CoW AMM LP Oracle Audit

Gnosis Ltd - Report by Côme du Crest

2025-04-14

Table of contents

CoW AMM LP Oracle Audit

This document presents the findings of a smart contract audit conducted by Côme du Crest for Gnosis Ltd.

Scope

The scope includes all contracts within cowdao-grants/cow-amm-lp-oracle as of commit `0xd9f2789` .

Context

The goal is to provide a fair price for the LP token based on the TVL of the underlying cow-amm pool. The formula for pricing the LP token is `TVL`/`LP_supply`. To avoid manipulation by unbalancing the pool, the LP token price must be computed when the pool is balanced. That is, the ratio of token balances matches the price ratio given by price oracles of the tokens. If we take the weights of the tokens in the weighted pool, the formula is:

$$Px * Bx * wy = Py * By * wx \tag{1}$$

Where `Px` is the price of token `x` given by the fair price oracle, `Bx` is the balance of `x` in the pool at equilibrium, `wx` is the weight of the token `x` in the pool.

The formula for TVL is:

$$TVL = Bx * Px + By * Py = Bx * Px * (1 + wy/wx) \tag{2}$$

The pool constant is:

$$k = Bx^{wx} * By^{wy} = Bx^{wx} * (Px/Py * Bx * wy/wx)^{wy} \tag{3}$$

Knowing this is the pool constant, it does not matter wether `Bx` is the balance of `x` when the pool is balanced or not. Computing the value of `k` can be done with the unbalanced pool's balances.

Re-injecting the formula for `k` to eliminate the unknown variable `Bx` we get:

$$TVL = k * Px^{wx} * Py^{wy} * ((wx/wy)^{wy} + (wy/wx)^{wx}) \tag{4}$$

Status

The report has been sent to the core developer.

Legal Information And Disclaimer

1. This report is based solely on the information provided by [Audited Company] (the "Company"), with the assumption that the information provided to Gnosis is authentic, accurate, complete, and not misleading as of the date of this report. Gnosis has not conducted any independent enquiries, investigations or due diligence in respect of the Company, its business or its operations.
2. Changes to the information contained in the documents, repositories and any other materials referenced in this report might affect or change the analysis and conclusions presented. Gnosis is not responsible for monitoring, nor will we be aware of, any future additions, modifications, or deletions to the audited code. As such, Gnosis does not assume any responsibility to update any information, content or data contained in this report following the date of its publication.
3. This report does not address, nor should it be interpreted as addressing, any regulatory, tax or legal matters, including but not limited to: tax treatment, tax consequences, levies, duties, data privacy, data protection laws, issues relating to the licensing of information technology, intellectual property, money laundering and countering the financing of terrorism, or any other legal restrictions or prohibitions. Gnosis disclaims any liability for omissions or errors in the findings or conclusions presented in this report.
4. The views expressed in this report are solely our views regarding the specific issues discussed within this report. This report is not intended to be exhaustive, nor should it be construed as an assurance, guarantee or warranty that the code is free from bugs, vulnerabilities, defects or deficiencies. Different use cases may carry different risks, and integration with third-party applications may introduce additional risks.
5. This report is provided for informational purposes only and should not be used as the basis for making investment or financial decisions. This report does not constitute investment research and should not be viewed as an invitation, recommendation, solicitation or offer to subscribe for or purchase any securities, investments, products or services. Gnosis is not a financial advisor, and this report does not constitute financial or investment advice.
6. The statements in this report should be considered as a whole, and no individual statement should be extracted or referenced independently.
7. To the fullest extent permitted by applicable laws, Gnosis disclaims any and all other liability, whether in contract, tort, or otherwise, that may arise from this report or the use thereof.

Issues

[Info] Oracle may become unresponsive when oracle feeds decimals are updated

Summary

The function to get the LP token price `latestRoundData()` reverts if the price feeds for the underlying pool tokens use more than 18 decimals. Knowing price feeds can be proxy contracts that can be updated, they could use 8 decimals at one point and 19 at another point, breaking the oracle. (see `ETH / USD price feed`)

Vulnerability Detail

Computing the LP token price starts by getting the price of the underlying tokens. It will scale the price to 18 decimals using `10 ** (18 - feedDecimals)` which will cause an underflow revert if the feed uses more than 18 decimals.

```
    function latestRoundData()
        external
        view
        returns (uint80 roundId, int256 answer, uint256 startedAt, uint256
            updatedAt, uint80 answeredInRound)
    {
        /* Get the price feed data */
        (int256 answer0, int256 answer1, uint256 updatedAt_) = _getFeedData();
        ...
    }

    function _getFeedData() internal view returns (int256 answer0, int256
        answer1, uint256 updatedAt) {
        /* Get latestRoundData from price feeds */
        (, int256 answer0_,, uint256 updatedAt0,) = FEED0.latestRoundData();
        (, int256 answer1_,, uint256 updatedAt1,) = FEED1.latestRoundData();

        /* Set update timestamp of oldest price feed */
        updatedAt = updatedAt0 < updatedAt1 ? updatedAt0 : updatedAt1;

        /* Adjust answers for price feed decimals */
        uint8 feed0Decimals = FEED0.decimals();
        uint8 feed1Decimals = FEED1.decimals();

        return (answer0_ * int256(10 ** (18 - feed0Decimals)), answer1_ *
            int256(10 ** (18 - feed1Decimals)), updatedAt);
    }
```

Impact

In the very unlikely event that the token price feeds are updated to use more than 18 decimals, the function `latestRoundData()` will revert which may break protocols relying on it.

Code Snippets

https://github.com/cowdao-grants/cow-amm-lp-oracle/blob/d9f27899c574f46133158e022c0534d98e096ee6/src/LPOracle.sol#L146

Recommendation

It is not too hard to scale the feed answer to 18 decimals if it uses more than 18 decimals. Something like:

```
scale = feedDecimals > 18 ? feedDecimals - 18 : 18 - feedDecimals;
answer = answer * 10 ** scale;
```

Otherwise, acknowledge the issue.

[Info] Tokens changing decimals will result in incorrect price

Summary

The decimals of a token is fetched and registered at deploy time by the oracle. If the decimals of a token is changed afterwards, the oracle will compute an incorrect price.

Vulnerability Detail

The constructor fetches and stores the tokens decimals:

```
    constructor(address _pool, address _feed0, address _feed1) {
        ...
        address[] memory tokens = IBCoWPool(POOL).getFinalTokens();
        TOKEN0 = IERC20(tokens[0]);
        TOKEN1 = IERC20(tokens[1]);

        /* Set token decimals */
        TOKEN0_DECIMALS = TOKEN0.decimals();
        TOKEN1_DECIMALS = TOKEN1.decimals();
        ...
    }
```

The decimals are later used when calculating the TVL:

```
    function _calculateTVL(int256 answer0, int256 answer1) internal view
        returns (uint256 tvl) {
        /* Get pool k value */
        int256 balance0 = int256(TOKEN0.balanceOf(POOL) * 10 ** (18 -
            TOKEN0_DECIMALS));
        int256 balance1 = int256(TOKEN1.balanceOf(POOL) * 10 ** (18 -
            TOKEN1_DECIMALS));
        ...
    }
```

Impact

In the extremely unlikely case where the decimals of a token would change, the oracle would compute a price using the stored incorrect decimals value for a token.

Code Snippets

https://github.com/cowdao-grants/cow-amm-lp-oracle/blob/d9f27899c574f46133158e022c0534d98e096ee6/src/LPOracle.sol#L70-L71

Recommendation

Acknowledge the issue.

Alternatively fetch the token decimals from the token contract when needed instead of storing them. This would currently enable problems with not handling tokens with more than 18 decimals that would need to be fixed.