pricing than a more classic oracle reporting secondary market prices, as these would be faster to reflect the price of `steth` following an incident.

Impact

Low.

Recommendation

Consider using a classic Chainlink price feed, or make sure to use safe enough parameters to handle a slashing event and/or a delay in the rebase without creating bad debt.

Developer Response

Acknowledged, no fix.

Gas Saving Findings

None.

Informational Findings

1. Informational - Document the scale of the rate parameter in FixedRateOracle.sol

The rate parameter should be expressed in the quote asset's decimals.

Technical Details

The FixedRateOracle.sol works by configuring a fixed conversion rate between the base and the quote assets.

```
44:         return ScaleUtils.calcOutAmount(inAmount, rate, scale, inverse);
```

The scale is configured as if the rate is interpreted as the *feed*, since the quote asset's decimals are used as the feed decimals.

```
34:         scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, quoteDecimals);
```

Impact

Informational.

Recommendation

Consider adding a comment to document the expected scale of the rate parameter.

Developer Response

Fixed in https://github.com/euler-xyz/euler-price-oracle/pull/61.

Final Remarks

This update to the Euler price oracles introduces three new adapters, enabling additional pricing sources and expanding the capabilities of the Euler Vault Kit. The adapters are simple and efficient, leveraging the libraries used by existing contracts. While a high-severity vulnerability was identified in one of the adapters, the auditors recognize Euler's strong commitment to security, demonstrated by their comprehensive test suite and prompt responses throughout the audit and post-audit remediation processes.