PUBLIC

Code Assessment

of the Gearbox V3 Oracles Smart Contracts

August 07, 2024

Produced for

Gearbox

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Gearbox Team,

Thank you for trusting us to help Gearbox Protocol with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Gearbox V3 Oracles according to Scope to support you in forming an opinion on their security risks.

Gearbox Protocol extends and refactors the oracle functionality used by the Gearbox Core V3.

The most critical subjects covered in our audit are the functional correctness of the contracts, the oracle configuration, and the interaction with the rest of the Gearbox system. No severe issues were uncovered. All the issues reported have been addressed. Security regarding all the aforementioned subjects is high.

The general subjects covered are access control, documentation and specification, gas efficiency, and the complexity of the implementation. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1
• Specification Changed	1
Low-Severity Findings	2
• Code Corrected	2

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the `contracts/` folder of the Gearbox V3 Oracles repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	25 Sept 2023	a5c8d378d8103d41f9b607e6a49d06b79a85847f	Initial Version
2	6 Nov 2023	9bf38997362424b4834f453c96dc7e1cb2ea6d8f	Version with fixes
3	7 Dec 2023	c6e4bd0a42331daeec599f3d8a688fab79f9879a	Minor improvements
4	7 Aug 2024	9fa370b78bf3880f02f3c410e52abe21acc345da	MellowLRT oracle

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

The following contracts are in the scope of the review:

```
interfaces:
    aave:
        IWAToken.sol
    balancer:
        IBalancerStablePool.sol
        IBalancerVault.sol
        IBalancerWeightedPool.sol
    compound:
        ICToken.sol
    curve:
        ICurvePool.sol
        IstETHPoolGateway.sol
    lido:
        IwstETH.sol
    yearn:
        IYVault.sol
    ILPPriceFeed.sol
oracles:
    aave:
        WrappedAaveV2PriceFeed.sol
    balancer:
        BPTStablePriceFeed.sol
        BPTWeightedPriceFeed.sol
    compound:
        CompoundV2PriceFeed.sol
    curve:
        CurveCryptoLPPriceFeed.sol
        CurveStableLPPriceFeed.sol
        CurveUSDPriceFeed.sol
    erc4626:
        ERC4626PriceFeed.sol
    lido:
        WstETHPriceFeed.sol
    updatable:
        RedstonePriceFeed.sol
    yearn:
        YearnPriceFeed.sol
    BoundedPriceFeed.sol
    CompositePriceFeed.sol
    LPPriceFeed.sol
    PriceFeedParams.sol
    SingleAssetLPPriceFeed.sol
    ZeroPriceFeed.sol
```

After (Version 4), the scope has been updated as follows:

- Added:

```
interfaces:
    mellow:
        IMellowChainlinkOracle.sol
        IMellowVault.sol
        IMellowVaultConfigurator.sol
oracles:
    mellow:
        MellowLRTPriceFeed.sol
```

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review, especially `FixedPoint` and `LogExpMath` are assumed to work correctly. Third-party libraries are out of the scope of this review. In particular, OpenZeppelin libraries and the contracts inherited by the Redstone price oracle are considered to function as expected. Updates in the core protocol not covered by the core V3 audit report are out of the scope of this review and the interactions with `integrations-v3` are assumed to work as expected.

2.2 System Overview

This system overview describes the initially received version ((Version 1)) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Gearbox Protocol offers a set of price oracles for the integrated protocols that will be used by version 3 of the Gearbox protocol.

Price feeds return the price of an asset in USD with 8 decimals in a format similar to Chainlink's via `latestRoundData()`. We could split the assets supported and their respective price feeds roughly into two categories: Primitives whose price becomes available by an off-chain price provider (`RedStonePriceFeed`) and derivatives whose price depends on some other, underlying, assets with some extra logic and data applied on them. An example of a primitive asset is ETH for which a RedStone oracle exists. An example of a derivative asset is LP tokens such as yETH. The price of yETH depends on the price of ETH queried by the respective oracle and the conversion rate from yETH to ETH provided by the respective Yearn vault.

The price feeds based on `RedStonePriceFeed` are expected to be updated only when needed with data packages for a specific `dataFeedId` that maps to a token. The data packages are prepared and signed by RedStone's data providers. An update to the price of a RedStone feed is only accepted if the supplied data packages have been signed by a minimum number of distinct signers. Each RedStone oracle has only one data feed, and the signed data packages must have been signed with the same timestamp, which can be up to 10 minutes in the past and 1 minute in the future relative to `block.timestamp`.

An important subcategory of the price feeds are these which extend `LPPriceFeed` and provide the prices for some lp tokens. The tokens currently supported are the following:

- AaveV2 LP token, via Gearbox Protocol's wrapped `aToken`
- BalancerV2 LP token for stable and weighted pools
- CompoundV2 LP token
- Curve LP token for stable and crypto pools
- crvUSD
- tokenized vaults token (ERC4626)
- Lido's wrapped stETH
- Yearn's vaults shares

The function `LPPriceFeed.latestRoundData()` has integrated bounds for the returned price. If the reported price is below a lower bound, the function reverts. If the reported price is above an upper bound computed based on the lower bound (`upper = 102% * lower`), the price is capped to the upper bound and the function does not revert. The lower bound can be set directly by the `controller`, or permissionlessly where the lower bound will be `0.2%` under the current price reported by the reserve price feed if there is one. The user can refer to the report for Core-v3 to learn more about the reserve price feed.

Other oracles can be used on top of the price feeds listed above, or other Chainlink-compatible oracles:

- `BoundedPriceFeed`: can be used to set an upper bound to a price reported by another oracle
- `CompositePriceFeed`: can be used to compute the price of the target token in USD by first getting its price in a base token denomination, and then getting the USD price if the base token, i.e, `B / A * USD / B = USD / A`
- `ZeroPriceFeed`: always return a zero price and last update timestamp

Some of the oracles perform internal price sanitization and thus the consumer does not have to sanitize it again (`skipPriceCheck = true`). These oracles are:

- all the oracles based on `LPPriceFeed`
- `BoundedPriceFeed`
- `CompositePriceFeed`
- `ZeroPriceFeed`

On the other hand, consumers should sanitize the prices given by `RedStonePriceFeed`.

2.2.1 Trust Model

- Users: not trusted

- Controller: trusted to update the `LPPriceFeed.lowerBound` value when necessary and in a non-adversarial manner. In the case of a rapid crash of a token and either the permissionless bound update is disabled or the reserve oracle is too slow, the controller is trusted to update the lower bound quickly enough so that liquidations are not blocked.
- Configurator: trusted to set and unset the `LPPriceFeed.updateBoundsAllowed` flag in a non-adversarial manner
- Signers of RedStone data packages: trusted to provide and sign up-to-date prices regularly.

2.3 Changes in Version 2

- The lowerbound in `LPPriceFeed` has been updated to be `1%` instead of `0.2%` under the current price reported by the reserve price feed
- The reserve price feed in `LPPriceFeed.updateBounds()` is enforced to be different than the primary price feed
- A new function to forbid permissionless bounds updates `LPPriceFeed.forbidBoundsUpdate` has been added. The `controller` or the `configurator` can call this function.

2.4 Changes in Version 3

- The lower bound of exchange rate of LPTokens is now set in the constructor.
- For the weighted balancer pools `getLPExchangeRate()` the invariant is multiplied by the number of assets. For `getAggregatePrice()`, the weighted price is divided by the number of assets. Users should note that the price of the LPTokens does not correspond to `getLPExchangeRate()` but is given by `latestRoundData()`.

2.5 Changes in Version 4

- A new oracle for Mellow LRT has been added.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	1

- Weights in Balancer Managed Pools Can Change Specification Changed

Low-Severity Findings	2

- Base Token Consistency Code Corrected
- Missing Duplicate Signer Check Code Corrected

Informational Findings	2

- Gas Optimizations Code Corrected
- Permissionless Updating Bounds Code Corrected

6.1 Weights in Balancer Managed Pools Can Change

Design Medium Version 1 Specification Changed

CS-GEARV3ORACLES-008

The Balancer V2 price feed reads the weights of the Balancer pool during deployment and stores them as `immutable`. The oracle also supports managed pools, with `getActualSupply()`, but from the Balancer documentation (https://web.archive.org/web/20230928124529/https://docs.balancer.fi/concepts/pools/managed.html#weights) the weights can be changed by the pool `owner`. Changing the weights of the pool will break the oracle as the price of the respective lp tokens depends on these weights.

Spec changed:

Gearbox Protocol responded that they do not intend to support Balancer-managed pools.

6.2 Base Token Consistency

Design Low Version 4 Code Corrected

CS-GEARV3ORACLES-001

In `MellowLRTPriceFeed`, the `baseToken` is given as a constructor parameter, but it can be read directly from Mellow's price oracle. This would ensure the correct token and decimals are used for the `_vault`.

Code corrected:

The base token of the vault is read from Mellow's price oracle.

6.3 Missing Duplicate Signer Check

Design | Low | Version 1 | Code Corrected

CS-GEARV3ORACLES-007

In the constructor of the `RedstonePriceFeed` contract, there is no check for duplicate signer address in the `_signers` array. If the number of unique signers does not exceed the required threshold, the oracle will not be able to update its price and will become unusable.

Code corrected:

A check for duplicates has been added to the constructor. The deployment will revert if there are duplicates or if the number of signers does not reach the required threshold.

6.4 Gas Optimizations

Informational | Version 1 | Code Corrected

CS-GEARV3ORACLES-005

- The function `latestRoundData` of contracts `ZeroPriceFeed` and `RedstonePriceFeed` unnecessarily define the variables `answer` and `updatedAt`.

Code corrected:

The unnecessary variables have been removed.

6.5 Permissionless Updating Bounds

Informational | Version 1 | Code Corrected

CS-GEARV3ORACLES-006

`LPPriceFeed.updateBounds()` allows everyone to update the bounds of an LP price oracle should `updateBoundsAllowed` be set to true. The bounds depend on the current LP price reported by the reserve oracle and the prices of the underlying assets. It is important to note that if the price of the LP protocol can be manipulated, then the bounds will also be wrongly updated. Hence, enabling the permissionless update of the bound should be carefully considered by the governance of the protocol.

Code corrected:

The buffer size has been increased from `0.2%` to `1%`, allowing a lower lower-bound.

The function `LPPriceFeed.setUpdateBoundsAllowed()` has been split into two functions:

- `LPPriceFeed.setUpdateBoundsAllowed()`: only the configurator can enable permissionless updates
- `LPPriceFeed.forbidBoundsUpdate()`: the configurator and the controller can disable permissionless updates

The function `LPPriceFeed.updateBounds()` has been modified in the following ways:

- a cooldown of 1 day between to permissionless updates is enforced
- the current `LPPriceFeed` explicitly cannot be its own reserve price feed
- the reserve exchange rate must be within the bounds

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Pricefeed Addresses Order

Informational Version 1 Acknowledged

CS-GEARV3ORACLES-004

In the contracts `BPTStablePriceFeed` and `CurveStableLPPriceFeed`, the number of assets is derived from the content of the `priceFeeds` array. The number of assets will be the number of non-zero addresses before the first zero address in the array. It means that if an array of the form `[priceFeed0, priceFeed1, address(0), priceFeed2]` is passed in the construction arguments, the oracle will only consider 2 assets.

Acknowledged:

Gearbox Protocol is aware of this special case.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Implementation of the Integrated ERC4626

Note Version 1

The implementation of the ERC4626 that are whitelisted for use within Gearbox Protocol V3 must be carefully reviewed. As the standard makes very few assumptions about the implementation of a tokenized vault, many issues might arise such as the possibility of read-only reentrancy.

8.2 Read-only Reentrancy on Curve and Balancer Pools

Note Version 1

Some of the `ETH` Curve pools send `ETH` to the LP before the other tokens when removing liquidity, this leads to possible read-only reentrancy. During such reentrancy, the value returned by `get_virtual_price()` is over-evaluated. An attacker could use this to do bad trades or increase its leverage, but it will become liquidatable right after the transaction.

This is also valid for Balancer V2 and the function `getRate()` and `getActualSupply` (see https://web.archive.org/web/20230928133723/https://docs.balancer.fi/reference/contracts/apis/managed.html#getactualsupply).

The prices of all LP price feeds are bounded. This means an attack-based read-only reentrancy can have very limited if any benefit for the attacker. Of course, this is based on the assumption that the bounds are set to reasonable values to prevent such attacks.

8.3 RedStone Price Update Frequency

Note Version 1

The use of RedStone price oracle in the system allows for high-frequency price updates compared to more traditional price oracles like ChainLink. This means, in case of rapid price fluctuations, credit accounts may become liquidatable or seem like they hold a higher value allowing them to borrow or withdraw more assets.

8.4 Use of Stored Exchange Rate for Compound V2

Note Version 1

In `CompoundV2PriceFeed` the price of the lp tokens is computed using `exchangeRateStored()`. The alternative is `exchangeRateCurrent()` which returns a more recent price as it internally calls `accrueInterest()`. It is important to note that `exchangeRateStored()` is called from within a view function and thus it's a view function itself. `exchangeRateCurrent()` cannot be used as it is not a view function.