Zellic

Prepared for
Mike Stone
Chaos Labs

Prepared by
Seunghyeon Kim
Chongyu Lv
Zellic

May 13, 2025

Benqi Oracle

Smart Contract Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Chaos Labs from May 8th to May 9th, 2025. During this engagement, Zellic reviewed Benqi Oracle's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the access control implemented well?
- Is the price oracles' stale-checking logic legitimate?
- Are there any unhandled cases for the price oracle?
- Are there any miscalculations for the precision and rounding?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

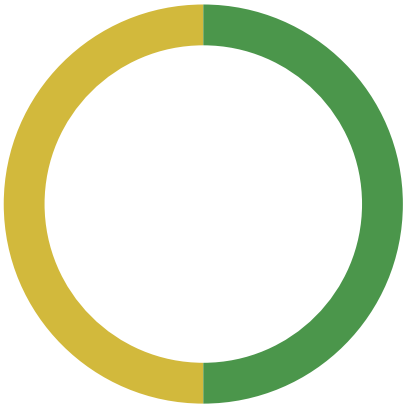
1.4. Results

During our assessment on the scoped Benqi Oracle contracts, we discovered two findings. No critical issues were found. One finding was of medium impact and one was of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Chaos Labs in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	1
Informational	0



2. Introduction

2.1. About Benqi Oracle

Chaos Labs contributed the following description of Benqi Oracle:

> Dual oracle system for the BENQI Finance protocol, combining Edge Oracle (primary) and Chainlink Oracle (backup).

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Benqi Oracle Contracts

Type	Solidity
Platform	EVM-compatible
Target	Benqi-DualOracle
Repository	https://github.com/clmikestone/Benqi-DualOracle ↗
Version	`78f10c0816a80c6d7cc1d299c31eba3185c04c00`
Programs	`BenqiDualOracle` `BenqiPriceOracle`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of three person-days. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Seunghyeon Kim
Engineer
seunghyeon@zellic.io

Chongyu Lv
Engineer
chongyu@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

May 8, 2025	Kick-off call
May 8, 2025	Start of primary review period
May 9, 2025	End of primary review period

3. Detailed Findings

3.1. Missing staleness check in `BenqiDualOracle::getUnderlyingPrice` when `oracleMode` is `EDGE_ONLY` or `CL_ONLY`

Target	BenqiDualOracle		
Category	Business Logic	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

BenqiDualOracle implements a dual oracle system with Edge as primary and Chainlink as backup. It has three different modes:

```
/// @notice Enum representing different oracle modes
enum OracleMode {
    EDGE_ONLY, // Use only Edge oracle
    CL_ONLY, // Use only Chainlink oracle
    DUAL // Use both oracles (default)
}
```

In the comments of the BenqiDualOracle contract, it is mentioned that staleness detection is one of the responsibilities of this contract:

```
* @dev Dual oracle system with mechanisms for:
*      - Staleness detection
```

However, in the `BenqiDualOracle::getUnderlyingPrice` implementation, when the `oracleMode` is `EDGE_ONLY` or `CL_ONLY`, no staleness detection is performed:

```
// If in Edge-only mode, return Edge price without checks
if (oracleMode == OracleMode.EDGE_ONLY) {
    (uint256 edgePriceResult,)
    = edgeOracle.getUnderlyingPriceWithTimestamp(qiToken);
    if (edgePriceResult == 0) revert EdgeOracleZeroPrice();
    return edgePriceResult;
}

// If in Chainlink-only mode, return Chainlink price without checks
if (oracleMode == OracleMode.CL_ONLY) {
    (uint256 chainlinkPriceResult,)
    = chainlinkOracle.getUnderlyingPriceWithTimestamp(qiToken);
```

```
    if (chainlinkPriceResult == 0) revert ChainlinkOracleZeroPrice();
    return chainlinkPriceResult;
}
```

For example, when `oracleMode == OracleMode.EDGE_ONLY`, the `BenqiPriceOracle::getUnderlyingPriceWithTimestamp` function will be called to obtain price information:

```
function getUnderlyingPriceWithTimestamp(IQiToken qiToken)
    external view override returns (uint256, uint256) {
    address qiTokenAddress = address(qiToken);
    IAggregatorV2V3Interface feed;

    if (qiTokenAddress == qiAvaxAddress) {
        feed = getFeed(AVAX_UNDERLYING);
    } else {
        address underlying = address(IQiErc20(address(qiToken)).underlying());
        feed = getFeed(underlying);
    }

    return getPriceAndTimestamp(feed, qiToken);
}
```

Finally, the `BenqiPriceOracle::getPriceAndTimestamp` function is called. In the `BenqiEdgeOracle::getPriceAndTimestamp` function, although `updatedAt` is obtained, no stale price check is performed based on it:

```
function getPriceAndTimestamp(IAggregatorV2V3Interface feed, IQiToken qiToken)
    internal
    view
    returns (uint256, uint256)
{
    address tokenAddress =
        address(qiToken) == qiAvaxAddress ? AVAX_UNDERLYING :
    address(IQiErc20(address(qiToken)).underlying());
    IEIP20Interface token = IEIP20Interface(tokenAddress);

    // Check for manually set price first
    if (prices[tokenAddress] != 0) {
        uint256 manualPrice = prices[tokenAddress];
        // Only adjust for decimals if it's not AVAX
        if (address(qiToken) != qiAvaxAddress) {
            uint256 tokenDecimalDelta = 18 - token.decimals();
            if (tokenDecimalDelta > 0) {
                manualPrice = manualPrice * (10 ** tokenDecimalDelta);
```

```
            }
        }
        return (manualPrice, block.timestamp);
    }

    // If no manual price, proceed with feed data
    (, int256 answer,, uint256 updatedAt,) = feed.latestRoundData();

    require(answer > 0, "negative or zero price not allowed");

    uint256 feedPrice = uint256(answer);

    // If it's not AVAX, adjust for token decimals
    if (address(qiToken) != qiAvaxAddress) {
        uint256 tokenDecimalDelta = 18 - token.decimals();
        if (tokenDecimalDelta > 0) {
            feedPrice = feedPrice * (10 ** tokenDecimalDelta);
        }
    }

    // Convert to 18 decimals if needed
    uint256 feedDecimalDelta = 18 - feed.decimals();
    if (feedDecimalDelta > 0) {
        feedPrice = feedPrice * (10 ** feedDecimalDelta);
    }

    return (feedPrice, updatedAt);
}
```

Impact

When `oracleMode` is `EDGE_ONLY` or `CL_ONLY`, code will execute with prices that do not reflect the current pricing, resulting in a potential loss of funds for users.

Recommendations

Consider adding a staleness check when `OracleMode` is `EDGE_ONLY` or `CL_ONLY`.

Remediation

This issue has been acknowledged by Chaos Labs, and a fix was implemented in commit `aa97b6fd` ↗.

3.2. Feed price validation can be bypassed

Target	BenqiDualOracle		
Category	Business Logic	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

In the BenqiDualOracle contract, the `validateFeedPriceDeviation` function is designed to validate a manually set price against the prices from the Edge and Chainlink oracles. However, if both the `edgeFeed` and `chainlinkFeed` are set to the zero address, the function effectively bypasses the validation process. This occurs because the function only performs validation checks if the feed addresses are nonzero. As a result, the manual price can be set without any comparison to oracle prices, potentially leading to inaccurate or manipulated price settings.

```
function validateFeedPriceDeviation(address asset, uint256 manualPrice)
    internal view {
    // Check Edge feed if available
    IAggregatorV2V3Interface edgeFeed = edgeOracle.getFeed(asset);
    if (address(edgeFeed) != address(0)) {
        uint256 feedPrice = getFeedPrice(edgeFeed);
        if (feedPrice > 0) {
            // Check for significant deviation (10x or 0.1x difference)
            if (manualPrice > feedPrice * 10 || feedPrice > manualPrice * 10) {
                revert ManualPriceDeviationError(asset, manualPrice,
    feedPrice);
            }
        }
    }

    // Check Chainlink feed if available
    IAggregatorV2V3Interface chainlinkFeed = chainlinkOracle.getFeed(asset);
    if (address(chainlinkFeed) != address(0)) {
        uint256 feedPrice = getFeedPrice(chainlinkFeed);
        if (feedPrice > 0) {
            // Check for significant deviation (10x or 0.1x difference)
            if (manualPrice > feedPrice * 10 || feedPrice > manualPrice * 10) {
                revert ManualPriceDeviationError(asset, manualPrice,
    feedPrice);
            }
        }
    }
```

```
}
```

Impact

The bypass of feed price validation could allow for manual prices to be set without any checks against oracle-provided prices. While the likelihood of both feeds being set to zero address is low, if it occurs, it could lead to incorrect price data being used within the system. This could affect the reliability and accuracy of the oracle system, although the overall impact is considered low due to the unlikelihood of this scenario.

Recommendations

To prevent the bypass of feed price validation, it is recommended to implement additional checks within the `validateFeedPriceDeviation` function. Specifically, the function should include logic to revert or throw an error if both `edgeFeed` and `chainlinkFeed` are set to the zero address. This will ensure that manual prices are always validated against at least one oracle price, maintaining the integrity and reliability of the oracle system.

Remediation

This issue has been acknowledged by Chaos Labs, and a fix was implemented in commit `967b0e3a` ↗.

Chaos Labs introduced a `manualOverrideAllowed` argument to several functions in the contract. This change is primarily related to the price configuration of BUSD. In order to maintain BUSD's price at a fixed value of $1.00 USD, it is necessary to set the price manually. Therefore, when the `manualOverrideAllowed` flag is explicitly set to `true`, the system is designed to permit price updates even if both oracle feeds are set to the zero address. This exception prevents the transaction from reverting in this specific scenario, ensuring compatibility with the intended manual price-setting mechanism.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. The `transferOracleAdmins` function could always be reverted

The `transferOracleAdmins` function in BenqiDualOracle allows the contract owner to assign a new admin address for both `edgeOracle` and `chainlinkOracle` by invoking their respective `setAdmin` methods. However, since each `setAdmin` function is protected by an `onlyAdmin` modifier and the admin state variable is a single fixed address per oracle contract, this call will revert unless the current caller is already the admin of each oracle.

```
function transferOracleAdmins(address newAdmin) external onlyOwner {
    if (newAdmin == address(0)) revert InvalidAddress();

    // Transfer Edge Oracle admin
    edgeOracle.setAdmin(newAdmin);

    // Transfer Chainlink Oracle admin
    chainlinkOracle.setAdmin(newAdmin);

    emit OracleAdminTransferred(newAdmin);
}
```

The Chaos Labs team clarified that the oracles are initially configured with the contract owner as the admin, so the function works without issues at deployment time. They also noted that this functionality is intended to support future deployments of new contracts, where transferring oracle admin rights may be necessary.

4.2. The internal function `BenqiPriceOracle::getFeedPrice` is not used

The `BenqiPriceOracle::getFeedPrice` function is internal, but no other public or external functions can call it.

```
function getFeedPrice(IAggregatorV2V3Interface feed)
    internal view returns (uint256) {
    // Price feeds store answers at their defined decimal places
    uint256 decimalDelta = 18 - feed.decimals();
    // Ensure that we don't multiply the result by 0
    if (decimalDelta > 0) {
```

```
            return uint256(feed.latestAnswer()) * (10 ** decimalDelta);
        } else {
            return uint256(feed.latestAnswer());
        }
    }
```

And this function uses the deprecated Chainlink's `latestAnswer` function. Consider deleting this function.

This issue has been acknowledged by Chaos Labs, and a fix was implemented in commit `b1fa9adf` ↗.

4.3. Incorrect revert error message

This is in the `BenqiDualOracle::setDeviationThreshold` function.

```
if (newMaxThreshold == 0) revert InvalidAddress();
```

When the function's `uint256` type parameter `newThreshold` is 0, the function will `revert InvalidAddress()`, which is inconsistent with the code semantics because `newThreshold` is not of address type. Consider adding an `InvalidnewThreshold` error, and when `newThreshold` is 0, `revert InvalidnewThreshold()`.

This issue has been acknowledged by Chaos Labs, and a fix was implemented in commit `475e51c0` ↗.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: BenqiDualOracle.sol

Function: `calculateDeviation(uint256 price1, uint256 price2)`

This function calculates the percentage deviation between two prices.

Inputs

- `price1`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to calculate the deviation of.
- `price2`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to calculate the deviation of.

Branches and code coverage

Intended branches

- Check if `price1` is zero.
  - ☑ Test coverage
- Check if `price2` is zero.
  - ☑ Test coverage
- Calculate the absolute difference between the two prices.
  - ☑ Test coverage
- Use the larger price as a denominator for a more conservative estimate.
  - ☑ Test coverage
- Return the deviation as a percentage in 18 decimals.
  - ☑ Test coverage

Function: `getFeedPrice(IAggregatorV2V3Interface feed)`

This function safely gets a normalized price from a feed.

Inputs

- `feed`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price feed to get the price of.

Branches and code coverage

Intended branches

- Check if the feed is available.
  - ☑ Test coverage
- Check if the feed price is greater than zero.
  - ☑ Test coverage
- Normalize the price.
  - ☑ Test coverage

Function: `getOraclePriceWithFreshness(IQiToken qiToken, bool isEdge, uint256 stalenessThreshold)`

This function gets a price from an oracle and checks if it is fresh.

Inputs

- `qiToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A `qiToken` to get the price of.
- `isEdge`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A boolean to check if the price should be retrieved from the Edge oracle.

- `stalenessThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A staleness threshold to check if the price is stale.

Branches and code coverage

Intended branches

- Check if the price is stale.
  - ☑ Test coverage
- Get the price from the Edge oracle.
  - ☑ Test coverage
- Get the price from the Chainlink oracle.
  - ☑ Test coverage
- Return the price and whether it is stale.
  - ☑ Test coverage

Negative behavior

- Revert if the price is zero.
  - ☑ Negative test

Function: `getUnderlyingPrice(IQiToken qiToken)`

This function gets the price of an underlying asset.

Inputs

- `qiToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A `qiToken` to get the price of.

Branches and code coverage

Intended branches

- Check if the asset is configured.

☑ Test coverage

- Check if the oracle mode is Edge only.

☑ Test coverage

- Check if the oracle mode is Chainlink only.

☑ Test coverage

- Get the price of the asset from the Edge oracle.

☑ Test coverage

- Get the price of the asset from the Chainlink oracle.

☑ Test coverage

- Return the price of the asset.

☑ Test coverage

Negative behavior

- Revert if the asset is not configured.

☑ Negative test

- Revert if the both oracles are stale.

☑ Negative test

- Revert if the price deviation is too high.

☑ Negative test

Function: `isPriceStale(uint256 timestamp, uint256 stalenessThreshold)`

This function checks if a price is stale based on time stamp.

Inputs

- `timestamp`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A time stamp to check if the price is stale.
- `stalenessThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A staleness threshold to check if the price is stale.

Branches and code coverage

Intended branches

- Check if the price is stale.
  - ☑ Test coverage

Function: `normalizePrice(uint256 price, uint8 currentDecimals)`

This function normalizes the price to 18 decimals.

Inputs

- `price`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to normalize.
- `currentDecimals`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The current number of decimals of the price.

Branches and code coverage

Intended branches

- Check if the current number of decimals is 18.
  - ☑ Test coverage
- Check if the current number of decimals is less than 18.
  - ☑ Test coverage
- Check if the current number of decimals is greater than 18.
  - ☑ Test coverage

Function: `setAssetOracles(address asset, address edgeFeed, address chainlinkFeed, uint256 stalenessThreshold, uint256 customDeviationThreshold)`

This function sets the oracles for an asset.

Inputs

- `asset`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An asset to set the oracles of.
- `edgeFeed`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An Edge feed to set the Edge oracle of.
- `chainlinkFeed`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A Chainlink feed to set the Chainlink oracle of.
- `stalenessThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A threshold to set the staleness of.
- `customDeviationThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A custom deviation threshold to set the deviation threshold of.

Branches and code coverage

Intended branches

- Try to get the price of the asset from the Edge oracle.
  - ☑ Test coverage
- Try to get the price of the asset from the Chainlink oracle.
  - ☑ Test coverage
- Check if the price is stale.
  - ☑ Test coverage
- Calculate the deviation between the Edge and Chainlink prices.
  - ☑ Test coverage
- Set the oracles for the asset.
  - ☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.
  - ☑ Negative test
- Revert if the asset is not a zero address.
  - ☑ Negative test
- Revert if the Edge feed is not a zero address.
  - ☑ Negative test
- Revert if the Chainlink feed is not a zero address.
  - ☑ Negative test
- Revert if the staleness threshold is zero.
  - ☑ Negative test
- Revert if the custom deviation threshold is greater than the max deviation threshold.
  - ☑ Negative test

Function: `setDeviationThreshold(uint256 newThreshold)`

This function sets the global deviation threshold.

Inputs

- `newThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A new deviation threshold to set the deviation threshold of.

Branches and code coverage

Intended branches

- Set the deviation threshold.
  - ☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.
  - ☑ Negative test
- Revert if the deviation threshold is zero.

- ☑ Negative test
- Revert if the deviation threshold is greater than the max deviation threshold.
  - ☑ Negative test

Function: `setDirectPrice(address asset, uint256 price, bool setInEdge, bool setInChainlink)`

This function sets the direct price for an asset in both underlying oracles.

Inputs

- `asset`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An asset to set the direct price of.
- `price`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to set the direct price of.
- `setInEdge`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A boolean to set if the price should be set in the Edge oracle.
- `setInChainlink`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A boolean to set if the price should be set in the Chainlink oracle.

Branches and code coverage

Intended branches

- Check if the asset is a zero address.
  - ☑ Test coverage
- Check if the price is zero.
  - ☑ Test coverage
- Validate the price.

☑ Test coverage

- Set the price in the Edge oracle.

☑ Test coverage

- Set the price in the Chainlink oracle.

☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.

☑ Negative test

- Revert if both `setInEdge` and `setInChainlink` are false.

☑ Negative test

- Revert if the asset is a zero address.

☑ Negative test

- Revert if the price is zero.

☑ Negative test

Function: `setMaxDeviationThreshold(uint256 newMaxThreshold)`

This function sets the maximum-allowed deviation threshold.

Inputs

- `newMaxThreshold`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A new maximum deviation threshold to set the maximum deviation threshold of.

Branches and code coverage

Intended branches

- Set the maximum deviation threshold.

☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.

☑ Negative test

- Revert if the new maximum deviation threshold is zero.

☑ Negative test

- Revert if the new maximum deviation threshold is greater than 50e16.

☑ Negative test

Function: `setOracleMode(OracleMode _mode)`

This function sets the oracle mode.

Inputs

- `_mode`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An oracle mode to set the mode of.

Branches and code coverage

Intended branches

- Set the oracle mode.

☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.

☑ Negative test

- Revert if the mode is not a valid mode.

☐ Negative test

Function: `setUnderlyingPrice(IQiToken qiToken, uint256 underlyingPriceMantissa, bool setInEdge, bool setInChainlink)`

This function sets the underlying price for a `qiToken` in both underlying oracles.

Inputs

- `qiToken`

    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: A `qiToken` to set the underlying price of.
- `underlyingPriceMantissa`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: A price to set the underlying price of.
- `setInEdge`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: A boolean to set if the price should be set in the Edge oracle.
- `setInChainlink`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: A boolean to set if the price should be set in the Chainlink oracle.

Branches and code coverage

Intended branches

- Check if the `qiToken` is a zero address.
    - ☑ Test coverage
- Check if both `setInEdge` and `setInChainlink` are false.
    - ☑ Test coverage
- Check if the price is zero.
    - ☑ Test coverage
- Validate the price.
    - ☑ Test coverage
- Set the price in the Edge oracle.
    - ☑ Test coverage
- Set the price in the Chainlink oracle.
    - ☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.
    - ☑ Negative test
- Revert if the `qiToken` is a zero address.

☑ Negative test

- Revert if both `setInEdge` and `setInChainlink` are false.

☑ Negative test

- Revert if the price is zero.

☑ Negative test

Function: `transferOracleAdmins(address newAdmin)`

This function transfers the admin rights of both oracle contracts to a new address.

Inputs

- `newAdmin`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A new admin to transfer the admin rights to.

Branches and code coverage

Intended branches

- Transfer the admin rights of the Edge oracle to the new admin.

☑ Test coverage

- Transfer the admin rights of the Chainlink oracle to the new admin.

☑ Test coverage

Negative behavior

- Revert if the caller is not an owner.

☑ Negative test

- Revert if the new admin is a zero address.

☑ Negative test

Function: `validateFeedPriceDeviation(address asset, uint256 manualPrice)`

This function validates a manual price against oracle feed prices to prevent large deviations.

Inputs

- `asset`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An asset to validate the feed price of.
- `manualPrice`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A manual price to validate the feed price of.

Branches and code coverage

Intended branches

- Check if the Edge feed is available.
  - ☑ Test coverage
- Check if the Chainlink feed is available.
  - ☑ Test coverage
- Check if the manual price is greater than the Edge feed price times 10.
  - ☑ Test coverage
- Check if the manual price is greater than the Chainlink feed price times 10.
  - ☑ Test coverage

Negative behavior

- Revert if the Edge feed is not available.
  - ☑ Negative test
- Revert if the Chainlink feed is not available.
  - ☑ Negative test
- Revert if the manual price is greater than the Edge feed price times 10.
  - ☑ Negative test
- Revert if the manual price is greater than the Chainlink feed price times 10.
  - ☑ Negative test

5.2. Module: BenqiPriceOracle.sol

Function: `assetPrices(address asset)`

This function is used to get the price of an asset.

Inputs

- `asset`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An asset to get the price of.

Branches and code coverage

Intended branches

- Get the price of the asset.
  - ☑ Test coverage

Function: `getFeedPrice(IAggregatorV2V3Interface feed)`

This function is used to get the price of a feed.

Inputs

- `feed`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A feed to get the price of.

Branches and code coverage

Intended branches

- Get the price of the feed.
  - ☑ Test coverage

Function: `getFeed(address token)`

This function is used to get the feed of an asset.

Inputs

- `token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A token to get the feed of.

Branches and code coverage

Intended branches

- Get the feed of the token.
  - ☑ Test coverage

Function: `getPriceAndTimestamp(IAggregatorV2V3Interface feed, IQiToken qiToken)`

This function is used to get the price and time stamp of a `qiToken`.

Inputs

- `feed`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A feed to get the price and time stamp of.
- `qiToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A token to get the price and time stamp of.

Branches and code coverage

Intended branches

- Check if the given `qiToken` is the `qiAvax` token.

☑ Test coverage

- Get the price of the `qiToken` from the feed.

☑ Test coverage

Negative behavior

- Revert if the answer from the `lastestRoundData` is not greater than zero.

☑ Negative test

Function: `getUnderlyingPriceWithTimestamp(IQiToken qiToken)`

This function is used to get the underlying price of a `qiToken` and the time stamp of the price.

Inputs

- `qiToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A token to get the underlying price of.

Branches and code coverage

Intended branches

- Check if the given `qiToken` is the `qiAvax` token.

☑ Test coverage

- Check if the given `qiToken` is not the `qiAvax` token.

☑ Test coverage

- Get the price of the `qiToken` from the feed.

☑ Test coverage

Function: `getUnderlyingPrice(IQiToken qiToken)`

This function is used to get the underlying price of a `qiToken`.

Inputs

- `qiToken`

- **Control**: Fully controlled by the caller.
- **Constraints**: None.
- **Impact**: A token to get the underlying price of.

Branches and code coverage

Intended branches

- Check if the given `qiToken` is the `qiAvax` token.
  - ☑ Test coverage
- Check if the given `qiToken` is not the `qiAvax` token.
  - ☑ Test coverage
- Get the price of the `qiToken` from the feed.
  - ☑ Test coverage

Function: `setAdmin(address newAdmin)`

This function is used to set the admin of the contract.

Inputs

- `newAdmin`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A new admin to set the admin to.

Branches and code coverage

Intended branches

- Set the admin of the contract.
  - ☑ Test coverage

Negative behavior

- Revert if the caller is not a service admin.
  - ☑ Negative test

Function: `setDirectPrice(address asset, uint256 price)`

This function is used to set the price of an asset.

Inputs

- `asset`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: An asset to set the price of.
- `price`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to set the price to.

Branches and code coverage

Intended branches

- Set the price of the asset.
  - ☑ Test coverage

Negative behavior

- Revert if the caller is not a service admin.
  - ☑ Negative test

Function: `setFeed(address token, address feed)`

This function is used to set the price of an asset.

Inputs

- `token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A token to set the feed of.
- `feed`
  - **Control**: Fully controlled by the caller.

- **Constraints**: None.
- **Impact**: A feed to set the feed to.

Branches and code coverage

Intended branches

- Set the feed of the token.
  - ☑ Test coverage

Negative behavior

- Revert if the caller is not a service admin.
  - ☑ Negative test
- Revert if the feed is the address of the contract or the zero address.
  - ☑ Negative test

Function: `setUnderlyingPrice(IQiToken qiToken, uint256 underlyingPriceMantissa)`

This function is used to set the underlying price of a `qiToken`.

Inputs

- `qiToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A token to set the underlying price of.
- `underlyingPriceMantissa`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: A price to set the underlying price to.

Branches and code coverage

Intended branches

- Check if the given `qiToken` is the `qiAvax` token.
  - ☑ Test coverage

- Set the price of the `qiToken`.

  ☑ Test coverage

Negative behavior

- Revert if the caller is not a service admin.

  ☑ Negative test

6. Assessment Results

During our assessment on the scoped Benqi Oracle contracts, we discovered two findings. No critical issues were found. One finding was of medium impact and one was of low impact.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.