OMNISCIA

May 10, 2024

SMART CONTRACT AUDIT REPORT

Euler Finance
EVK Price Oracles


omniscia.io


info@omniscia.io


Online report: euler-finance-evk-price-oracles

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.

omniscia.io


info@omniscia.io

EVK Price Oracles Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
c91093256f	April 1st 2024	1d801106a2
eeb1847df7	May 3rd 2024	46fcf8f104
eeb1847df7	May 10th 2024	a300a47bfd

Audit Overview

We were tasked with performing an audit of the Euler Finance codebase and in particular their EVK Price Oracles module.

High-Level Description

The EVK Price Oracles repository is meant to represent multiple oracle implementations that target specific protocols as well as a top-level router implementation that will detect which oracle it should route a quoting call to.

The security review focused on answering all the `Security Questions` that were present in the repository's `README.md` file as well as validating that the integrations of each protocol have been securely performed.

Euler Router

This module is responsible for properly routing any quoting calls to the correct oracle implementations, as well as wrapping and unwrapping the input and output assets if they represent **EIP-4626** vaults.

An important error was identified in the conversion mechanism's recursion when the output asset (`quote`) is a wrapped variant, causing it to produce invalid results when **either of the two following conditions is met**:

The first condition is effectively a subset of the second condition due to Solidity arithmetic operations being non-commutative, however, it is outlined for simplicity.

Chainlink Oracle

The Chainlink oracle represents a straightforward implementation that queries the latest price by a Chainlink oracle, applies a basic staleness check on the latest report, and yields the post-conversion amount by performing the calculation in the correct direction.

From a documentational perspective, we highlighted that stablecoin asset classes are not meant to be considered equivalent and this finding expands to the `README.md` file and specifically the following statement:

> Note that we consider pricing "by analogy" a valid use case, i.e. using a ETH/USD feed for pricing WETH/GUSD.

The above statement is **incorrect** as stablecoins are never on-par with their stable asset (i.e. `1 USDC != 1 USDT != 1 USD`). We advise such terminology to be removed from the codebase, avoiding insecure deployment of oracles which would result in exploitable arbitrage opportunities.

From a security perspective, we denoted two best practices that are also recommended by the Chainlink specification and are not presently adhered to by the oracle.

Chronicle Oracle

The Chronicle oracle represents an implementation meant to interface with the Chronicle Protocol, however, we observed a potential issue with the integrated project.

The Chronicle Protocol oracles appear to enforce access control on all price-querying functions via a `Toll` system, yet the `ChronicleOracle` does not appear to register itself as an authorized invoker.

The integration code available in the Chronicle Protocol documentation highlights that a `SelfKisser` implementation needs to be integrated with, however, the production deployment of the Chronicle Protocol most likely does not have an instance of it deployed.

Based on the aforementioned data, we advise the Chronicle Oracle implementation to be set aside as a future implementation once the Chronicle Protocol matures.

Alternatively, we advise the self-registration (or manual registration) mechanism that the Euler Finance team may have directly agreed with the Chronicle Protocol to be detailed.

Lido Oracle

The Lido Oracle is meant to utilize the `stETH` asset directly to convert bidirectionally in the `stETH-wstETH` quotation path.

The contract is securely integrated, and direct integration with `stETH` instead of `wstETH` is performed as a matter of optimization as the relevant `wstETH` functions would simply forward calls to the `stETH` implementation (i.e. `wstETH::getWstETHByStETH` simply forwards the call to `stETH::getSharesByPooledEth`).

A systemic risk similar to the Rocket Pool can be observed whereby users can be privy to upcoming rebase operations and thus arbitrage the instantaneous changes in conversion rates.

Rebase operations occur only once per 24-hour intervals, and in case the rebase oracles of Lido Finance do not achieve a consensus, the rebase may be skipped entirely and thus compound to the next epoch.

So as to avoid this systemic risk, we advise the code to integrate with the Lido `AccountingOracle` to

impose a "staleness" check based on when consensus was last reached.

MakerDAO Oracle (sDAI)

The MakerDAO Oracle is meant to integrate with the savings `DAI` Pot to convert between the `sDAI` and `DAI` tokens.

A flaw was identified in the conversion mechanism whereby the conversion rate utilized could be considered stale as the latest cumulative interest rate is not calculated.

Pyth Oracle

The Pyth Oracle provides a way to integrate with any oracle supported by the Pyth Network, utilizing a custom execution path for `ScaleUtils` that supports positive exponents.

As outlined in findings within the report, positive exponents are not considered to be valid values by the Pyth Network itself, and thus the code can be simplified similarly to other oracles.

Redstone Oracle

The Redstone Oracle exposes a contract that integrates with Redstone's "on-demand" price measurements by exposing a separate function via which price measurements are maintained in the contract.

A flaw in the staleness check was identified whereby the actual staleness of the data point consumed would be higher than the actual timestamp the price was reported at by the Redstone oracle.

Rocket Pool Oracle

The Rocket Pool Oracle is meant to utilize the `rETH` asset directly to convert bidirectionally in the `ETH-rETH` quotation path.

Similarly to the Lido Oracle, the Rocket Pool Oracle suffers from the same systemic risk of delayed rebase reports that occur once per 24 hours and may be delayed for a longer duration.

To impose similar staleness checks to the Lido Oracle, we advise the `RocketNetworkBalances` contract deployment to be integrated with that reports the last block balances were recorded at.

The delta between the current `block.number` and the value reported by the `IRocketNetworkBalances::getBalancesBlock` function can be utilized to calculate the time that has elapsed since the last update, permitting a staleness check to be imposed.

Uniswap V3 Oracle

The Uniswap V3 oracle implementation implements a TWAP based mechanism to calculate a time-

weighted average price for a particular Uniswap V3 AMM pair.

In this implementation, we observed that the cardinality of the AMM pair interacted with is not increased, and that certain mathematical operations were incorrectly performed using checked arithmetics.

Overview Conclusion

Systemic risks are inherent to any off-chain data source that is utilized as an on-chain price measurement mechanism.

Within the audit report, we describe that the Rocket Pool and Lido oracle implementations are especially susceptible to arbitrage due to their delayed reporting mechanism.

The actual level of arbitrage that would be exposed by these mechanisms is minimal, and at the time of the production of this report the following percentage adjustments were observed:

The influence of these deviations may be considered negligible for the use-case of the Euler Finance team, and as such the complex staleness checks may be considered unnecessary.

As an additional point, the `ETH-rETH` oracle in particular can be utilized in conjunction with an AMM pair (i.e. Curve) with a deviation threshold between the two rates permitted which would provide a more resistant evaluation of the assets' exchange rate.

We advise the Euler Finance team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Euler Finance team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Euler Finance and have identified that all exhibits have either been adequately dealt with, have been safely acknowledged, or have been properly contested.

We consider all outputs of the audit report properly consumed by the Euler Finance team rendering this audit engagement to be concluded.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	31	24	0	7
Minor	7	7	0	0
Medium	2	2	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **4 findings utilizing static analysis** tools as well as identified a total of **37 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/euler-xyz/euler-price-oracle**
- Commit: c91093256f62ca7d35cc35885d1b8a77d567983d
- Language: Solidity
- Network: Ethereum
- Revisions: **c91093256f**, **eeb1847df7**

Contracts Assessed

File	Total Finding(s)
src/adapter/chainlink/AggregatorV3Interface.sol (AVI)	0
src/adapter/BaseAdapter.sol (BAR)	0
src/adapter/chainlink/ChainlinkOracle.sol (COE)	4
src/adapter/chronicle/ChronicleOracle.sol (COL)	2
src/lib/Errors.sol (ESR)	0
src/EulerRouter.sol (ERR)	3
src/lib/Governable.sol (GEL)	2
src/adapter/lido/LidoOracle.sol (LOE)	3
src/adapter/pyth/PythOracle.sol (POE)	7
src/adapter/rocketpool/RethOracle.sol (ROE)	3

src/adapter/redstone/RedstoneCoreOracle.sol (RCO)	6
src/adapter/maker/SDaiOracle.sol (SDO)	3
src/lib/ScaleUtils.sol (SUS)	4
src/adapter/uniswap/UniswapV3Oracle.sol (UVO)	4

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.23` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely located in external dependencies and can thus be safely ignored.

All `pragma` statements of the system have been locked to `0.8.23`, the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **15 potential issues** within the codebase of which **8 were ruled out to be false positives** or negligible findings.

The remaining **7 issues** were validated and grouped and formalized into the **4 exhibits** that follow:

ID	Severity	Addressed	Title
GEL-01S	Informational	Nullified	Inexistent Sanitization of Input Address
LOE-01S	Informational	Acknowledged	Inexistent Sanitization of Input Addresses
POE-01S	Informational	Nullified	Illegible Numeric Value Representations
ROE-01S	Informational	Nullified	Inexistent Sanitization of Input Addresses

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Euler Finance's EVK-compatible oracle system.

As the project at hand implements multiple oracle integrations interfaced via a router, intricate care was put into ensuring that the **call flows within the system conform to the specifications and restrictions** laid forth within the protocol's specification and that **all external interactions are securely performed**.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. Additionally, we validated the following security questions that were part of the repository:

1. Adapters check whether the raw price data is valid or indicates an error condition (e.g. negative price or invalid signature). Are all cases caught?
2. Is the validation of the `PythStructs.Price` in the Pyth adapter correct? Should the `expo` boundary be increased or decreased?
3. Are there real-world cases where we would need a scaling factor larger than 10^38 in `ScaleUtils`? Interested in examples for Pyth, Redstone and Chainlink feeds. Note that we consider pricing "by analogy" a valid use case, i.e. using a ETH/USD feed for pricing WETH/GUSD.
4. Are there quirky feeds in Pyth, Redstone and Chainlink which break the assumptions of the adapters? For example, the Redstone adapter has `FEED_DECIMALS=8` hardcoded as a constant, whereas the Chainlink adapter relies that the aggregator decimals correctly correspond to the actual decimals.
5. Are there timing games / OEV opportunities that arise from the price caching logic in Redstone and Pyth adapters?
6. Are the on-chain exchange rate oracles (sDAI, rETH, stEth) immune to manipulation? Are there additional conditions that we can check which could signal that these rates cannot be trusted?
7. Could any of the hardcoded addresses change under normal operation conditions e.g. as part of an upgrade?

As a result of their investigation, we **pinpointed multiple minor-to-moderate vulnerabilities** within the system which could have had **severe ramifications** to its overall operation under specific circumstances.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to the extent it need be, however, the Chronicle Protocol oracle implementation requires additional documentation as the protocol integrated with is not yet mature.

A total of **37 findings** were identified over the course of the manual review of which **26 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
COE-01M	Unknown	Nullified	Inexistent Volatility Protection Mechanisms
COE-02M	Informational	Acknowledged	Inexistent Validation of Acceptable Answer Range
COE-03M	Informational	Yes	Misleading Specification of Usability
COE-04M	Informational	Yes	Potentially Unsupported Function Signature
COL-01M	Informational	Yes	Inexistent Registration of Chronicle Subscriber

COL-02M	Potentially Unsupported Function Signature
ERR-01M	Improper Oracle Resolution Mechanism
ERR-02M	Incorrect Oracle Resolution of EIP-4626 Vaults
LOE-01M	Potentially Stale Calculation of Exchange Rate (Asynchronous Rewards / Penalties)
POE-01M	Inexistent Configurability of Confidence Width
POE-02M	Inexistent Prevention of Overpayment
POE-03M	Potentially Unsupported Function Signature
POE-04M	Improper Validation of Exponent
RCO-01M	Improper Integration of Redstone On-Demand Feeds

RCO-02M	Inexistent Capability of Functionality Overrides
RCO-03M	Potentially Unsupported Function Signature
RCO-04M	Improper Assumption of Oracle Decimals
RCO-05M	Misconceived Data Staleness
ROE-01M	Potentially Stale Calculation of Exchange Rate (Asynchronous Rewards / Penalties)
SDO-01M	Insecure Usage of Outdated Interest Rate Accumulator
SUS-01M	Potential Increase of Acceptable Values
SUS-02M	Potential Negation Overflow
UVO-01M	Inexistent Validation of Observation Cardinality Length

UVO-02M	Informational	Yes	Insecure Typecasting Operation (TWAP)
UVO-03M	Minor	Yes	Insecure Calculation of Mean Tick
UVO-04M	Minor	Yes	Potentially Insecure TWAP Window

Code Style

During the manual portion of the audit, we identified **11 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
ERR-01C	Informational	Yes	Imprecise Terminology
GEL-01C	Informational	Yes	Redundant Variable Caching
LOE-01C	Informational	Nullified	Potential Usage of Library
POE-01C	Informational	Yes	Potentially Redundant Function Implementation
POE-02C	Informational	Yes	Redundant Handling of Positive Exponent
RCO-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
ROE-01C	Informational	Nullified	Potential Usage of Library
SDO-01C	Informational	Nullified	Potential Usage of Library
SDO-02C	Informational	Yes	Repetitive Value Literal
SUS-01C	Informational	Yes	Inefficient Erasure of Upper Bits

SUS-02C	Informational	Nullified	Repetitive Value Literal

Governable Static Analysis Findings

GEL-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	Governable.sol:L18-L20

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/lib/Governable.sol

SOL

```sol
18  constructor(address _governor) {
19      _setGovernor(_governor);
20  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

The Euler Finance team evaluated this exhibit and opted to dispute it as configuring a `governor` as the `address(0)` is a desirable feature of the suite.

Given that the exhibit outlines behaviour that aligns with the business requirements of the Euler Finance team, we consider this exhibit to be nullified.

LidoOracle Static Analysis Findings

LOE-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	LidoOracle.sol:L20-L23

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/adapter/lido/LidoOracle.sol

SOL

```sol
20  constructor(address _stEth, address _wstEth) {
21      stEth = _stEth;
22      wstEth = _wstEth;
23  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

The Euler Finance team evaluated this exhibit and opted to acknowledge it as they intend to deploy oracles via factory contracts meant to mitigate human error.

PythOracle Static Analysis Findings

POE-01S: Illegible Numeric Value Representations

Type	Severity	Location
Code Style	Informational	PythOracle.sol:L15, L74

Description:

The linked representations of numeric literals are sub-optimally represented decreasing the legibility of the codebase.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
15  uint256 internal constant MAX_CONF_WIDTH = 500;
```

Recommendation:

To properly illustrate each value's purpose, we advise the following guidelines to be followed. For values meant to depict fractions with a base of `1e18`, we advise fractions to be utilized directly (i.e. `1e17` becomes `0.1e18`) as they are supported. For values meant to represent a percentage base, we advise each value to utilize the underscore (`_`) separator to discern the percentage decimal (i.e. `10000` becomes `100_00`, `300` becomes `3_00` and so on). Finally, for large numeric values we simply advise the underscore character to be utilized again to represent them (i.e. `1000000` becomes `1_000_000`).

Alleviation:

The relevant variable has been omitted per remediation conducted for exhibit **`POE-01M`** rendering this recommendation no longer applicable.

RethOracle Static Analysis Findings

ROE-01S: Inexistent Sanitization of Input Addresses

Type	Severity	Location
Input Sanitization	Informational	RethOracle.sol:L20-L23

Description:

The linked function(s) accept `address` arguments yet do not properly sanitize them.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

src/adapter/rocketpool/RethOracle.sol

SOL

```sol
20  constructor(address _weth, address _reth) {
21      weth = _weth;
22      reth = _reth;
23  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that each `address` specified is non-zero.

Alleviation:

The Euler Finance team has proceeded with removing the `RethOracle` from the codebase in light of exhibit **`ROE-01M`**, rendering this exhibit no longer applicable.

ChainlinkOracle Manual Review Findings

COE-01M: Inexistent Volatility Protection Mechanisms

Type	Severity	Location
External Call Validation	Unknown	ChainlinkOracle.sol:L54

Description:

The `ChainlinkOracle::_getQuote` function will not enforce any volatility checks and will simply yield a price that protocols can utilize.

This is insecure from a DeFi perspective as prices generated from volatile assets are not expected to be consumed, and this is listed as a security consideration by the Chainlink documentation.

Impact:

The severity of this exhibit is considered unknown as it is not clear whether the oracles of the Euler Finance team should guard against volatility.

Example:

src/adapter/chainlink/ChainlinkOracle.sol

SOL

```sol
46  /// @notice Get the quote from the Chainlink feed.
47  /// @param inAmount The amount of `base` to convert.
48  /// @param _base The token that is being priced.
49  /// @param _quote The token that is the unit of account.
50  /// @return The converted amount using the Chainlink feed.
51  function _getQuote(uint256 inAmount, address _base, address _quote) internal view
override returns (uint256) {
52      bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);
53
54      (, int256 answer,, uint256 updatedAt,) =
AggregatorV3Interface(feed).latestRoundData();
55      if (answer <= 0) revert Errors.PriceOracle_InvalidAnswer();
```

Example (Cont.):

```sol
56      uint256 staleness = block.timestamp - updatedAt;
57      if (staleness > maxStaleness) revert Errors.PriceOracle_TooStale(staleness,
maxStaleness);
58
59      uint256 price = uint256(answer);
60      return ScaleUtils.calcOutAmount(inAmount, price, scale, inverse);
61  }
```

Recommendation:

We advise the Euler Finance team to evaluate this trait, and potentially incorporate volatility protections into the `ChainlinkOracle` itself.

Alleviation:

The Euler Finance team evaluated this exhibit, and clarified that they do not consider volatility protections to be a feature exposed by the Euler Finance oracle system.

As such, we consider this exhibit as inapplicable given that lack of validation is a desirable business trait of the system.

COE-02M: Inexistent Validation of Acceptable Answer Range

Type	Severity	Location
External Call Validation	Informational	ChainlinkOracle.sol:L54

Description:

The `AggregatorV3Interface::aggregator` of a particular proxy oracle in Chainlink has an acceptable set of values that it can report that are exposed by its `IOffchainAggregator::minAnswer` and `IOffchainAggregator::maxAnswer` variables. A price reported close to these values indicates a sharp market event and as such should in most circumstances not be consumed.

Additionally, the aggregators may continue to report the minimum / maximum price they are able to thereby bypassing the `staleness` check whilst reporting a price that may be incorrect.

Example:

src/adapter/chainlink/ChainlinkOracle.sol

SOL

```sol
46  /// @notice Get the quote from the Chainlink feed.
47  /// @param inAmount The amount of `base` to convert.
48  /// @param _base The token that is being priced.
49  /// @param _quote The token that is the unit of account.
50  /// @return The converted amount using the Chainlink feed.
51  function _getQuote(uint256 inAmount, address _base, address _quote) internal view
override returns (uint256) {
52      bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);
53
54      (, int256 answer,, uint256 updatedAt,) =
AggregatorV3Interface(feed).latestRoundData();
55      if (answer <= 0) revert Errors.PriceOracle_InvalidAnswer();
```

Example (Cont.):

```sol
56      uint256 staleness = block.timestamp - updatedAt;
57      if (staleness > maxStaleness) revert Errors.PriceOracle_TooStale(staleness,
maxStaleness);
58
59      uint256 price = uint256(answer);
60      return ScaleUtils.calcOutAmount(inAmount, price, scale, inverse);
61  }
```

Recommendation:

As this particular warning is listed in the security best practices of Chainlink, we advise it to be potentially incorporated into the operation of the `ChainlinkOracle`. As a potential solution, the minimum and maximum acceptable answers by the aggregator of a data feed proxy can be extracted and the `answer` detected could be evaluated as being within the range of the aggregator and specifically outside of a narrow margin close to the limits.

Alleviation:

The Euler Finance team evaluated this exhibit, and clarified that they do not wish to incorporate the Chainlink price range limitations within the oracle.

As such, we consider this exhibit as safely acknowledged.

COE-03M: Misleading Specification of Usability

Type	Severity	Location
Standard Conformity	Informational	ChainlinkOracle.sol:L32

Description:

The `ChainlinkOracle::constructor` documentation specifies that the base and quote tokens are not required to correspond to the feed assets, however, this is incorrect unless under a very specific set of circumstances.

The example specified is a condition **under which security would be compromised** as the `USD` is not always at parity with the `USDC` stablecoin.

Example:

src/adapter/chainlink/ChainlinkOracle.sol

SOL

```sol
26  /// @notice Deploy a ChainlinkOracle.
27  /// @param _base The address of the base asset corresponding to the feed.
28  /// @param _quote The address of the quote asset corresponding to the feed.
29  /// @param _feed The address of the Chainlink price feed.
30  /// @param _maxStaleness The maximum allowed age of the price.
31  /// @dev Base and quote are not required to correspond to the feed assets.
32  /// For example, the ETH/USD feed can be used to price WETH/USDC.
33  constructor(address _base, address _quote, address _feed, uint256 _maxStaleness)
{
34      base = _base;
35      quote = _quote;
```

Example (Cont.):

```sol
        feed = _feed;
        maxStaleness = _maxStaleness;

        // The scale factor is used to correctly convert decimals.
        uint8 baseDecimals = IERC20(base).decimals();
        uint8 quoteDecimals = IERC20(quote).decimals();
        uint8 feedDecimals = AggregatorV3Interface(feed).decimals();
        scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, feedDecimals);
    }
```

Recommendation:

We strongly advise the comment line to be removed, and the overall recommendation to be avoided.

While developers are free to disassociate the `base`, `quote`, and `feed` implementations it is ill-advised to do so in most circumstances as even "identical" assets naturally deviate from each other in price.

Alleviation:

The potentially misinterpreted statement has been removed from the `ChainlinkOracle::constructor` documentation as advised.

COE-04M: Potentially Unsupported Function Signature

Type	Severity	Location
Standard Conformity	Informational	ChainlinkOracle.sol:L40, L41

Description:

The code of the `ChainlinkOracle::constructor` will invoke the `IERC20::decimals` function as exposed by the `forge-std` library, however, the `IERC20::decimals` function **is not actually part of the EIP-20 specification**.

Impact:

Most **EIP-20** assets do implement the `IERC20::decimals` function signature, however, it is not mandated by the standard and as such a small subset of **EIP-20** tokens is incompatible with the `ChainlinkOracle` presently.

Example:

src/adapter/chainlink/ChainlinkOracle.sol

SOL

```sol
26  /// @notice Deploy a ChainlinkOracle.
27  /// @param _base The address of the base asset corresponding to the feed.
28  /// @param _quote The address of the quote asset corresponding to the feed.
29  /// @param _feed The address of the Chainlink price feed.
30  /// @param _maxStaleness The maximum allowed age of the price.
31  /// @dev Base and quote are not required to correspond to the feed assets.
32  /// For example, the ETH/USD feed can be used to price WETH/USDC.
33  constructor(address _base, address _quote, address _feed, uint256 _maxStaleness)
{
34      base = _base;
35      quote = _quote;
```

Example (Cont.):

```sol
        feed = _feed;
        maxStaleness = _maxStaleness;

        // The scale factor is used to correctly convert decimals.
        uint8 baseDecimals = IERC20(base).decimals();
        uint8 quoteDecimals = IERC20(quote).decimals();
        uint8 feedDecimals = AggregatorV3Interface(feed).decimals();
        scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, feedDecimals);
    }
```

Recommendation:

In case all **EIP-20** assets are expected to be supported, we advise decimals to either be opportunistically queried or for decimals to be supplied as input arguments thus permitting any token to have a `ChainlinkOracle` deployed.

Alleviation:

A common `BaseAdapter::_getDecimals` implementation has been introduced in the `BaseAdapter` upstream contract that will attempt to fetch the `IERC20::decimals` of an asset and default to `32` if they cannot be fetched.

As such, we consider this exhibit fully alleviated.

ChronicleOracle Manual Review Findings

COL-01M: Inexistent Registration of Chronicle Subscriber

Type	Severity	Location
Standard Conformity	Informational	ChronicleOracle.sol:L43

Description:

The Chronicle oracle system implements an access control mechanism for its oracles permitting only those who have properly registered themselves as consumers to interact with them.

This can be confirmed on the presently-live oracles on the Ethereum mainnet, such as the `Chronicle_GNO_USD_2` oracle **located here**.

Example:

src/adapter/chronicle/ChronicleOracle.sol

SOL

```sol
/// @notice Deploy a ChronicleOracle.
/// @param _base The address of the base asset corresponding to the feed.
/// @param _quote The address of the quote asset corresponding to the feed.
/// @param _feed The address of the Chronicle price feed.
/// @param _maxStaleness The maximum allowed age of the price.
/// @dev Base and quote are not required to correspond to the feed assets.
/// For example, the ETH/USD feed can be used to price WETH/USDC.
constructor(address _base, address _quote, address _feed, uint256 _maxStaleness)
{
    base = _base;
    quote = _quote;
```

Example (Cont.):

```sol
        feed = _feed;
        maxStaleness = _maxStaleness;

        // The scale factor is used to correctly convert decimals.
        uint8 baseDecimals = IERC20(base).decimals();
        uint8 quoteDecimals = IERC20(quote).decimals();
        uint8 feedDecimals = IChronicle(feed).decimals();
        scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, feedDecimals);
    }

    /// @notice Get the quote from the Chronicle feed.
    /// @param inAmount The amount of `base` to convert.
    /// @param _base The token that is being priced.
    /// @param _quote The token that is the unit of account.
    /// @return The converted amount using the Chronicle feed.
    function _getQuote(uint256 inAmount, address _base, address _quote) internal view
override returns (uint256) {
        bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);

        (uint256 price, uint256 age) = IChronicle(feed).readWithAge();
        if (price == 0) revert Errors.PriceOracle_InvalidAnswer();

        uint256 staleness = block.timestamp - age;
        if (staleness > maxStaleness) revert Errors.PriceOracle_TooStale(staleness,
maxStaleness);

        return ScaleUtils.calcOutAmount(inAmount, price, scale, inverse);
    }
```

Recommendation:

The Chronicle oracle system presently appears immature, and access control is imposed but no on-chain mechanism appears to be present in the Ethereum mainnet to acquire access.

Development examples make use of a `SelfKisser` instance that permits a Chronicle oracle user to authenticate themselves via the `ISelfKisser::selfKiss` function, however, this approach is incorrect for the Ethereum mainnet as no such contract is present there (at least publicly) and the **`SelfKisser` repository specifies it will not be deployed in production**.

Due to these reasons, we advise the `ChronicleOracle` implementation to be considered incomplete until the Chronicle project matures. If authentication is expected to be manually provided by the Chronicle team directly, we advise this to be denoted in the codebase as the `ChronicleOracle` is not a permissionless contract that can be deployed by anyone in such a case.

Alleviation:

The Euler Finance team reached out to the Chronicle team for clarifications and confirmed that explicit authorization must be bestowed to the `ChronicleOracle` before it can fetch price measurements from a Chronicle oracle.

Based on this information and our recommendation, the documentation of the `ChronicleOracle` contract was updated to outline this restriction which we consider as sufficient warning for potential deployers of the `ChronicleOracle` contract.

COL-02M: Potentially Unsupported Function Signature

Type	Severity	Location
Standard Conformity	Informational	ChronicleOracle.sol:L41, L42

Description:

The code of the `ChronicleOracle::constructor` will invoke the `IERC20::decimals` function as exposed by the `forge-std` library, however, the `IERC20::decimals` function **is not actually part of the EIP-20 specification**.

Impact:

Most **EIP-20** assets do implement the `IERC20::decimals` function signature, however, it is not mandated by the standard and as such a small subset of **EIP-20** tokens is incompatible with the `ChronicleOracle` presently.

Example:

src/adapter/chronicle/ChronicleOracle.sol

SOL

```sol
27  /// @notice Deploy a ChronicleOracle.
28  /// @param _base The address of the base asset corresponding to the feed.
29  /// @param _quote The address of the quote asset corresponding to the feed.
30  /// @param _feed The address of the Chronicle price feed.
31  /// @param _maxStaleness The maximum allowed age of the price.
32  /// @dev Base and quote are not required to correspond to the feed assets.
33  /// For example, the ETH/USD feed can be used to price WETH/USDC.
34  constructor(address _base, address _quote, address _feed, uint256 _maxStaleness)
{
35      base = _base;
36      quote = _quote;
```

Example (Cont.):

```sol
        feed = _feed;
        maxStaleness = _maxStaleness;

        // The scale factor is used to correctly convert decimals.
        uint8 baseDecimals = IERC20(base).decimals();
        uint8 quoteDecimals = IERC20(quote).decimals();
        uint8 feedDecimals = IChronicle(feed).decimals();
        scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, feedDecimals);
    }
```

Recommendation:

In case all **EIP-20** assets are expected to be supported, we advise decimals to either be opportunistically queried or for decimals to be supplied as input arguments thus permitting any token to have a `ChronicleOracle` deployed.

Alleviation:

A common `BaseAdapter::_getDecimals` implementation has been introduced in the `BaseAdapter` upstream contract that will attempt to fetch the `IERC20::decimals` of an asset and default to `32` if they cannot be fetched.

As such, we consider this exhibit fully alleviated.

EulerRouter Manual Review Findings

ERR-01M: Improper Oracle Resolution Mechanism

Type	Severity	Location
Logical Fault	Minor	EulerRouter.sol:L49, L114

Description:

The `oracles` entry during a `EulerRouter::govSetConfig` function call is assigned-to only once. Per the design principles of the oracles and specifically the bidirectional trait, an oracle for the `base-quote` path would be valid for the `quote-base` path as well.

Impact:

The `EulerRouter` will report that it does not have an oracle defined for a `base` and `quote` pair even though it might actually have registered one.

Example:

src/EulerRouter.sol

SOL

```sol
43  /// @notice Configure a PriceOracle to resolve base/quote.
44  /// @param base The address of the base token.
45  /// @param quote The address of the quote token.
46  /// @param oracle The address of the PriceOracle that resolves base/quote.
47  /// @dev Callable only by the governor.
48  function govSetConfig(address base, address quote, address oracle) external
onlyGovernor {
49      oracles[base][quote] = oracle;
50      emit ConfigSet(base, quote, oracle);
51  }
```

Recommendation:

We advise the code to either assign both `oracles[base][quote]` and `oracles[quote][base]`, or to order the input `base` and `quote` values at the beginning of the **`EulerRouter::_resolveOracle`** function.

Alleviation:

The latter of our two recommendations has been incorporated into the codebase, ordering the assets prior to an assignment in the `oracles` data entry and exposing a `EulerRouter::getConfiguredOracle` function that will perform this sorting internally to fetch the correct oracle.

ERR-02M: Incorrect Oracle Resolution of EIP-4626 Vaults

Type	Severity	Location
Logical Fault	Medium	EulerRouter.sol:L125

Description:

The `EulerRouter::_resolveOracle` function will incorrectly resolve the oracle path of a quote from any `base` asset to an **EIP-4626** `quote` asset as it will mutate the input amount as if it were shares of the `quote` vault.

This vulnerability will manifest itself under the following condition:

All division operations in Solidity are non-commutative due to truncation, causing all **EIP-4626** to produce incorrect results whenever their exchange rate results in a truncation.

Additionally, any non-commutative **EIP-4626** implementation **would cause significant discrepancies** in the converted amounts, and an **EIP-4626** may be non-commutative by design as it is not prevented by the standard.

Impact:

The current oracle resolution mechanism will misbehave with a varying degree of severity, depending on the "level" at which the **EIP-4626** is non-commutative (i.e. simple truncation vs. more important deviations such as exponential formulae, like a bonding curve).

Example:

src/EulerRouter.sol

```sol
/// @notice Resolve the PriceOracle to call for a given base/quote pair.
/// @param inAmount The amount of `base` to convert.
/// @param base The token that is being priced.
/// @param quote The token that is the unit of account.
/// @dev Implements the following recursive resolution logic:
/// 1. Check the base case: `base == quote` and terminate if true.
/// 2. If a PriceOracle is configured for base/quote in the `oracles` mapping,
///    return it without transforming the other variables.
/// 3. If `base` is configured as an ERC4626 vault with internal pricing,
///    transform inAmount by calling `convertToAssets` and recurse by
substituting `asset` for `base`.
```

Example (Cont.):

```sol
/// 4. If `quote` is configured as an ERC4626 vault with internal pricing,
///    transform inAmount by calling `convertToAssets` and recurse by substituting `asset` for `quote`.
/// 5. If there is a fallback oracle, return it without transforming the other variables, else revert.
/// @return The resolved inAmount.
/// @return The resolved base.
/// @return The resolved quote.
/// @return The resolved PriceOracle to call.
function _resolveOracle(uint256 inAmount, address base, address quote)
    internal
    view
    returns (uint256, /* inAmount */ address, /* base */ address, /* quote */ address /* oracle */ )
{
    // Check the base case
    if (base == quote) return (inAmount, base, quote, address(0));
    // 1. Check if base/quote is configured.
    address oracle = oracles[base][quote];
    if (oracle != address(0)) return (inAmount, base, quote, oracle);
    // 2. Recursively resolve `base`.
    address baseAsset = resolvedVaults[base];
    if (baseAsset != address(0)) {
        inAmount = IERC4626(base).convertToAssets(inAmount);
        return _resolveOracle(inAmount, baseAsset, quote);
    }
    // 3. Recursively resolve `quote`.
    address quoteAsset = resolvedVaults[quote];
    if (quoteAsset != address(0)) {
        inAmount = IERC4626(quote).convertToShares(inAmount);
        return _resolveOracle(inAmount, base, quoteAsset);
```

Example (Cont.):

```sol
    }
    // 4. Return the fallback or revert if not configured.
    oracle = fallbackOracle;
    if (oracle == address(0)) revert Errors.PriceOracle_NotSupported(base,
quote);
    return (inAmount, base, quote, oracle);
}
```

Recommendation:

Mutation of the `inAmount` value in the current mechanism is misleading, as the code solely behaves properly if the `quoteAsset` and the `base` are identical in which case the mutation is essentially applied to the output amount.

As such, we advise the code's recursion to be restructured so as to properly perform the inner-most vault conversion operation first, bubbling the result upwards and converting it on each step to produce the final result.

Alleviation:

The Euler Finance team thoroughly evaluated the ramifications of this exhibit as well as the `EulerRouter::_resolveOracle` call flows throughout the use cases of the `EulerRouter`, and opted to omit recursion for the `quote` asset altogether.

As such, the described misbehaviour is no longer applicable rendering the exhibit alleviated by omission.

LidoOracle Manual Review Findings

LOE-01M: Potentially Stale Calculation of Exchange Rate (Asynchronous Rewards / Penalties)

Type	Severity	Location
External Call Validation	Informational	LidoOracle.sol:L33, L35

Description:

The `stETH` contract implementation exposes utility functions for calculating the **on-chain conversion of two assets** rather than the exchange rate that is influenced by the "true" value of the asset. In practical terms, the `IStEth::getSharesByPooledEth` and `IStEth::getPooledEthByShares` functions do not reflect future profits **already captured** that the Lido contracts will reflect via processing of the latest Lido oracle consensus.

Per the Lido documentation, the balances of the Lido system are updated at a 24-hour interval providing adequate time for a user to be knowledgeable of the shift in the exchange rate and to capitalize on it.

Impact:

The **`LidoOracle::_getQuote`** function will consistently undervalue the `wstETH` asset in relation to the `stETH` asset due to not accounting for earned but not yet reflected rewards of the Beacon Chain node operators that are affiliated with Lido. Additionally, the same function may overvalue the `wstETH` asset in relation to the `stETH` asset if losses incurred by Lido have not yet been reflected on-chain.

Example:

src/adapter/lido/LidoOracle.sol

SOL

```sol
/// @notice Get a quote by querying the exchange rate from the stEth contract.
/// @dev Calls `getSharesByPooledEth` for stEth/wstEth and `getPooledEthByShares` for wstEth/stEth.
/// @param inAmount The amount of `base` to convert.
/// @param base The token that is being priced. Either `stEth` or `wstEth`.
/// @param quote The token that is the unit of account. Either `wstEth` or `stEth`.
/// @return The converted amount.
function _getQuote(uint256 inAmount, address base, address quote) internal view override returns (uint256) {
    if (base == stEth && quote == wstEth) {
        return IStEth(stEth).getSharesByPooledEth(inAmount);
    } else if (base == wstEth && quote == stEth) {
```

Example (Cont.):

```sol
            return IStEth(stEth).getPooledEthByShares(inAmount);
        }
        revert Errors.PriceOracle_NotSupported(base, quote);
    }
```

Recommendation:

The Lido ecosystem contains an `AccountingOracle` deployment that can be integrated to calculate the last time a consensus was processed as well as whether a consensus is presently pending.

These attributes can be utilized to prevent an exchange rate from being considered as valid if the time since the last processed consensus exceeds `24 hours`, or if a consensus is pending processing imminently.

Alleviation:

The Euler Finance team evaluated this exhibit and while they agree with the overall assessment provided, they disputed the exhibit's severity as they consider it informational.

Specifically, the discrepancy between the rate utilized by the `stETH` contract and the actual rate that the overall accounting system of the Lido project reflects is miniscule as evidenced by extensive use of the exchange rate as-is in multiple high-TVL markets.

After a re-assessment of the exhibit's impact coupled with the fact that the exhibit pertains to an exchange rate between **internal assets of the Lido ecosystem**, we concur with the Euler Finance team's view in relation to the exhibit's severity and consider it safely acknowledged.

PythOracle Manual Review Findings

POE-01M: Inexistent Configurability of Confidence Width

Type	Severity	Location
Standard Conformity	Informational	PythOracle.sol:L15

Description:

The `MAX_CONF_WIDTH` variable within the `PythOracle` is not configurable despite the oracle being meant to be compatible with multiple asset classes.

Impact:

A ±5% confidence interval for stablecoins is considered a significant deviation in contrast to high-volatility assets such as cryptocurrencies.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
14  /// @dev The confidence interval can be at most (-5%,+5%) wide.
15  uint256 internal constant MAX_CONF_WIDTH = 500;
```

Recommendation:

We advise the variable to be up to the deployer's discretion as stablecoins and other relatively stable assets would require a narrower confidence interval to avoid manipulation.

Alleviation:

The `MAX_CONF_WIDTH` constant has been removed from the contract in favour of the newly introduced `maxConfWidth` variable that is `immutable` and configured during the contract's deployment.

POE-02M: Inexistent Prevention of Overpayment

Type	Severity	Location
Standard Conformity	Informational	PythOracle.sol:L52

Description:

The `PythOracle::updatePrice` function permits the caller to overpay the `IPyth` oracle which we consider incorrect.

Impact:

A caller is permitted to overpay for a price update and the `IPyth` oracle **will not perform any refund of surplus funds**. While it is the responsibility of the caller to ensure sufficient native funds have been provided, the update fee may change between a transaction's submission and a transaction's execution in the network and as such dynamic evaluation of the necessary funds is a better approach.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
48  /// @notice Update the price of the Pyth feed.
49  /// @param updateData Price update data. Must be fetched off-chain.
50  /// @dev The required fee can be computed by calling `getUpdateFee` on Pyth with
the length of the `updateData` array.
51  function updatePrice(bytes[] calldata updateData) external payable {
52      IPyth(pyth).updatePriceFeeds{value: msg.value}(updateData);
53  }
```

Recommendation:

We advise the code to calculate the update fee via the `IPyth::getUpdateFee` function, and to forward the exact amount of native funds necessary.

The code can either mandate the exact amount of native funds necessary have been transmitted, or to refund any remainder to the caller, either of which we consider a valid alleviation.

Alleviation:

The exhibit has been rendered inapplicable due to the removal of the relevant function as part of efforts addressing `POE-01C`.

POE-03M: Potentially Unsupported Function Signature

Type	Severity	Location
Standard Conformity	Informational	PythOracle.sol:L44, L45

Description:

The code of the `PythOracle::constructor` will invoke the `IERC20::decimals` function as exposed by the `forge-std` library, however, the `IERC20::decimals` function **is not actually part of the EIP-20 specification**.

Impact:

Most **EIP-20** assets do implement the `IERC20::decimals` function signature, however, it is not mandated by the standard and as such a small subset of **EIP-20** tokens is incompatible with the `PythOracle` presently.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
32  /// @notice Deploy a PythOracle.
33  /// @param _pyth The address of the Pyth oracle proxy.
34  /// @param _base The address of the base asset corresponding to the feed.
35  /// @param _quote The address of the quote asset corresponding to the feed.
36  /// @param _feedId The id of the feed in the Pyth network.
37  /// @param _maxStaleness The maximum allowed age of the price.
38  constructor(address _pyth, address _base, address _quote, bytes32 _feedId,
uint256 _maxStaleness) {
39      pyth = _pyth;
40      base = _base;
41      quote = _quote;
```

Example (Cont.):

```sol
        feedId = _feedId;
        maxStaleness = _maxStaleness;
        baseDecimals = IERC20(_base).decimals();
        quoteDecimals = IERC20(_quote).decimals();
    }
```

Recommendation:

In case all **EIP-20** assets are expected to be supported, we advise decimals to either be opportunistically queried or for decimals to be supplied as input arguments thus permitting any token to have a `PythOracle` deployed.

Alleviation:

A common `BaseAdapter::_getDecimals` implementation has been introduced in the `BaseAdapter` upstream contract that will attempt to fetch the `IERC20::decimals` of an asset and default to `32` if they cannot be fetched.

As such, we consider this exhibit fully alleviated.

POE-04M: Improper Validation of Exponent

Type	Severity	Location
Logical Fault	Minor	PythOracle.sol:L74

Description:

The exponent of the Pyth Network system is guaranteed to be negative as it is meant to increase the accuracy of the reported value. A positive exponent would result in a reduction of the reported price and thus a loss of accuracy which would not be normal.

The aforementioned assumption is also upheld by the Pyth Network's own **Solidity SDK** which considers an exponent above `0` to be invalid.

Impact:

A positive exponent should be considered an invalid measurement as it would result in the data point being expanded and thus being inaccurate due to truncated values.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
72  function _fetchPriceStruct() internal view returns (PythStructs.Price memory) {
73      PythStructs.Price memory p = IPyth(pyth).getPriceNoOlderThan(feedId,
maxStaleness);
74      if (p.price <= 0 || p.conf > uint64(p.price) * MAX_CONF_WIDTH / 10_000 || p.expo
> 16 || p.expo < -16) {
75          revert Errors.PriceOracle_InvalidAnswer();
76      }
77      return p;
78  }
```

Recommendation:

We advise the code to ensure that the exponent is negative, and to potentially permit more decimals than `16` in the negative range.

Additionally, we advise the `price` to be validated as strictly positive given that a price of `0` should be considered invalid as well per the Pyth Network SDK.

Alleviation:

The Euler Finance team introduced exponent validation that prevents it from being positive as well as below the value of `-20`, rendering this exhibit alleviated.

For transparency, the Euler Finance team reached out to the Pyth Network team for clarification in relation to the permitted exponent values and concluded that positive exponents are ultimately meant to be supported even if the official SDK implies otherwise.

To this end, the changes introduced for this exhibit have been reverted in the latest implementation of the Euler Finance codebase and **the exhibit's recommendations should not be followed by other projects**.

RedstoneCoreOracle Manual Review Findings

RCO-01M: Improper Integration of Redstone On-Demand Feeds

Type	Severity	Location
Logical Fault	Informational	RedstoneCoreOracle.sol:L57-L67, L69-L81

Description:

The Redstone oracle system is meant to allow an on-demand price oracle system to be utilized by smart contracts by utilizing low-level mutations of a transaction's `calldata` to append signatures that validate a particular price measurement that the transaction may require.

The `RedstoneCoreOracle` implementation by the Euler Finance team attempts to incorporate the **on-demand price feeds** by utilizing a **push model** whereby "on-demand" price measurements are submitted via the `RedstoneCoreOracle::updatePrice` function and consequently consumed by the `RedstoneCoreOracle::_getQuote` function.

This contradicts the entire premise of the on-demand data points, as the `RedstoneCoreOracle` functions identically to the "classic" **data feeds** already provided by Redstone.

Impact:

We do not consider the current Redstone oracle integration standard or secure. The `RedstoneCoreOracle::_getQuote` function will not succeed if a price point has not been submitted via the `RedstoneCoreOracle::updatePrice` function, and the oracles of the Euler Finance Oracle repository **should not require active maintenance**.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

```sol
/// @notice Ingest a signed update message and cache it on the contract.
/// @dev Validation logic inherited from PrimaryProdDataServiceConsumerBase.
function updatePrice() external {
    // Use the cache if the previous price is still fresh.
    if (block.timestamp < lastUpdatedAt +
RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) return;

    uint256 price = getOracleNumericValueFromTxMsg(feedId);
    if (price > type(uint208).max) revert Errors.PriceOracle_Overflow();
    lastPrice = uint208(price);
    lastUpdatedAt = uint48(block.timestamp);
```

Example (Cont.):

```sol
}

/// @notice Get the quote from the Redstone feed.
/// @param inAmount The amount of `base` to convert.
/// @param _base The token that is being priced.
/// @param _quote The token that is the unit of account.
/// @return The converted amount using the Redstone feed.
function _getQuote(uint256 inAmount, address _base, address _quote) internal view override returns (uint256) {
    bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);

    uint256 staleness = block.timestamp - lastUpdatedAt;
    if (staleness > maxStaleness) revert Errors.PriceOracle_TooStale(staleness, maxStaleness);

    return ScaleUtils.calcOutAmount(inAmount, lastPrice, scale, inverse);
}
```

Recommendation:

In order to correctly integrate with Redstone, we advise either of the following two approaches to be incorporated to the `RedstoneCoreOracle` contract.

Approach A involves updating the system to utilize the "classic" oracles of the Redstone ecosystem, thereby no longer requiring the manual **`RedstoneCoreOracle::updatePrice`** mechanism and behaving similarly to the other oracles within the system that do not require active maintenance.

Approach B is more complicated, and involves taking advantage of the `ProxyConnector` of the Redstone ecosystem. Specifically, the **`RedstoneCoreOracle::_getQuote`** function is capable of processing Redstone oracle payloads if the top-level **`EulerRouter::getQuote`** / **`EulerRouter::getQuotes`** functions were mutated to forward the Redstone-related `calldata` via the **`ProxyConnector::proxyCalldataView` function**.

The second of the aforementioned approaches would ensure that the correct on-demand Redstone model is utilized, delegating responsibility of proper `calldata` appendment to the caller of the `EulerRouter` and preventing the contract from becoming inoperable unless a separate, top-level call to **`RedstoneCoreOracle::updatePrice`** is made.

Alleviation:

The Euler Finance team has evaluated our analysis and, after careful deliberation, consider it to not be reflective of the Euler Finance oracle system's actual purpose.

In detail, the Euler Finance team clarified that the EVK which is meant to integrate with the Euler Finance oracle system supports batch transactions that will permit the price of the oracle to be updated prior to its usage in a secure manner.

Furthermore, the Euler Finance team clarified that they consciously chose to integrate with the push model to allow protocols the option of utilizing such oracle providers.

As the exhibit relied on a misconception about the intentions of the Euler Finance oracle system in relation to data availability, we consider it to be invalid and have lowered its severity to informational to reflect this fact.

RCO-02M: Inexistent Capability of Functionality Overrides

Type	Severity	Location
Standard Conformity	Informational	RedstoneCoreOracle.sol:L16, L24

Description:

The official documentation of the Redstone system indicates that contracts which interface with their oracles should be upgradeable to be able to adjust their logic.

The rationale behind this approach is that the Redstone on-demand data feed system utilizes a signature threshold scheme, and compromised signers could generate incorrect data points that would be accepted by the `RedstoneCoreOracle`.

Impact:

In comparison to other price feed protocols such as Chainlink, the Redstone oracle on-demand system is validated locally and cannot be influenced by external actions. As such, there is no inherent "circuit-breaker" or other action that the Redstone team can take to invalidate the signers already registered by a `PrimaryProdDataServiceConsumerBase`.

In case of a systematic failure / compromise of Redstone's signers, the `RedstoneCoreOracle` will not be equipped to upgrade its logic or halt its operations in response to such an event.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

SOL

```sol
16  contract RedstoneCoreOracle is PrimaryProdDataServiceConsumerBase, BaseAdapter {
```

Recommendation:

We advise the contract to be made upgradeable, permitting functions such as the `PrimaryProdDataServiceConsumerBase::getAuthorisedSignerIndex` and `PrimaryProdDataServiceConsumerBase::getUniqueSignersThreshold` validation functions to be updated in case of a systematic failure of Redstone.

Alternatively, we advise a circuit breaker to be introduced, permitting price measurements via the contract to revert and thus prompting integrators to utilize alternative data feeds in case of the same scenario.

Alleviation:

The Euler Finance team evaluated this exhibit and contested its severity as the oracle adapters within the Euler Finance codebase are considered to be `immutable`, with the responsibility of using a correct oracle delegated to a higher-level implementation such as a router that will be capable of overwriting the oracle to be used for a particular asset pair with an updated variant.

As such, we consider this exhibit safely acknowledge and its severity diminished accordingly per the intended purpose of the Euler Finance oracle system.

RCO-03M: Potentially Unsupported Function Signature

Type	Severity	Location
Standard Conformity	Informational	RedstoneCoreOracle.sol:L52, L53

Description:

The code of the `RedstoneCoreOracle::constructor` will invoke the `IERC20::decimals` function as exposed by the `forge-std` library, however, the `IERC20::decimals` function **is not actually part of the EIP-20 specification**.

Impact:

Most **EIP-20** assets do implement the `IERC20::decimals` function signature, however, it is not mandated by the standard and as such a small subset of **EIP-20** tokens is incompatible with the `RedstoneCoreOracle` presently.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

SOL

```
36  /// @notice Deploy a RedstoneCoreOracle.
37  /// @param _base The address of the base asset corresponding to the feed.
38  /// @param _quote The address of the quote asset corresponding to the feed.
39  /// @param _feedId The identifier of the price feed.
40  /// @param _maxStaleness The maximum allowed age of the price.
41  /// @dev Base and quote are not required to correspond to the feed assets.
42  /// For example, the ETH/USD feed can be used to price WETH/USDC.
43  constructor(address _base, address _quote, bytes32 _feedId, uint256
_maxStaleness) {
44      if (_maxStaleness <
RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) {
45          revert Errors.PriceOracle_InvalidConfiguration();
```

Example (Cont.):

```sol
    }

    base = _base;
    quote = _quote;
    feedId = _feedId;
    maxStaleness = _maxStaleness;
    uint8 baseDecimals = IERC20(base).decimals();
    uint8 quoteDecimals = IERC20(quote).decimals();
    scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, FEED_DECIMALS);
}
```

Recommendation:

In case all **EIP-20** assets are expected to be supported, we advise decimals to either be opportunistically queried or for decimals to be supplied as input arguments thus permitting any token to have a `RedstoneCoreOracle` deployed.

Alleviation:

A common `BaseAdapter::_getDecimals` implementation has been introduced in the `BaseAdapter` upstream contract that will attempt to fetch the `IERC20::decimals` of an asset and default to `32` if they cannot be fetched.

As such, we consider this exhibit fully alleviated.

RCO-04M: Improper Assumption of Oracle Decimals

Type	Severity	Location
Standard Conformity	Minor	RedstoneCoreOracle.sol:L17, L54

Description:

The `RedstoneCoreOracle` contract assumes that all price points reported by Redstone will have `8` decimals of accuracy, however, no such trait is guaranteed.

Inspection of the **official Redstone oracle mono-repo** highlights that the system supports dynamic decimals across both its off-chain and on-chain infrastructure, and certain "classic" data feeds **explicitly yield a number different than** `8` for their decimals (**example**).

Impact:

Although most use-cases indicate that `8` decimals are in-use by Redstone, their system explicitly supports a different number of decimals and as such the `RedstoneCoreOracle` contract implementation should too.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

SOL

```sol
17  uint8 internal constant FEED_DECIMALS = 8;
18  /// @notice The address of the base asset corresponding to the feed.
19  address public immutable base;
20  /// @notice The address of the quote asset corresponding to the feed.
21  address public immutable quote;
22  /// @notice The identifier of the price feed.
23  /// @dev See https://app.redstone.finance/#/app/data-services/redstone-primary-prod
24  bytes32 public immutable feedId;
25  /// @notice The maximum allowed age of the price.
26  uint256 public immutable maxStaleness;
```

Example (Cont.):

SOL

```sol
/// @notice The scale factors used for decimal conversions.
Scale internal immutable scale;
/// @notice The last updated price.
/// @dev This gets updated after calling `updatePrice`.
uint208 public lastPrice;
/// @notice The timestamp of the last update.
/// @dev Gets updated ot `block.timestamp` after calling `updatePrice`.
uint48 public lastUpdatedAt;

/// @notice Deploy a RedstoneCoreOracle.
/// @param _base The address of the base asset corresponding to the feed.
/// @param _quote The address of the quote asset corresponding to the feed.
/// @param _feedId The identifier of the price feed.
/// @param _maxStaleness The maximum allowed age of the price.
/// @dev Base and quote are not required to correspond to the feed assets.
/// For example, the ETH/USD feed can be used to price WETH/USDC.
constructor(address _base, address _quote, bytes32 _feedId, uint256
_maxStaleness) {
    if (_maxStaleness <
RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) {
        revert Errors.PriceOracle_InvalidConfiguration();
    }

    base = _base;
    quote = _quote;
    feedId = _feedId;
    maxStaleness = _maxStaleness;
    uint8 baseDecimals = IERC20(base).decimals();
    uint8 quoteDecimals = IERC20(quote).decimals();
    scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, FEED_DECIMALS);
```

Example (Cont.):

```sol
}
```

Recommendation:

We advise the decimals to be user-configured on deployment, ensuring that they match the accuracy of the data point reported by the on-demand oracles of Redstone.

Alleviation:

The Euler Finance team evaluated this exhibit and, after confirming with the Redstone team directly, proceeded with adjusting decimals to be configurable instead of presumed as `8` effectively alleviating this exhibit.

RCO-05M: Misconceived Data Staleness

Type	Severity	Location
Logical Fault	Minor	RedstoneCoreOracle.sol:L44, L51, L77-L78

Description:

The data staleness validated by the `RedstoneCoreOracle::_getQuote` function is slightly non-standard as the `block.timestamp` recorded by the `RedstoneCoreOracle::updatePrice` function does not correspond to the actual timestamp the price was recorded in.

In detail, the `RedstoneCoreOracle::updatePrice` function will accept a price reported at a timestamp that is up to 3 minutes behind or 1 minute ahead of the current `block.timestamp`. As a result, the actual "staleness" validated by the `RedstoneCoreOracle::_getQuote` function is `staleness - 3 minutes` or `staleness + 1 minutes`.

The latter of the two is of no concern as a fresher-than-considered price is evaluated, however, the former of the two would result in potentially incorrect data staleness being enforced.

Impact:

The actual data staleness enforced by the `RedstoneCoreOracle::_getQuote` differs from the one that the user has specified, permitting a data point that is `maxStaleness + RedstoneDefaultsLib::DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS` in the past to be consumed.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

```sol
/// @notice Deploy a RedstoneCoreOracle.
/// @param _base The address of the base asset corresponding to the feed.
/// @param _quote The address of the quote asset corresponding to the feed.
/// @param _feedId The identifier of the price feed.
/// @param _maxStaleness The maximum allowed age of the price.
/// @dev Base and quote are not required to correspond to the feed assets.
/// For example, the ETH/USD feed can be used to price WETH/USDC.
constructor(address _base, address _quote, bytes32 _feedId, uint256 _maxStaleness) {
    if (_maxStaleness < RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) {
        revert Errors.PriceOracle_InvalidConfiguration();
```

Example (Cont.):

```sol
    }

    base = _base;
    quote = _quote;
    feedId = _feedId;
    maxStaleness = _maxStaleness;
    uint8 baseDecimals = IERC20(base).decimals();
    uint8 quoteDecimals = IERC20(quote).decimals();
    scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals, FEED_DECIMALS);
}

/// @notice Ingest a signed update message and cache it on the contract.
/// @dev Validation logic inherited from PrimaryProdDataServiceConsumerBase.
function updatePrice() external {
    // Use the cache if the previous price is still fresh.
    if (block.timestamp < lastUpdatedAt +
RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) return;

    uint256 price = getOracleNumericValueFromTxMsg(feedId);
    if (price > type(uint208).max) revert Errors.PriceOracle_Overflow();
    lastPrice = uint208(price);
    lastUpdatedAt = uint48(block.timestamp);
}

/// @notice Get the quote from the Redstone feed.
/// @param inAmount The amount of `base` to convert.
/// @param _base The token that is being priced.
/// @param _quote The token that is the unit of account.
/// @return The converted amount using the Redstone feed.
```

Example (Cont.):

```sol
function _getQuote(uint256 inAmount, address _base, address _quote) internal view override returns (uint256) {
    bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);

    uint256 staleness = block.timestamp - lastUpdatedAt;
    if (staleness > maxStaleness) revert Errors.PriceOracle_TooStale(staleness, maxStaleness);

    return ScaleUtils.calcOutAmount(inAmount, lastPrice, scale, inverse);
}
```

Recommendation:

We advise the `maxStaleness` value to either be stored as `_maxStaleness - RedstoneDefaultsLib::DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS`, or to be utilized in `RedstoneCoreOracle::_getQuote` as `maxStaleness - RedstoneDefaultsLib::DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS`.

Alleviation:

The oracle was refactored to employ two distinct staleness concepts; one applicable to the actual price measurements from the Redstone system and the other applicable to the local cache system.

In addition to these changes, the Euler Finance team introduced documentation highlighting that a user can choose the most suitable Redstone price in the `[block.timestamp - maxPriceStaleness, block.timestamp]` interval and consider it an unavoidable trait inherent to any on-chain integration.

As the staleness concepts have been properly distinguished and the inherent risk of the Redstone oracle integration has been adequately documented, we consider all possible remediative actions in relation to this exhibit to have been carried out in the codebase.

RethOracle Manual Review Findings

ROE-01M: Potentially Stale Calculation of Exchange Rate (Asynchronous Rewards / Penalties)

Type	Severity	Location
External Call Validation	Minor	RethOracle.sol:L33, L35

Description:

The `rETH` contract implementation exposes utility functions for calculating the **on-chain conversion of the two assets** rather than the exchange rate that is influenced by the "true" value of the asset. In practical terms, the `IReth::getEthValue` and `IReth::getRethValue` functions do not reflect future profits **already captured** that the Rocket Pool contracts will reflect via a manual update of the Rocket Pool's network balances.

Per the Rocket Pool documentation, the balances of the Rocket Pool system are updated at a 24-hour interval providing adequate time for a user to be knowledgeable of the shift in the exchange rate and to capitalize on it.

Impact:

The **`RethOracle::_getQuote`** function will consistently undervalue the `rETH` asset in relation to the `ETH` asset due to not accounting for earned but not yet reflected rewards of the Beacon Chain node operators that are affiliated with Rocket Pool. Additionally, the same function may overvalue the `rETH` asset in relation to the `ETH` asset if losses incurred by Rocket Pool have not yet been reflected on-chain.

Example:

src/adapter/rocketpool/RethOracle.sol

```sol
/// @notice Get a quote by querying the exchange rate from the rETH contract.
/// @dev Calls `getEthValue` for rETH/WETH and `getRethValue` for WETH/rETH.
/// @param inAmount The amount of `base` to convert.
/// @param base The token that is being priced. Either `rETH` or `WETH`.
/// @param quote The token that is the unit of account. Either `WETH` or `rETH`.
/// @return The converted amount.
function _getQuote(uint256 inAmount, address base, address quote) internal view override returns (uint256) {
    if (base == reth && quote == weth) {
        return IReth(reth).getEthValue(inAmount);
    } else if (base == weth && quote == reth) {
```

Example (Cont.):

```sol
            return IReth(reth).getRethValue(inAmount);
        }
        revert Errors.PriceOracle_NotSupported(base, quote);
    }
```

Recommendation:

The Rocketpool ecosystem contains a `RocketNetworkBalances` deployment that marks the last block that a balance update occurred. This value can be utilized to detect the time that has elapsed since the last balance update, and prevent an exchange rate from being considered as valid if the time exceeds `24 hours` or a smaller duration.

Alternatively or as an additional measure, the code could query the TWAP exchange rate of an `ETH-rETH` AMM and enforce a deviation threshold between the two price measurements.

Alleviation:

The Euler Finance team evaluated this exhibit and, after a careful assessment and economical analysis of the operational security and accuracy of the exchange rate provided by the Rocketpool contracts, proceeded with removing the oracle implementation from the codebase altogether.

As such, we consider this exhibit indirectly alleviated as it prompted an economical analysis of the Rocketpool oracle that resulted in a tangible adjustment to the codebase.

SDaiOracle Manual Review Findings

SDO-01M: Insecure Usage of Outdated Interest Rate Accumulator

Type	Severity	Location
Logical Fault	Medium	SDaiOracle.sol:L37, L39

Description:

The `IPot::chi` member is meant to indicate the latest interest rate accumulator, however, the `SDaiOracle::_getQuote` function integrates with it incorrectly as the `IPot::rho` value (the last time the interest accumulator was updated) may not match the current timestamp.

Impact:

The `SDaiOracle::_getQuote` function will constantly over-value the `DAI` asset in relation to the `sDAI` asset due to not calculating an up-to-date cumulative interest rate value from the `dsrPot`.

Example:

src/adapter/maker/SDaiOracle.sol

SOL

```sol
29  /// @notice Get a quote by querying the exchange rate from the DSR Pot contract.
30  /// @dev Calls `chi`, the interest rate accumulator, to get the exchange rate.
31  /// @param inAmount The amount of `base` to convert.
32  /// @param base The token that is being priced. Either `sDai` or `dai`.
33  /// @param quote The token that is the unit of account. Either `dai` or `sDai`.
34  /// @return The converted amount.
35  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
36      if (base == sDai && quote == dai) {
37          return inAmount * IPot(dsrPot).chi() / 1e27;
38      } else if (base == dai && quote == sDai) {
```

Example (Cont.):

```sol
            return inAmount * 1e27 / IPot(dsrPot).chi();
        }
        revert Errors.PriceOracle_NotSupported(base, quote);
    }
```

Recommendation:

As invoking `IPot::drip` is impossible due to the function being state-mutating, we advise the statements within the pot's `IPot::drip` function to be replicated locally to calculate the latest up-to-date cumulative interest rate (`chi`) value.

To achieve this, the calculation `rmul(rpow(dsr, now - rho, ONE), chi)` needs to be replicated locally by fetching the `IPot::dsr`, `IPot::rho`, and `IPot::chi` values from the `dsrPot` and performing the `WadRayMath` based calculations.

Alleviation:

The accurate exchange rate between the `DAI` and `sDAI` assets is now calculated via the newly introduced `SDaiOracle::_getExchangeRate` function using the `Solady` library for performing the `rpow` operation, effectively alleviating this exhibit per our recommendation.

ScaleUtils Manual Review Findings

SUS-01M: Potential Increase of Acceptable Values

Type	Severity	Location
Mathematical Operations	Informational	ScaleUtils.sol:L79, L82

Description:

The `ScaleUtils::calcOutAmount` function will perform a sequence of calculations which may result in interim overflow operations that could potentially be avoided by restructuring the arithmetic order of operations.

Specifically, the calculation `inAmount * unitPrice` may overflow when executing the code's `else` path, however, the `inAmount * priceScale` calculation may succeed and the result of the `result * unitPrice / feedScale` with arbitrary precision may be calculate-able with an end-result that fits within a `256` bit number.

Similarly, the `if` branch of the code executes a `priceScale * unitPrice` calculation that may underflow when the divisions can be performed in sequence.

Impact:

The current approach is sound, and in most circumstances the overflow would occur in an upstream call that would utilize the result of the function.

In any case, as the `ScaleUtils::calcOutAmount` function is meant to be a library we consider an increase of its operational range to be potentially desirable.

Example:

src/lib/ScaleUtils.sol

```sol
/// @notice Convert the price by applying scale factors.
/// @param inAmount The amount of `base` to convert.
/// @param unitPrice The unit price reported by the feed.
/// @param scale The scale factors returned by `calcScale`.
/// @param inverse Whether to price base/quote or quote/base.
/// @return The resulting outAmount.
function calcOutAmount(uint256 inAmount, uint256 unitPrice, Scale scale, bool inverse)
    internal
    pure
    returns (uint256)
```

Example (Cont.):

```sol
{
    uint256 priceScale = (Scale.unwrap(scale) << 128) >> 128;
    uint256 feedScale = Scale.unwrap(scale) >> 128;
    if (inverse) {
        // (inAmount * feedScale) / (priceScale * unitPrice)
        return FixedPointMathLib.fullMulDiv(inAmount, feedScale, priceScale * unitPrice);
    } else {
        // (inAmount * unitPrice * priceScale) / feedScale
        return FixedPointMathLib.fullMulDiv(inAmount * unitPrice, priceScale, feedScale);
    }
}
```

Recommendation:

We advise the code to potentially accommodate for these cases, re-ordering the arithmetic sequences if any of the interim calculations overflow and thus increasing the total range of values that the `ScaleUtils::calcOutAmount` function will produce a result for.

Alleviation:

The Euler Finance team evaluated this exhibit and specified that the current safe operational range of the function is adequate for their intents and purposes.

SUS-02M: Potential Negation Overflow

Type	Severity	Location
Mathematical Operations	Informational	ScaleUtils.sol:L61

Description:

The referenced statement will perform a negation without validating that the `diff` value will properly fit in the `int8` data type if negated.

Impact:

As evidenced in exhibits within the `PythOracle`, this particular code segment will not be executed. Additionally, the negation would overflow and thus the code would simply `revert` as negations are performed using checked arithmetic.

As such, this exhibit cannot constitute a vulnerability that is higher than informational.

Example:

src/lib/ScaleUtils.sol

SOL

```sol
53  /// @notice Calculate the scale factors for converting a unit price.
54  /// @param baseDecimals The decimals of the base asset.
55  /// @param quoteDecimals The decimals of the quote asset.
56  /// @param priceDecimals The decimals of the feed price, not incorporated into
the price.
57  /// @return The scale factors used for price conversions.
58  function calcScale(uint8 baseDecimals, uint8 quoteDecimals, int8 priceDecimals)
internal pure returns (Scale) {
59      int8 diff = int8(baseDecimals) - priceDecimals;
60      if (diff > 0) return from(quoteDecimals, uint8(diff));
61      else return from(quoteDecimals + uint8(-diff), 0);
62  }
```

Recommendation:

We advise the code to ensure the `diff` value is different than the minimum possible (`type(int8).min`), yielding a descriptive error message in such a case as the negation would overflow.

Alleviation:

The relevant function of this exhibit was removed as part of remediative efforts for **`POE-02C`**, rendering this exhibit inapplicable.

UniswapV3Oracle Manual Review Findings

UVO-01M: Inexistent Validation of Observation Cardinality Length

Type	Severity	Location
Standard Conformity	Informational	UniswapV3Oracle.sol:L40

Description:

The Uniswap V3 TWAP system relies on a set of observations made within a finite-length array that is initialized at `1`. This means that a TWAP of a Uniswap V3 pool is insecure to utilize unless its cardinality has been expanded to accommodate for the TWAP window that the caller wishes to query for.

As detailed extensively in the Uniswap V3 documentation, the cardinality of a pool should be increased the longer the TWAP window is desired to be by invoking the `IUniswapV3Pool::increaseObservationCardinalityNext` function.

Impact:

A `UniswapV3Oracle` may fail to execute properly out-of-the-box due to not validating the cardinality of the pool that it is deployed for.

Example:

src/adapter/uniswap/UniswapV3Oracle.sol

SOL

```sol
44  /// @notice Get a quote by calling the pool's TWAP oracle.
45  /// @param inAmount The amount of `base` to convert.
46  /// @param base The token that is being priced. Either `tokenA` or `tokenB`.
47  /// @param quote The token that is the unit of account. Either `tokenB` or
`tokenA`.
48  /// @return The converted amount.
49  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
50      if (!((base == tokenA && quote == tokenB) || (base == tokenB && quote ==
tokenA))) {
51          revert Errors.PriceOracle_NotSupported(base, quote);
52      }
53      // Size limitation enforced by the pool.
```

Example (Cont.):

```sol
        if (inAmount > type(uint128).max) revert Errors.PriceOracle_Overflow();

        uint32[] memory secondsAgos = new uint32[](2);
        secondsAgos[0] = twapWindow;

        // Calculate the mean tick over the twap window.
        (int56[] memory tickCumulatives,) = IUniswapV3Pool(pool).observe(secondsAgos);
        int24 tick = int24((tickCumulatives[1] - tickCumulatives[0]) /
int32(twapWindow));
        return OracleLibrary.getQuoteAtTick(tick, uint128(inAmount), base, quote);
    }
```

Recommendation:

The cardinality required per minute varies from chain-to-chain, and represents the number of samples that should be taken per minute for a particular token to be considered accurate.

For example, in the Ethereum mainnet, a block is generated at a median of `12` seconds meaning that each minute has `5` blocks within. As such, a cardinality of `4-5` is considered acceptable per minute.

The cardinality at which the pool is set to should be evaluated during its **`UniswapV3Oracle::constructor`** and as such, we advise an `IUniswapV3Pool::increaseObservationCardinalityNext` function call to be introduced to it with an argument generated via a `_cardinaltyPerMinute` input argument.

Alleviation:

The Euler Finance team opted to acknowledge this exhibit as they do not consider it a vulnerability given that the `UniswapV3Oracle` requires further preparation beyond cardinality before it is deployed and utilized.

As such, we consider this exhibit as a safely acknowledged usability enhancement.

UVO-02M: Insecure Typecasting Operation (TWAP)

Type	Severity	Location
Input Sanitization	Informational	UniswapV3Oracle.sol:L61

Description:

The `UniswapV3Oracle::_getQuote` function will cast its configured `twapWindow` to an `int32` data type which the `uint32` data type may not fit into.

Impact:

An incorrectly configured `twapWindow` would result in an incorrect calculation of the arithmetic mean tick. To note, a severity of `informational` has been assigned as the likelihood of such an incident in a production environment is low due to the impossibly-high required number of observations necessary for the `IUniswapV3Pool::observe` lookup to succeed.

Example:

src/adapter/uniswap/UniswapV3Oracle.sol

SOL

```sol
44  /// @notice Get a quote by calling the pool's TWAP oracle.
45  /// @param inAmount The amount of `base` to convert.
46  /// @param base The token that is being priced. Either `tokenA` or `tokenB`.
47  /// @param quote The token that is the unit of account. Either `tokenB` or
`tokenA`.
48  /// @return The converted amount.
49  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
50      if (!((base == tokenA && quote == tokenB) || (base == tokenB && quote ==
tokenA))) {
51          revert Errors.PriceOracle_NotSupported(base, quote);
52      }
53      // Size limitation enforced by the pool.
```

Example (Cont.):

```sol
        if (inAmount > type(uint128).max) revert Errors.PriceOracle_Overflow();

        uint32[] memory secondsAgos = new uint32[](2);
        secondsAgos[0] = twapWindow;

        // Calculate the mean tick over the twap window.
        (int56[] memory tickCumulatives,) =
IUniswapV3Pool(pool).observe(secondsAgos);
        int24 tick = int24((tickCumulatives[1] - tickCumulatives[0]) /
int32(twapWindow));
        return OracleLibrary.getQuoteAtTick(tick, uint128(inAmount), base, quote);
    }
```

Recommendation:

We advise the `UniswapV3Oracle::constructor` to ensure that the `_twapWindow` being configured is less-than-or-equal-to the value of `type(int32).max` which is less than the maximum value of a `uint32` variable.

Alleviation:

The `UniswapV3Oracle::constructor` was updated to further sanitize the `_twapWindow` as being a value that does not exceed the maximum representable by the `int32` data type, effectively alleviating this exhibit.

UVO-03M: Insecure Calculation of Mean Tick

Type	Severity	Location
Mathematical Operations	Minor	UniswapV3Oracle.sol:L61

Description:

In the original Uniswap V3 implementation of the `OracleLibrary::consult` function, the delta between two cumulative tick values is permitted to underflow. Additionally, the median `tick` calculated should always **round towards negative infinity** as otherwise a rounding error may result in a rounding operation that would over-evaluate the tick and thus potential quote.

Impact:

Observations by the `UniswapV3Oracle::_getQuote` may fail to execute under extreme cumulative tick circumstances, and will incorrectly calculate the arithmetic mean tick when the delta between the cumulative values is negative.

Example:

src/adapter/uniswap/UniswapV3Oracle.sol

SOL

```sol
44  /// @notice Get a quote by calling the pool's TWAP oracle.
45  /// @param inAmount The amount of `base` to convert.
46  /// @param base The token that is being priced. Either `tokenA` or `tokenB`.
47  /// @param quote The token that is the unit of account. Either `tokenB` or
`tokenA`.
48  /// @return The converted amount.
49  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
50      if (!((base == tokenA && quote == tokenB) || (base == tokenB && quote ==
tokenA))) {
51          revert Errors.PriceOracle_NotSupported(base, quote);
52      }
53      // Size limitation enforced by the pool.
```

Example (Cont.):

```sol
        if (inAmount > type(uint128).max) revert Errors.PriceOracle_Overflow();

        uint32[] memory secondsAgos = new uint32[](2);
        secondsAgos[0] = twapWindow;

        // Calculate the mean tick over the twap window.
        (int56[] memory tickCumulatives,) =
IUniswapV3Pool(pool).observe(secondsAgos);
        int24 tick = int24((tickCumulatives[1] - tickCumulatives[0]) /
int32(twapWindow));
        return OracleLibrary.getQuoteAtTick(tick, uint128(inAmount), base, quote);
    }
```

Recommendation:

We advise the code to be made functionally identical to its original Uniswap V3 counterpart, wrapping the subtraction in an `unchecked` code block and ensuring that the resulting `tick` rounds towards negative infinity by performing the conditional and subtraction referenced by this exhibit.

Alleviation:

The downward-rounding mechanism has been copied based on the `OracleLibrary` of Uniswap V3 as advised, however, the `unchecked` statement advised was not introduced as the official Uniswap V3 implementation deviates between its `v0.8` and non-`v0.8` variants.

We consider both the `v0.8` and the non-`v0.8` variants as correct with checked arithmetic providing additional peace of mind rendering this exhibit fully addressed.

UVO-04M: Potentially Insecure TWAP Window

Type	Severity	Location
Standard Conformity	Minor	UniswapV3Oracle.sol:L15

Description:

The minimum TWAP window enforced by the `UniswapV3Oracle` is insecure for low liquidity pools and from a Proof-of-Stake perspective as it is likely for a single validator to generate all blocks within the minimum TWAP window.

Impact:

The present minimum TWAP window recommendation may be insufficient for certain use cases and under a Proof-of-Stake scheme.

Example:

src/adapter/uniswap/UniswapV3Oracle.sol

SOL

```sol
14  /// @dev The minimum length of the TWAP window is 1 minute.
15  uint32 internal constant MIN_TWAP_WINDOW = 60 seconds;
```

Recommendation:

We advise the minimum to be increased to a figure of `30 minutes`, ensuring a sufficient order book depth is observed in the AMM pair and minimizing the risk of a PoS validator manipulating the transactions within the blocks they are mining to execute a TWAP manipulation attack with no downside.

Alleviation:

The `MIN_TWAP_WINDOW` was updated to `5 minutes` and PoS related warnings were introduced to the contract's documentation, rendering this exhibit as addressed to the greatest extent it can be.

EulerRouter Code Style Findings

ERR-01C: Imprecise Terminology

Type	Severity	Location
Code Style	Informational	EulerRouter.sol:L77-L78, L85-L86

Description:

The referenced `if` conditionals will yield the **mutated** `inAmount` if the `base` and `quote` assets match which is misleading as the `inAmount` might have been mutated as part of **EIP-4626** vault conversions and thus be an `outAmount` value.

Example:

src/EulerRouter.sol

SOL

```
73  /// @inheritdoc IPriceOracle
74  function getQuote(uint256 inAmount, address base, address quote) external view
returns (uint256) {
75      address oracle;
76      (inAmount, base, quote, oracle) = _resolveOracle(inAmount, base, quote);
77      if (base == quote) return inAmount;
78      return IPriceOracle(oracle).getQuote(inAmount, base, quote);
79  }
80
81  /// @inheritdoc IPriceOracle
82  function getQuotes(uint256 inAmount, address base, address quote) external view
returns (uint256, uint256) {
```

Example (Cont.):

```sol
        address oracle;
        (inAmount, base, quote, oracle) = _resolveOracle(inAmount, base, quote);
        if (base == quote) return (inAmount, inAmount);
        return IPriceOracle(oracle).getQuotes(inAmount, base, quote);
    }
```

Recommendation:

As the `inAmount` might be re-used as an optimization, we advise extensive documentation to be introduced to the **`EulerRouter::getQuote`** and **`EulerRouter::getQuotes`** functions while the return argument of the **`EulerRouter::_resolveOracle`** should have its documentation updated to a different name than `inAmount` that is more accurate of its purpose.

Alleviation:

The documentation of the **`EulerRouter::_resolveOracle`** was updated to highlight that the returned value is a `resolvedAmount` rather than `inAmount` which we consider an adequate terminology change to address this exhibit.

Governable Code Style Findings

GEL-01C: Redundant Variable Caching

Type	Severity	Location
Gas Optimization	Informational	Governable.sol:L40, L42

Description:

The referenced declaration will read the value of `governor` from storage, write to it, and then emit an event for its adjustment.

Example:

src/lib/Governable.sol

SOL

```sol
/// @notice Set the governor address.
/// @param newGovernor The address of the new governor.
function _setGovernor(address newGovernor) internal {
    address oldGovernor = governor;
    governor = newGovernor;
    emit GovernorSet(oldGovernor, newGovernor);
}
```

Recommendation:

We advise the code to instead emit the event before the adjustment takes place, permitting the `governor` token to be directly utilized in the event's emission and thus optimizing the code's memory expansion costs.

Alleviation:

The `oldGovernor` local variable has been omitted per our recommendation, emitting the `GovernorSet` event prior to mutating the `governor` entry optimally.

LidoOracle Code Style Findings

LOE-01C: Potential Usage of Library

Type	Severity	Location
Code Style	Informational	LidoOracle.sol:L32, L34, L37

Description:

The referenced statements can be executed via the `ScaleUtils::getDirectionOrRevert` function.

Example:

src/adapter/lido/LidoOracle.sol

SOL

```
25  /// @notice Get a quote by querying the exchange rate from the stEth contract.
26  /// @dev Calls `getSharesByPooledEth` for stEth/wstEth and `getPooledEthByShares`
for wstEth/stEth.
27  /// @param inAmount The amount of `base` to convert.
28  /// @param base The token that is being priced. Either `stEth` or `wstEth`.
29  /// @param quote The token that is the unit of account. Either `wstEth` or
`stEth`.
30  /// @return The converted amount.
31  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
32      if (base == stEth && quote == wstEth) {
33          return IStEth(stEth).getSharesByPooledEth(inAmount);
34      } else if (base == wstEth && quote == stEth) {
```

Example (Cont.):

```sol
            return IStEth(stEth).getPooledEthByShares(inAmount);
        }
        revert Errors.PriceOracle_NotSupported(base, quote);
    }
```

Recommendation:

We advise this to be done so, optimizing the code's legibility.

Alleviation:

The Euler Finance team stated that they do not consider the recommended change a legibility improvement as they wish straightforward wrapped variant oracles to utilize a direct comparison.

Given that the approach the Euler Finance team wishes to retain is uniformly applied to the class of oracles described, we consider this exhibit as an inapplicable recommendation and concur with the Euler Finance team's assessment.

PythOracle Code Style Findings

POE-01C: Potentially Redundant Function Implementation

Type	Severity	Location
Standard Conformity	Informational	PythOracle.sol:L48-L53

Description:

In contrast to the `RedstoneCoreOracle::updatePrice` function that records local data points, the `PythOracle::updatePrice` function will simply forward a call to a permissionless function of the `IPyth` implementation.

Impact:

As the `PythOracle::updatePrice` function does not expose any novel statements and simply forwards a call with no additional logic, its presence is unnecessary and integrators should make sure the oracles are updated whenever necessary.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
48  /// @notice Update the price of the Pyth feed.
49  /// @param updateData Price update data. Must be fetched off-chain.
50  /// @dev The required fee can be computed by calling `getUpdateFee` on Pyth with
the length of the `updateData` array.
51  function updatePrice(bytes[] calldata updateData) external payable {
52      IPyth(pyth).updatePriceFeeds{value: msg.value}(updateData);
53  }
```

Recommendation:

We advise the function to be removed altogether, ensuring the caller is responsible for updating the `IPyth` oracle data points if necessary.

Alleviation:

The relevant function has been properly removed from the `PythOracle` implementation, and the Euler Finance team has clarified that callers are expected to update the Pyth oracle price via a batch EVK call directly.

POE-02C: Redundant Handling of Positive Exponent

Type	Severity	Location
Gas Optimization	Informational	PythOracle.sol:L66

Description:

Per the Pyth Network Solidity SDK **validation** as well as what a positive exponent would infer, the Pyth Network is expected to report solely non-positive exponent values during its operation. As a result, custom handling of a potentially positive exponent in the `ScaleUtils` implementation is unnecessary.

Example:

src/adapter/pyth/PythOracle.sol

SOL

```sol
55  /// @notice Fetch the latest Pyth price and transform it to a quote.
56  /// @param inAmount The amount of `base` to convert.
57  /// @param _base The token that is being priced.
58  /// @param _quote The token that is the unit of account.
59  /// @return The converted amount.
60  function _getQuote(uint256 inAmount, address _base, address _quote) internal view
override returns (uint256) {
61      bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);
62
63      PythStructs.Price memory priceStruct = _fetchPriceStruct();
64      uint256 price = uint256(uint64(priceStruct.price));
```

Example (Cont.):

```sol
65
66      Scale scale = ScaleUtils.calcScale(baseDecimals, quoteDecimals,
int8(priceStruct.expo));
67      return ScaleUtils.calcOutAmount(inAmount, price, scale, inverse);
68  }
69
70  /// @notice Get the latest Pyth price and perform sanity checks.
71  /// @dev Reverts if price is non-positive, confidence is too wide, or exponent is
too large.
72  function _fetchPriceStruct() internal view returns (PythStructs.Price memory) {
73      PythStructs.Price memory p = IPyth(pyth).getPriceNoOlderThan(feedId,
maxStaleness);
74      if (p.price <= 0 || p.conf > uint64(p.price) * MAX_CONF_WIDTH / 10_000 || p.expo
> 16 || p.expo < -16) {
75          revert Errors.PriceOracle_InvalidAnswer();
76      }
77      return p;
78  }
```

Recommendation:

We advise the `priceStruct.expo` to be cast to a `uint8` after being negated, optimizing the bytecode size of the contracts whilst reducing their complexity.

Alleviation:

Alongside remediations carried out for **`POE-04M`**, the referenced statement was optimized to invoke **`ScaleUtils::from`** directly thus addressing this exhibit.

RedstoneCoreOracle Code Style Findings

RCO-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	RedstoneCoreOracle.sol:L77

Description:

The linked mathematical operation is guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/adapter/redstone/RedstoneCoreOracle.sol

SOL

```sol
57  /// @notice Ingest a signed update message and cache it on the contract.
58  /// @dev Validation logic inherited from PrimaryProdDataServiceConsumerBase.
59  function updatePrice() external {
60      // Use the cache if the previous price is still fresh.
61      if (block.timestamp < lastUpdatedAt +
RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS) return;
62
63      uint256 price = getOracleNumericValueFromTxMsg(feedId);
64      if (price > type(uint208).max) revert Errors.PriceOracle_Overflow();
65      lastPrice = uint208(price);
66      lastUpdatedAt = uint48(block.timestamp);
```

Example (Cont.):

```sol
}

/// @notice Get the quote from the Redstone feed.
/// @param inAmount The amount of `base` to convert.
/// @param _base The token that is being priced.
/// @param _quote The token that is the unit of account.
/// @return The converted amount using the Redstone feed.
function _getQuote(uint256 inAmount, address _base, address _quote) internal view override returns (uint256) {
    bool inverse = ScaleUtils.getDirectionOrRevert(_base, base, _quote, quote);

    uint256 staleness = block.timestamp - lastUpdatedAt;
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statement to be wrapped in an `unchecked` code block thereby optimizing its execution cost.

Alleviation:

The Euler Finance team considered the optimization outlined by the exhibit and opted to retain the current implementation in place.

RethOracle Code Style Findings

ROE-01C: Potential Usage of Library

Type	Severity	Location
Code Style	Informational	RethOracle.sol:L32, L34, L37

Description:

The referenced statements can be executed via the `ScaleUtils::getDirectionOrRevert` function.

Example:

src/adapter/rocketpool/RethOracle.sol

SOL

```sol
32  if (base == reth && quote == weth) {
33      return IReth(reth).getEthValue(inAmount);
34  } else if (base == weth && quote == reth) {
35      return IReth(reth).getRethValue(inAmount);
36  }
37  revert Errors.PriceOracle_NotSupported(base, quote);
```

Recommendation:

We advise this to be done so, optimizing the code's legibility.

Alleviation:

The Euler Finance team has proceeded with removing the `RethOracle` from the codebase in light of exhibit `ROE-01M`, rendering this exhibit no longer applicable.

SDaiOracle Code Style Findings

SDO-01C: Potential Usage of Library

Type	Severity	Location
Code Style	Informational	SDaiOracle.sol:L36, L38, L41

Description:

The referenced statements can be executed via the `ScaleUtils::getDirectionOrRevert` function.

Example:

src/adapter/maker/SDaiOracle.sol

SOL

```sol
29  /// @notice Get a quote by querying the exchange rate from the DSR Pot contract.
30  /// @dev Calls `chi`, the interest rate accumulator, to get the exchange rate.
31  /// @param inAmount The amount of `base` to convert.
32  /// @param base The token that is being priced. Either `sDai` or `dai`.
33  /// @param quote The token that is the unit of account. Either `dai` or `sDai`.
34  /// @return The converted amount.
35  function _getQuote(uint256 inAmount, address base, address quote) internal view
override returns (uint256) {
36      if (base == sDai && quote == dai) {
37          return inAmount * IPot(dsrPot).chi() / 1e27;
38      } else if (base == dai && quote == sDai) {
```

Example (Cont.):

```sol
            return inAmount * 1e27 / IPot(dsrPot).chi();
        }
        revert Errors.PriceOracle_NotSupported(base, quote);
    }
```

Recommendation:

We advise this to be done so, optimizing the code's legibility.

Alleviation:

The Euler Finance team stated that they do not consider the recommended change a legibility improvement as they wish straightforward wrapped variant oracles to utilize a direct comparison.

Given that the approach the Euler Finance team wishes to retain is uniformly applied to the class of oracles described, we consider this exhibit as an inapplicable recommendation and concur with the Euler Finance team's assessment.

SDO-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	SDaiOracle.sol:L37, L39

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/adapter/maker/SDaiOracle.sol

SOL

```sol
37  return inAmount * IPot(dsrPot).chi() / 1e27;
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation:

The referenced value literal `1e27` has been properly relocated to a contract-level `constant` declaration labelled `RAY`, optimizing the code's legibility.

ScaleUtils Code Style Findings

SUS-01C: Inefficient Erasure of Upper Bits

Type	Severity	Location
Gas Optimization	Informational	ScaleUtils.sol:L75

Description:

The referenced statement is meant to zero out the upper bits of the `scale` variable's raw representation to acquire its lower `128` bits, however, it does so inefficiently as it performs two bitwise shift operations.

Example:

src/lib/ScaleUtils.sol

SOL

```
75  uint256 priceScale = (Scale.unwrap(scale) << 128) >> 128;
```

Recommendation:

We advise a bitwise mask to be utilized via an `AND` (`&`) operation, requiring a single EVM instruction to acquire the desired value.

Alleviation:

The code was updated to utilize a `PRICE_SCALE_MASK` with a bitwise `AND` operation as advised, optimizing the code's gas cost whilst increasing the statement's legibility.

SUS-02C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	ScaleUtils.sol:L25, L75, L76

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/lib/ScaleUtils.sol

SOL

```sol
25  return Scale.wrap((10 ** feedExponent << 128) | 10 ** priceExponent);
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation:

The Euler Finance team evaluated this recommendation and opted to not relocate the value of `10` to a `constant` declaration as they do not consider it a legibility enhancement.

We concur with this assessment and in view of the overall codebase's style and thus mark this exhibit as invalid.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.