Veridise. **Auditing Report**

**Hardening Blockchain Security with Formal Methods**

**FOR**

# Optimism Proxy and Cozy Multi-Oracle Price Feed

Veridise Inc.

July 3, 2023

► **Prepared For:**

Client

► **Prepared By:**

Benjamin Mariano
Nicholas Brown
Ian Neal

► **Contact Us:** contact@veridise.com

► **Version History:**

| | |
|---|---|
| June 28, 2023 | V1 |
| June 30, 2023 | V2 |
| July 3, 2023 | V3 |

# Contents



 

# 1 Executive Summary

From June 26, 2023 to June 28, 2023, Client engaged Veridise to review the security of the Optimism Proxy and Cozy Multi-Oracle Price Feed, which consist of two different contracts: a multi-oracle price feed for Cozy protection tokens (PTokens) and an Optimism L1 proxy contract used to hold assets and execute transactions on Optimism on behalf of L1 addresses. Veridise conducted the assessment over 3 person-weeks, with 3 engineers reviewing code over 3 days on commit 6bf780f and 7218c79. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

**Code assessment.** The Client developers provided the source code of the Optimism Proxy and Cozy Multi-Oracle Price Feed contracts for review. The code includes a number of tests that were useful for auditors to better understand the code. The code was also well documented with READMEs as well as in-code comments.

**Summary of issues detected.** The audit uncovered 6 total issues. Issues found include unclear documentation of behavior (V-CLI-VUL-001), potentially restrictive size limits on returns from proxy transactions (V-CLI-VUL-002), and an uninitialized variable that could lead to minor disruptions in service for the Cozy multi-oracle price feed (V-CLI-VUL-003).

**Disclaimer.** We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

# 2 Project Dashboard

**Table 2.1:** Application Summary.

| Name | Version | Type | Platform |
|---|---|---|---|
| Optimism Proxy | 6bf780f | Solidity | Optimism |
| Cozy Multi-Oracle Price Feed | 7218c79 | Solidity | Optimism |

**Table 2.2:** Engagement Summary.

| Dates | Method | Consultants Engaged | Level of Effort |
|---|---|---|---|
| June 26 - June 28, 2023 | Manual & Tools | 3 | 3 person-weeks |

**Table 2.3:** Vulnerability Summary.

| Name | Number | Resolved |
|---|---|---|
| Critical-Severity Issues | 0 | 0 |
| High-Severity Issues | 0 | 0 |
| Medium-Severity Issues | 0 | 0 |
| Low-Severity Issues | 0 | 0 |
| Warning-Severity Issues | 1 | 1 |
| Informational-Severity Issues | 5 | 5 |
| TOTAL | 6 | 6 |

**Table 2.4:** Category Breakdown.

| Name | Number |
|---|---|
| Usability Issue | 5 |
| Maintainability | 1 |

# 3 Audit Goals and Scope

## 3.1 Audit Goals

The engagement was scoped to provide a security assessment of Client's smart contracts.

In our audit, we sought to answer the following questions:

- Can a malicious user steal funds from the proxy contract?
- Can a malicious user execute transactions illegally through the proxy contract?
- Are all relevant access controls maintained for the proxy contract?
- Is the price calculated as expected for the Cozy multi-oracle price feed?
- Is the Cozy multi-oracle price feed vulnerable to flashloan attacks?
- Can the Cozy multi-oracle price feed be compromised with denial of service attacks?

## 3.2 Audit Methodology & Scope

**Audit Methodology.** To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following technique:

- *Static Analysis.* We leverage static analysis to identify common vulnerabilities in smart contracts using our tool Vanguard.

*Scope*. The scope of this audit is limited to the `OptimismL1Proxy.sol` and `CozyMultiOraclePriceFeed.sol` contracts provided by the Client developers (as well as associated interfaces), which contain the smart contract implementation of the Optimism Proxy and Cozy Multi-Oracle Price Feed.

*Methodology*. Veridise auditors inspected provided tests, and read the Optimism Proxy and Cozy Multi-Oracle Price Feed documentation. They then began a manual audit of the code assisted by tooling.

## 3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

**Table 3.1:** Severity Breakdown.

| | Somewhat Bad | Bad | Very Bad | Protocol Breaking |
|---|---|---|---|---|
| Not Likely | Info | Warning | Low | Medium |
| Likely | Warning | Low | Medium | High |
| Very Likely | Low | Medium | High | Critical |

**Table 3.2:** Likelihood Breakdown

| | |
|---|---|
| Not Likely | A small set of users must make a specific mistake |
| Likely | Requires a complex series of steps by almost any user(s)<br>- OR -<br>Requires a small set of users to perform an action |
| Very Likely | Can be easily performed by almost anyone |

**Table 3.3:** Impact Breakdown

| | |
|---|---|
| Somewhat Bad | Inconveniences a small number of users and can be fixed by the user |
| Bad | Affects a large number of people and can be fixed by the user<br>- OR -<br>Affects a very small number of people and requires aid to fix |
| Very Bad | Affects a large number of people and requires aid to fix<br>- OR -<br>Disrupts the intended behavior of the protocol for a small group of users through no fault of their own |
| Protocol Breaking | Disrupts the intended behavior of the protocol for a large group of users through no fault of their own |

# 4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

**Table 4.1:** Summary of Discovered Vulnerabilities.

| ID | Description | Severity | Status |
|---|---|---|---|
| V-CLI-VUL-001 | Unclear documentation of proxy functions | Warning | Fixed |
| V-CLI-VUL-002 | Small MAX_COPY value for excessivelySafeCall | Info | Fixed |
| V-CLI-VUL-003 | bidPriceWad not initialized by constructor | Info | Fixed |
| V-CLI-VUL-004 | Immutable max staleness | Info | Acknowledged |
| V-CLI-VUL-005 | Immutable oracles | Info | Acknowledged |
| V-CLI-VUL-006 | Magic number used instead of constant | Info | Fixed |

## 4.1 Detailed Description of Issues

### 4.1.1 V-CLI-VUL-001: Unclear documentation of proxy functions

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | 6bf780f |
| **Type** | Usability Issue | **Status** | Fixed |
| **File(s)** | `OptimismL1Proxy.sol` | | |
| **Location(s)** | executeFunction(), executeTransferEth() | | |

The documentation of `executeFunction()` and `executeTransferEth()` describes that the proxy must have ETH greater than or equal to the `msgValue_` (or `value_` for `executeTransferEth()`) argument of the functions, or the transaction will revert. However, `excessivelySafeCall()` will not cause the function to revert when the `msgValue_` is greater than the amount of ETH in the contract. Instead it will return a `success` value of `false` and the function will emit a `FunctionCallFailed` event.

**Impact** Users may be expecting the entire transaction to revert if `msgValue_` is too large, so the actual behavior may be unexpected.

**Recommendation** Update documentation for these functions to indicate that the function call/ETH transfer will fail and emit a `FunctionCallFailed` if `msgValue_` is too large.

**Developer Response** Behavior is as intended; the developers have updated documentation.

### 4.1.2 V-CLI-VUL-002: Small MAX_COPY value for excessivelySafeCall

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 6bf780f |
| **Type** | Usability Issue | **Status** | Fixed |
| **File(s)** | `OptimismL1Proxy.sol` | | |
| **Location(s)** | executeFunction() | | |

`MAX_COPY` is set to 150 bytes for all calls to `excessivelySafeCall()` . This may be too small for some legitimate calls to fully report their return value.

**Impact** As an example, a user might want to execute a function that returns a struct of 5 `uint256` values. This struct will be `32*5 = 160` bytes long, so it will not be fully copied by the call to `excessivelySafeCall()` . The `FunctionCallSuccess` event for this transaction will not include all of the desired data and there is no way for the user to get the full return value by calling `executeFunction()`.

**Recommendation** Increase `MAX_COPY` to a larger value so that a case like the one described above is unlikely to occur, or allow the user to optionally customize the value passed in as the `_maxCopy` argument of `excessivelySafeCall()` .

**Developer Response** Developers added a guarded setter for `MAX_COPY` in case they want to change this value in the future.

### 4.1.3 V-CLI-VUL-003: bidPriceWad not initialized by constructor

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 7218c79 |
| **Type** | Usability Issue | **Status** | Fixed |
| **File(s)** | `CozyMultiOraclePriceFeed.sol` | | |
| **Location(s)** | constructor(), price() | | |

`bidPriceWad` is not initialized by the constructor, but it is required to be non-zero when calling `price()` . The contract will not be able to provide prices until `setBidPriceWad()` is called.

**Impact** If the owner forgets to call `setBidPriceWad()` manually after deploying the contract, the price feed will not function.

**Recommendation** Initialize `bidPriceWad` in the constructor because it is a necessary step in the initialization of the contract. If this is not feasible, add clear warnings to the documentation that `setBidPriceWad()` must be called with a non-zero value in order for `price()` to be able to execute successfully.

**Developer Response** The developers will add this initialization to the constructor so the deployer can choose an initial value or intentionally initialize to 0.

### 4.1.4 V-CLI-VUL-004: Immutable max staleness

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 7218c79 |
| **Type** | Usability Issue | **Status** | Acknowledged |
| **File(s)** | `CozyMultiOraclePriceFeed.sol` | | |
| **Location(s)** | N/A | | |

The values of `staleAfterChainlinkA` and `staleAfterChainlinkB` can only be set upon contract initialization.

**Impact** If either `staleAfterChainlinkA` or `staleAfterChainlinkB` is set too low, could lead to service interruptions if average block time spikes temporarily or generally increases. Alternatively, if either of these times is too long, can result in inaccurate pricing. If these values need to be changed, the multi-oracle price feed needs to be redeployed and all contracts that depend on the multi-oracle price feed must also be changed to refer to the new deployment.

**Recommendation** Add `onlyOwner` setter functions for both `staleAfterChainlinkA` and `staleAfterChainlinkB`.

**Developer Response** This is based on an existing implementation where these variables are immutable. The developers will keep the implementation the same to optimize gas usage because these values are very unlikely to change. They also want to keep the implementation as close to the canon price feed implementation as possible.

### 4.1.5 V-CLI-VUL-005: Immutable oracles

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 7218c79 |
| **Type** | Usability Issue | **Status** | Acknowledged |
| **File(s)** | `CozyMultiOraclePriceFeed.sol` | | |
| **Location(s)** | N/A | | |

`chainlinkA` and `chainlinkB` are declared as immutable oracles.

**Impact** If one of these oracles stops working (i.e., always return stale data), this contract will always result in a reversion due to stale oracle data.

This also makes updating the oracles (e.g., when the oracles themselves are upgraded or a change in oracles is desired) more challenging, as the contract must be redeployed and all contracts that depend on this oracle must be changed as well.

**Recommendation** Add `onlyOwner` setter functions for both of these oracles.

**Developer Response** This is based on an existing implementation where these variables are immutable. The developers will keep the implementation the same to optimize gas usage because these values are very unlikely to change. They also want to keep the implementation as close to the canon price feed implementation as possible.

### 4.1.6 V-CLI-VUL-006: Magic number used instead of constant

| Severity | Info | Commit | 7218c79 |
|---|---|---|---|
| Type | Maintainability | Status | Fixed |
| File(s) | `CozyMultiOraclePriceFeed.sol` | | |
| Location(s) | price() | | |

The `CozyMultiOraclePriceFeed` defines a `WAD_DECIMALS` constant, but in the WAD conversion in the `price()` function, `1e18` is used directly instead of a computation based on the `WAD_DECIMALS`.

```
// First, determine the price of A in B with respect to the bid price, where chainlinkA is A/C, chainlinkB
// is B/C, and the Set has B underlying asset.
// e.g. The price of 1 ETH in PToken protection value with USDC underlying using ETH/USD and USDC/USD oracles:
// Protection in USDC = (1 ETH price in USD / 1e6 USDC price in USD) * (1e18 / bidPriceWAD) * USDC decimals
//                    = (1700e8 / 0.99e8) * (1e18 / 0.02e18) * 1e6
//                    = 85858.585858e6
// Note: The Set shares the same amount of decimals as the underlying asset
uint256 protectionValueInB_ = aPriceWAD_ * **1e18** * 10**decimalsSet / (bPriceWAD_ * bidPriceWAD);
```

**Impact** If the code is ever refactored or repurposed for a different unit other than a WAD, it could be possible to miss this computation and cause the newly modified code to have a computational error.

**Recommendation** Replace `1e18` with an expression based on `WAD_DECIMALS`, i.e. `10**WAD_DECIMALS` .

**Developer Response** The developers have updated this to use `WAD_DECIMALS` .