OpenZeppelin | security

Price Oracle Audit

May 17, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-04-08 To 2024-04-24
Languages	Solidity

Total Issues	11 (5 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	2 (0 resolved, 1 partially resolved)
Notes & Additional Information	8 (4 resolved)

Scope

We audited the euler-xyz/euler-price-oracle repository at commit eeb1847.

In scope were the following files:

```
src/
| EulerRouter.sol
├── adapter/
|   ├── BaseAdapter.sol
|   └── CrossAdapter.sol
├── chainlink/
|   ├── AggregatorV3Interface.sol
|   └── ChainlinkOracle.sol
├── chronicle/
|   ├── ChronicleOracle.sol
|   └── IChronicle.sol
├── lido/
|   ├── IStEth.sol
|   └── LidoOracle.sol
├── maker/
|   ├── IPot.sol
|   └── SDaiOracle.sol
├── pyth/
|   ├── PythOracle.sol
├── redstone/
|   ├── RedstoneCoreOracle.sol
├── uniswap/
|   ├── UniswapV3Oracle.sol
├── interfaces/
|   ├── IPriceOracle.sol
└── lib/
|   ├── Errors.sol
|   ├── Governable.sol
|   └── ScaleUtils.sol
```

System Overview

The audited codebase consists of different oracle integrations that are all meant to adhere to the same `IPriceOracle` interface. The `IPriceOracle` interface exposes two functions – `getQuote` and `getQuotes`. Both functions accept an `inAmount` parameter that represents the amount of `base` asset that is meant to be quoted in terms of the `quote` asset. The output of those functions is an `outAmount` for the `getQuote`, and a `bid` and an `ask` `outAmount` for `getQuotes`. The `bid` and `ask` are meant to be two different prices to allow for future use cases that are not yet supported since every oracle integration currently returns the same `bid` and `ask` price. All oracle integrations, apart from exposing the `IPriceOracle` interface, also extend from the `BaseAdapter`.

The `BaseAdapter` inherits the `IPriceOracle` interface. It contains the `getQuote`, `getQuotes`, and the `_getDecimals` function used within any adapter's constructor to setup the proper scaling variables to be used for correct accounting based on priced assets' decimals. The `CrossAdapter` contract is meant to provide chaining of two oracles in a sequence. For example, the Maker oracle provides the exchange rate between `sDAI` and `DAI` but this can be chained with a `DAI` to `USD` into a `CrossAdapter` instance so that the `sDAI-USD` price can be retrieved in one call.

In order to provide integrators with a unique address to be called for several assets' prices, there is the `EulerRouter` contract. This router adheres to the same `IPriceOracle` interface but also offers aggregator functionality that can effectively route the caller's call through the appropriate `PriceOracle` instance. The configuration of all the possible routes is saved into the `oracles` mapping, which links asset pairs to the corresponding `PriceOracle`. In addition, ERC-4626 Euler vaults can be configured as assets being priced, but their configuration is stored in a separate `resolvedVaults` mapping. Finally, if no configuration is found for a given asset pair, the `fallback` oracle will be called, if set. This fallback oracle supposedly is another instance of a second router that might host a different configuration and that can extend the initial router functionality.

Integrated oracles are described below.

Chainlink Price Feeds

The `ChainlinkOracle` contract provides access to Chainlink price feeds through the use of the `latestRoundData` function call which effectively returns the price of a given asset. The price is checked to be positive and not stale based on a `maxStaleness` parameter defined in the constructor. Finally the `outAmount` is returned according to the `inAmount` and the retrieved price.

Chronicle Price Oracle

Similarly, the `ChronicleOracle` contract uses the `readWithAge` call into Chronicle's feed and checks whether the price is non-zero and not stale, again based on a `maxStaleness` parameter. Same as Chainlink's oracle, the `outAmount` is then returned.

Maker sDAI/DAI Price Oracle

The `SDaiOracle` contract returns the amount of `DAI` corresponding to an `inAmount` of `sDAI` or the other way around based on the order of `base` and `quote` parameters passed in the function call. It uses the same logic contained in the `Pot`'s `drip` function to effectively take into account the latest exchange rate between the two assets. The logic is extrapolated and repeated to avoid state changes and keep the function call as `view`. In this case, there's no check nor validation about the value or staleness of the returned exchange rate.

Lido wstETH/sETH Price Oracle

Similar to the Maker contract, the `LidoOracle` contract provides a way to retrieve the exchange rate between `stETH` and `wstETH` through the `Lido` contract. Differently from the Maker oracle, the exchange rate is directly retrieved from the contract, without the need to replicate logic like Maker's `drip` function. Similarly, there is no check of the value or staleness of the exchange rate retrieved from Lido.

Pyth Price Oracle

In the `PythOracle` contract, the price is retrieved through the `getPriceNoOlderThan` call, which is sufficient to guarantee the non-staleness of the price retrieved. Moreover, the price is checked to be non-negative, for its confidence level to sit in between a `maxConfWidth`

parameter, and for the Pyth price exponent returned to stay within a valid range. After proper scaling, the `outAmount` is then returned using the retrieved price.

It is interesting to note that since anyone can update the Pyth oracle's price (with the only condition being that the timestamp in the payload is higher), a user can make this oracle return two different prices in the same transaction.

Redstone Core Price Oracle

The `RedstoneCoreOracle` contract has a slightly different mechanism from others. An `updatePrice` function must be called to update a `cachedPrice` variable which is then used, whenever the `getQuote(s)` function is called. If the `cachedPrice` has not been updated in a `maxCacheStaleness` time frame, the price retrieval will fail. However, it is also worth noticing that within the same time frame, the price cannot be updated, forcing it to stay the same during this period. The extra `validateTimestamp` function is needed from Redstone's integration since it is called within the `updatePrice` execution. In this case, the price is not checked to be non-zero.

Uniswap V3 TWAP Price Oracle

The last oracle integration is the canonical Uniswap v3 TWAP oracle, named `UniswapV3Oracle`. In this oracle, there are several assumptions about the pool used. Namely that it should have enough in-range liquidity, enough observations cardinality, and that the observations used are not stale. The time frame in which the time-weighted average of the price is calculated is defined in the `twapWindow` variable and the `outAmount` is directly calculated through the pool's tick logic, taking into account any possible slippage.

Other Contracts

Apart from the interfaces relative to each individual oracle and the `IPriceOracle` interface, there is the `Errors` contract which defines custom errors used across the contracts, and the `ScaleUtils` library, which is in charge of handling the sorting of asset pairs, scaling, and `outAmount` calculations.

Privileged Roles

In order to set the right configuration in the `EulerRouter` and any other instance of it like the `fallbackOracle`, a `Governable` contract is used which resembles OpenZeppelin's `Ownable` contract. During our conversations with them, the team stated their intention to renounce the ownership of the `EulerRouter` once the configuration has been set. Actions that might be executed exclusively by the `EulerRouter`'s governor are:

- Set the `oracles` mapping to configure specific oracles.
- Set specific ERC-4626 vaults resolutions.
- Set the `fallbackOracle` within the `EulerRouter` contract.

Assumptions

Given all the potential integrations and uses that can be made with the current general-purpose codebase, some assumptions were made by the Euler team and/or the auditors while reviewing it:

- We assume that the lack of standardized use of battle-tested libraries like `Ownable` by the OpenZeppelin library is purely dictated by the gas-saving approach of the team that wanted to minimize code deployed on-chain, given the restricted use that has to be made as part of the administrative decisions. For the same exact reasons that ownership of the router contract is meant to be renounced, we understand that the team did not want to design ownership capabilities with a 2-step transfer mechanism that prevents accidentally losing the ownership of contracts.
- We assume that the plain use of `msg.sender` instead of the `Context` contract provided `_msgSender` is intentional and that the team is aware that it will prevent the use of ERC-2771 capabilities for the governor's actions.
- The lack of input validation within `govSetResolvedVault`, `govSetConfig`, and `govSetFallbackOracle` is intentional given the fact that the `governor` is expected to be a trustworthy actor that provides correct inputs all the time.
- Many adapters' constructors call the `_getDecimals` function to retrieve the number of decimals of the `base` and `quote` assets. Using a non-standard pattern, the `_getDecimals` function might fail when calling `decimals` in the queried asset and instead of failing, will default to assign `18` as the number of decimals to the asset of the failed attempt. We assume this is safely taken into consideration every time an asset is configured and will never result in a misconfiguration of the oracle.

Medium Severity

M-01 Redstone Oracle Adapter Is Flawed

The Redstone adapter works by locally caching the prices received through the Redstone pull-based oracle. The user supplies the off-chain verified price by calling the `updatePrice` function on the adapter. This call is supposed to be part of a batch transaction to the Ethereum Vault Connector. This design makes the RedstoneCore Adapter function like a push-based oracle.

The `updatePrice` function does not let you update the price if `maxCacheStaleness` has not passed since the time of the last update. During times of high volatility, the price cannot be updated in a timely manner opening the doors for:

- Borrowing more of an underpriced asset
- Borrowing more value due to overpriced collaterals
- Not being able to liquidate in a timely manner

Another notable observation is that the adapter can return a price that is `block.timestamp - (maxCacheStaleness + maxPriceStaleness)` old. This can lead to prices being more stalled than assumed.

The impact and consequences of this issue are very high. However, the likelihood is ultimately determined by the `maxCacheStaleness` and `maxPriceStaleness` time periods. If these are short enough, combined with the low likelihood of flash crashes, the overall chances of this being a serious issue are contained.

Consider allowing `updatePrice` to be called when the `block.timestamp` is greater than the `cacheUpdatedAt` and devising a way where the staleness check includes both the `cacheStaleness` and the `priceStaleness`. Alternatively, use the Redstone pull-based oracle as intended without locally caching the values in the `RedstoneCoreOracle` adapter.

Update: *Resolved in pull request #40 at commit b422959.*

Low Severity

L-01 Code Behaves Inconsistently

The codebase incorporates several oracle adapters behind the same `IPriceOracle` interface. However, each specific adapter has its own set of assumptions and differences in the inner mechanics. There are also some inconsistencies between several adapters, specifically when it comes to prices being zero.

- Chainlink and Chronicle oracles prohibit the price from being zero.
- In the Lido oracle, if `inAmount` is small enough, the `outAmount` can be truncated to zero. The same can happen in the `Dai/sDai` oracle.
- Pyth, Uniswap V3, and Redstone oracles can directly return a zero price.

Consider implementing either of the following two changes to make the adapter behavior consistent:

- Prohibit the price from being zero (and negative) in all the adapters.
- Allow the price to be zero (and positive) in all the adapters and handle the special case of zero prices in the vaults.

Update: *Partially resolved in pull request #32. Prices being truncated to zero are not deemed incorrect. The Euler team stated:*

> *We have modified `PythOracle` and `RedstoneCoreOracle` to reject a signed price of 0. Truncation to 0 is possible in all adapters and we consider it correct behavior. 0 is the correct answer for the question that `getQuote` answers: "the amount of `quote` corresponding to `inAmount` of `base`". Note that truncation is possible in all adapters, not just the ones mentioned.*

L-02 Chainlink Adapter Can Return Incorrect Price During Flash Crashes

Some Chainlink price feeds might have a built-in minimum and maximum price that they can return. In the event the price falls below the minimum price or crosses the maximum price, the Chainlink oracle will return an incorrect price. This can lead to catastrophic consequences.

Consider allowing the deployer to define a percentage margin and if the price returned by Chainlink is within that narrow percentage of the minimum price or the maximum price, the adapter should revert. The minimum price and maximum price can be retrieved from the `OffchainAggregator` contract of the price feed at the `minAnswer` and `maxAnswer` variables.

Update: *Acknowledged, not resolved. The Euler team stated:*

> *Acknowledged. We chose not to use `minAnswer` and `maxAnswer` as indicators for the Chainlink oracle malfunctioning. It is unclear whether these values are expected to change, while reading them on every call would add a gas overhead to what would be a very hot path in production.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice proves beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for the maintainers of these libraries to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact.

Consider adding a NatSpec comment containing a security contact above the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #38. The Euler team stated:*

> *Fixed by adding `@custom:security-contact security@euler.xyz` to all contracts.*

N-02 Price Retrieval From Oracles Might Fail

The price retrieval action of some oracles is not generally controlled for potential failures. The Chainlink oracle's `latestRoundData` call might fail if Chainlink decides to restrict access to their oracles, Chronicle's oracle call can fail if the calling address is not whitelisted, and Pyth's `getPriceNoOlderThan` call might fail for several reasons as well. Similarly, other oracles can fail for a variety of reasons.

These are just examples and it seems that the codebase in general does not catch potential failures gracefully. If this is intentional, consider explicitly stating it in the docstrings. Otherwise, consider wrapping price retrieval calls into `try/catch` blocks and fail with specific error messages.

Update: *Acknowledged, not resolved. The Euler team stated:*

> *We have added in-code documentation in `ChronicleOracle` detailing that the adapter must be whitelisted prior to use. We have also added other pieces of in-code documentation to list some of the revert conditions of the connected oracles. An adapter reverting is the only correct behavior when an external call fails. In such a scenario, the adapter cannot safely provide a quote, and therefore cannot answer the `getQuote*` query in a satisfactory manner. Other options are returning 0 or a magic value, however we believe that this would constitute behavior that is unexpected, flaky, and highly detrimental to an uninformed consumer contract. We have decided not to `try/catch` external calls in the adapters due to several reasons:*
>
> *- It is only a semantic change. The adapter will still revert but the revert data will contain a standardized error message instead of the propagated vendor revert data.*
>
> *- It increases the verbosity and complexity of the adapters. This is because an external call may need to catch and translate multiple vendor errors.*
>
> *- Some revert conditions are impossible to catch or interpret. For example, panic errors or empty revert data.*
>
> *- Errors are generally not considered breaking changes, thus the behavior of adapters may change due silently, affecting connected consumers.*
>
> *- It promotes consumer code that branches based on revert data, which is considered an anti-pattern in Solidity. This is because of the listed reasons and also because errors can be easily spoofed by external code.*

> *We note that wherever possible, `euler-price-oracle` adapters revert with semantic custom errors (e.g., `PriceOracle_InvalidConfiguration`), and we agree that descriptive standardized errors are a best practice. Our disagreement is about mapping revert data of external calls to `euler-price-oracle`-specific errors.*

N-03 Oracle Price Updates Can Be Sandwiched

Oracles are meant to provide pricing of assets for Euler's vaults. These prices are used to perform typical operations of a synthetic/lending protocol. A push-based oracle's (like Chainlink and Chronicle) price update transaction can be seen in the mempool and users might decide to front run and/or back run such price updates to benefit from a rapid shift in price.

While we did explore how this can be an attack vector into the vault codebase, we found no practical examples. However, we deem this information important enough to raise it as an issue. A lot of research has been done about the topic and common solutions involve either the use of fees or delays between actions to remove incentives from value extraction on sandwich attacks within the same block or between a couple of blocks.

Consider reviewing the topic and adding mitigations wherever the team deems it necessary for such types of attacks.

Update: *Acknowledged, will resolve. The Euler team stated:*

> *Acknowledged. All network-based oracles have a degree of information asymmetry as pending price data can either be observed from the node network itself or consumed at the source directly. We are looking into the viability of this sandwich opportunity on a per-oracle basis. A good way to reduce the likelihood of a sandwich attack is to deploy a vault with a more conservative loan-to-value ratio. This is because the sandwiched price update will have to deviate more than `1-LTV%` to incur bad debt in the system. Adding delays to EVK vault actions would be unacceptable as it would negatively affect user experience and vault composability. We are currently experimenting with solutions in `euler-price-oracle` such as adding a spread to the mid-price in `getQuotes` that could also reduce the likelihood of a sandwich attack.*

N-04 `updatePrice` Is Updating the State Without Event Emission

The `updatePrice` function in the RedstoneCoreOracle contract is not emitting an event whenever a new price is set.

Consider adding an event emission every time a state variable changes its value.

Update: *Resolved in pull request #32. The Euler team stated:*

> *Fixed. The adapter now emits `CacheUpdated(uint256 price, uint256 priceTimestamp)` when the price is updated.*

N-05 Incorrect Docstrings

In the codebase, there are some cases in which docstrings are incorrect.

- In line 20 of the `IPriceOracle` interface, "get" should be "spend".
- In lines 78-79 of the `PythOracle` contract, "exponent confidence is too wide" is incorrect since the only existing confidence level is on the price and not in the exponent.
- In line 81 of the `RedstoneCoreOracle`, it is mentioned that `validateTimestamp` will be called in `getQuote` but this is not true since `getQuote` does not update the price at all.

Consider reviewing the codebase for incorrect docstrings in order to improve its overall readability and correctness.

Update: *Resolved in pull request #32. The Euler team stated:*

> *We added all missing pieces of NatSpec to the contracts. We also added in-code comments around the logic wherever applicable. We fixed several typos, wording issues, and invalid and stale pieces of documentation.*

N-06 Missing Assumption

The `UniswapV3Oracle` assumes that enough in-range liquidity is present in the pool for the price to not suffer major slippage effects. In addition, it is assumed that the observations used to calculate the price are not outdated and stale but this is missing in the docstrings.

Consider adding the latter assumption to the list of assumptions in the `UniswapV3Oracle` contract.

Update: *Acknowledged, not resolved.*

N-07 Missing Docstrings

Throughout the codebase, there are several parts that do not have docstrings. For instance, the `AggregatorV3Interface`, `IChronicle`, `IPot`, and `IStEth` interfaces are all lacking any sort of docstrings.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #32. The Euler team stated:*

> *We added all missing pieces of NatSpec to the contracts. We also added in-code comments around the logic wherever applicable. We fixed several typos, wording issues, and invalid and stale pieces of documentation.*

N-08 Lido Adapter Can Be Front Run

The `Lido adapter` uses the `getSharesByPooledEth` and the `getPooledEthByShares` functions of the `stETH` contract to price `stETH` into `wstETH` and vice versa. `wstETH` is a wrapped version of the `stETH` token and it takes into account the rewards gained (or losses suffered) by the Lido validators on the beacon chain. However, there is a delay between the accrual of validator rewards and this being reflected on-chain in the `stETH` contract. This creates an information asymmetry between the users and the vault. A malicious user can closely monitor this to calculate the next price of the `wstETH` token and use this to their advantage.

Consider reviewing the architecture of the Lido ecosystem thoroughly to find out ways to secure the system against any attack vectors arising out of such information asymmetry. Alternatively, consider using a different price feed like Pyth's `wstETH/USD` feed to price `wstETH`.

Update: *Acknowledged, not resolved. The Euler team stated:*

> *Updates to the `stETH` exchange rate are triggered though the Lido `AccountingOracle`. A trusted oracle committee (5/9 multisig) submits data about the economic state of Lido validators. Updates currently happen every 225 epochs (~24 hours) for accounting updates and every 75 epochs (~8 hours) for withdrawal requests. Before a report is applied, it is sanitized to adhere to several consistency rules, including*

> *capping the size of the rebase to `[-5%, +0.75%]`. Historical data since `2023-04-25` shows that the largest rebase was `0.0228%` at `2023-05-06` and the lowest was `0.0132%` at `2023-04-25`. Therefore, we can conclude that under normal functioning of the Lido system, the rebase rate is far too low to make sandwiching profitable.*
>
> *There are several multi-billion lending protocols that use the `stETH` exchange rate without known adverse events. We consider this strong empirical evidence of the absence of a viable attack plan arising from the rebase mechanism under normal operating conditions.*
>
> *As a side note, Aave implements additional validation on the combined exchange rate using their correlated-asset price oracle (CAPO). In Ethereum Mainnet Aave V3, the yearly growth of `wstETH/ETH` is capped to `9.68%`. Interestingly, CAPO does not impose a limit in the downward direction. Implementing an exchange rate cap mechanism with `IPriceOracle` is possible. However, we believe no immediate action is necessary due to the analysis above.*
>
> *EVK vault creators that do not wish to take the exchange rate from the Lido contracts can instead connect to one of many direct `wstETH` oracles such as the Chronicle `wstETH/USD` oracle, the Pyth `wstETH/USD`, or the Redstone Core `wstETH/ETH` oracle.*
>
> *However, we cannot issue a recommendation to use one over the other since there are fewer data sources for `wstETH` and these direct oracles may thus be considered more manipulable.*

Conclusion

The Euler Price Oracle codebase provides several oracle integrations in a way that one unique router can provide asset pricing through a common `IPriceOracle` interface. The system can be used by any protocol wishing to integrate such oracles, including Euler's ERC-4626 vaults.

Overall, we found the codebase to be mature with clear documentation and secure patterns adopted. Some specific oracles, as highlighted by the reported issues, behave inconsistently and might also be front run or back run in their price update mechanism. The implications of this definitely affect integrators that make use of these oracles. Special attention should be paid to the Redstone integration since it is the most sensitive when it comes to flash crashes and rapid price changes.

The Euler team was responsive and provided us with timely answers and support. Given the explicit intentions of keeping the system immutable with no owner, we suggest monitoring for oracle price retrieval failures and having failover mechanisms in place for integrators to eventually opt out if some of the oracles start to give unexpected results or failures.