Report

v. 3.0

Customer

Gearbox

Smart Contract Audit

Oracles

13th December 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	13.12.23	A. Zveryanskaya	Initial Draft
0.2	13.12.23	A. Zveryanskaya	Minor revision
1.0	13.12.23	A. Zveryanskaya	Release
1.1	13.12.23	A. Zveryanskaya	Scope update, numeration fix
2.0	13.12.23	A. Zveryanskaya	Release
2.1	13.12.23	A. Zveryanskaya	Minor revision
3.0	13.12.23	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Gearbox Protocol brings you composable leverage. It allows anyone to margin trade on Uniswap, leverage farm on Curve, leverage liquid stake on Lido, and use 10X more capital on protocols you already love. Making decentralized leverage a reality thanks to Credit Account abstraction, bringing together lending and prime brokerage in the same protocol.

3 Project scope

We were asked to review:

- Original code in commit 539af9a
- Fixes in commit c6e4bd0

Files:

interfaces/			
	ILPPriceFeed.sol		
libraries/			
	FixedPoint.sol	LogExpMath.sol	
oracles/			
	PriceFeedParams.sol	BoundedPriceFeed.sol	CompositePriceFeed.sol
	LPPriceFeed.sol	SingleAssetLPPrice Feed.sol	ZeroPriceFeed.sol
oracles/aave/			
	WrappedAaveV2Price Feed.sol		
oracles/balancer/			
	BPTStablePrice Feed.sol	BPTWeightedPrice Feed.sol	
oracles/compound/			
	CompoundV2Price Feed.sol		
oracles/curve/			
	CurveCryptoLPPrice Feed.sol	CurveStableLPPrice Feed.sol	CurveUSDPrice Feed.sol
oracles/erc4626/			
	ERC4626PriceFeed.sol		
oracles/lido/			
	WstETHPriceFeed.sol		

oracles/updatable/
RedstonePriceFeed.sol
oracles/yearn/
YearnPriceFeed.sol

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 0 critical, 19 major, and a few less important issues.

Issues	Info	Fixed
Severity		
Critical	**0**	**0**
Major	**15**	**4**
Moderate	**2**	**2**
Minor	**25**	**7**

Fixed 13 out of 55 issues

6 Major Issues

CVF-1. FIXED

- **Category** Unclear behavior
- **Source** RedstonePriceFeed.sol

Description This allows a zero signers threshold, which is weird.

Client Comment *signersThreshold now must be non-zero.*

```
122 if (_signersThreshold > 10) revert InvalidSignerSetException();
```

CVF-2. FIXED

- **Category** Suboptimal
- **Source** CurveCryptoLPPriceFeed.sol

Recommendation It should be "* (WAD / USD_FEED_SCALE)" to allow division to be performed at compile time.

Client Comment *Fixed by introducing a constant "WAD_OVER_USD_FEED_SCALE".*

```
62 uint256 product = uint256(answer) * WAD / USD_FEED_SCALE;
```

```
65 product = product.mulDown(uint256(answer) * WAD / USD_FEED_SCALE);
```

```
69     product = product.mulDown(uint256(answer) * WAD / USD_FEED_SCALE
       ↪ );
```

CVF-3. FIXED

- **Category** Suboptimal
- **Source** CurveCryptoLPPriceFeed.sol

Recommendation Should be "/ (WAD / USD_FEED_SCALE)" to allow division to be performed at compile time.

Client Comment *Fixed by introducing a constant "WAD_OVER_USD_FEED_SCALE".*

```
72 answer = int256(nCoins * product.powDown(WAD / nCoins) *
      ↪ USD_FEED_SCALE / WAD);
```

CVF-4. INFO

- **Category** Suboptimal
- **Source** BPTStablePriceFeed.sol

Description Performing these checks on every invocation is suboptimal, as the "numAssets" variable is immutable.

Recommendation Consider implementing four versions of the contract with 2, 3, 4, and 5 price feeds.

Client Comment *Having multiple versions of very similar contracts significantly complicates development and deployment without substantial benefit.*

```
74 if (numAssets > 2) {
```

```
78     if (numAssets > 3) {
```

```
82         if (numAssets > 4) {
```

CVF-5. INFO

- **Category** Suboptimal
- **Source** BPTWeightedPriceFeed.sol

Description The same conditions are checked multiple times. Also, in case a ">=n" check failed, there is no reason to proceed with subsequent checks.

Recommendation Consider refactoring like this: if (numAssets >= 3) { index2 = indices[2]; weight2 = weights[2]; ... if (numAssets >= 4) { index3 = indices[3]; weight3 = weights[3]; ... } }

Client Comment *Acknowledged, but we don't worry that much about constructor performance and it makes initializing immutable variables more readable.*

```
115 index2 = numAssets >= 3 ? indices[2] : 0; // F: [OBWLP-1]
    index3 = numAssets >= 4 ? indices[3] : 0; // F: [OBWLP-1]
    index4 = numAssets >= 5 ? indices[4] : 0; // F: [OBWLP-1]
    index5 = numAssets >= 6 ? indices[5] : 0; // F: [OBWLP-1]
    index6 = numAssets >= 7 ? indices[6] : 0; // F: [OBWLP-1]
120 index7 = numAssets >= 8 ? indices[7] : 0; // F: [OBWLP-1]
```

```
124 weight2 = numAssets >= 3 ? weights[2] : 0; // F: [OBWLP-1]
    weight3 = numAssets >= 4 ? weights[3] : 0; // F: [OBWLP-1]
    weight4 = numAssets >= 5 ? weights[4] : 0; // F: [OBWLP-1]
    weight5 = numAssets >= 6 ? weights[5] : 0; // F: [OBWLP-1]
    weight6 = numAssets >= 7 ? weights[6] : 0; // F: [OBWLP-1]
    weight7 = numAssets >= 8 ? weights[7] : 0; // F: [OBWLP-1]
```

```
134 scale2 = numAssets >= 3 ? _tokenScale(tokens[index2]) : 0; // F: [
       ↪ OBWLP-1]
    scale3 = numAssets >= 4 ? _tokenScale(tokens[index3]) : 0; // F: [
       ↪ OBWLP-1]
    scale4 = numAssets >= 5 ? _tokenScale(tokens[index4]) : 0; // F: [
       ↪ OBWLP-1]
    scale5 = numAssets >= 6 ? _tokenScale(tokens[index5]) : 0; // F: [
       ↪ OBWLP-1]
    scale6 = numAssets >= 7 ? _tokenScale(tokens[index6]) : 0; // F: [
       ↪ OBWLP-1]
    scale7 = numAssets >= 8 ? _tokenScale(tokens[index7]) : 0; // F: [
       ↪ OBWLP-1]
```

(143, 152, 161, 263, 278)

CVF-6. INFO

- **Category** Suboptimal
- **Source** BPTWeightedPriceFeed.sol

Recommendation The set of indexes where weighted price should be updated, precomputed and stored in an internal variable as a bit mask.

Client Comment *Acknowledged, but the gas savings are too insignificant to harm readability.*

186
```
if (i == numAssets - 1 || weights[i] != weights[i + 1]) {
```

218
```
if (i == len - 1 || weights[i] != weights[i + 1]) {
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** BPTWeightedPriceFeed.sol

Recommendation It would be more efficient to use a binary decision tree like this:

```
if (i < 4) \{
  if (i < 2) \{
    if (i == 0) return (priceFeed0, stalenessPeriod0, skipCheck0);
    else return (priceFeed1, stalenessPeriod1, skipCheck1);
  \} else \{
    if (i == 2) return (priceFeed2, stalenessPeriod2, skipCheck2);
    else return (priceFeed3, stalenessPeriod3, skipCheck3);
  \}
\} else \{
  if (i < 6) \{
    if (i == 4) return (priceFeed4, stalenessPeriod4, skipCheck4);
    else return (priceFeed5, stalenessPeriod5, skipCheck5);
  \} else \{
    if (i == 6) return (priceFeed6, stalenessPeriod6, skipCheck6);
    else if (i == 7) return (priceFeed7, stalenessPeriod7,
       ↪ skipCheck7);
  \}
\}
```

Client Comment *Acknowledged, but the gas saving are too insignificant to introduce this change.*

```
248 if (i == 0) return (priceFeed0, stalenessPeriod0, skipCheck0);
    if (i == 1) return (priceFeed1, stalenessPeriod1, skipCheck1);
250 if (i == 2) return (priceFeed2, stalenessPeriod2, skipCheck2);
    if (i == 3) return (priceFeed3, stalenessPeriod3, skipCheck3);
    if (i == 4) return (priceFeed4, stalenessPeriod4, skipCheck4);
    if (i == 5) return (priceFeed5, stalenessPeriod5, skipCheck5);
    if (i == 6) return (priceFeed6, stalenessPeriod6, skipCheck6);
    if (i == 7) return (priceFeed7, stalenessPeriod7, skipCheck7);
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** BPTWeightedPriceFeed.sol

Description While quick sort algorithm is quite efficient in general case, it is considered inefficient for small arrays, and non-recursions algorithms are used for such arrays: https://en.wikipedia.org/wiki/Quicksort#Optimizations

Recommendation As here the maximum array size is 8, consider using some non-recursive algorithm with unrolled loops. For example bubble sort with 28 comparisons.

Client Comment *Acknowledged, but this function is only called once in constructor so inefficiencies are accetpable.*

```
308 function _quickSort(uint256[] memory data, uint256[] memory indices,
    ↪  uint256 low, uint256 high) private pure {
```

CVF-9. FIXED

- **Category** Procedural
- **Source** LPPriceFeed.sol

Description Silently doing nothing in case an operation is not allowed is a bad practice.

Recommendation Consider reverting instead.

Client Comment *Fixed, it now reverts instead.*

```
127 if (!updateBoundsAllowed) return;
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Description Precision of these number is worse than the number format permits.

Recommendation Consider increasing precision. For example, e^(2^7) rounded to the closest integer is 38877084059945950922226736883574780727281750630829988858

Client Comment *Acknowledged, yet we're trying not to make any changes to "LogExpMath" and "FixedPoint" except those necessary for compiling the project.*

```
60 int256 constant a0 =
   ↪ 38877084059945950922200000000000000000000000000000000000; //
   ↪ ^e(x0) (no decimals)

62 int256 constant a1 = 6235149080811616882910000000; // ^e(x1) (no
   ↪ decimals)

66 int256 constant a2 = 7896296018268069516100000000000000; // ^e(x2)

68 int256 constant a3 = 888611052050787263676000000; // ^e(x3)

70 int256 constant a4 = 298095798704172827474000; // ^e(x4)
```

CVF-11. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Description This function seems overcomplicated and suboptimal.

Recommendation Calculating 2^x would be more efficient that e^x as the integer part of the argument could be handled by shifting the result. Representing the argument as a binary fixed point rather than decimal fixed point would allow using bit operation instead of comparisons and subtractions when handling the fractional part of the argument. Note that e^x = 2^(log2 e * x), and multiplying x by log2 e could be combined with converting x from a decimal fixed point number into a binary fixed point number.

Client Comment *See response to 10*

```
145 function exp(int256 x) internal pure returns (int256) {
```

CVF-12. INFO

- **Category** Unclear behavior
- **Source** LogExpMath.sol

Description Reverting on too little argument seems weird.

Recommendation Consider just returning zero in such a case.

Client Comment *See response to 10.*

```
147 require(x >= MIN_NATURAL_EXPONENT && x <= MAX_NATURAL_EXPONENT, "
      ↪ invalid␣exponent");
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Recommendation It would be more efficient to precompute ONE_20 * n constants.

Client Comment *See response to 10.*

```
240 term = ((term * x) / ONE_20) / 2;
243 term = ((term * x) / ONE_20) / 3;
246 term = ((term * x) / ONE_20) / 4;
249 term = ((term * x) / ONE_20) / 5;
252 term = ((term * x) / ONE_20) / 6;
255 term = ((term * x) / ONE_20) / 7;
258 term = ((term * x) / ONE_20) / 8;
261 term = ((term * x) / ONE_20) / 9;
264 term = ((term * x) / ONE_20) / 10;
267 term = ((term * x) / ONE_20) / 11;
270 term = ((term * x) / ONE_20) / 12;
```

CVF-14. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Description This function is overcomplicated and suboptimal.

Recommendation It would be more efficient to calculate a base-2 logarithm as a binary-ixed point number, and then convert to a natural logarithm represented as a decimal fixed point. In particular switching to base-2 logarithm would allow bit-shifting the argument to make it fit into [1..2) range, and adding the shift amount to the result. always having the argument in [1..2) range would allow using higer precision calcualtions (127 bits). Calcualting the result as a binary fixed point number would allow calcualting it bit by bit.

Client Comment *See response to 10.*

```
331 function _ln(int256 a) private pure returns (int256) {
```

CVF-15. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Recommendation These multiplication and divisions could by avoided by just squaring “a” between iterations like this:

```
if (a >= a2) sum += x2;
a = a * a / ONE\_20;
if (a >= a2) sum += x3;
a = a * a / ONE\_20;
if (a >= a2) sum += x4;
a = a * a / ONE\_20;
```

And if “a” would be a binary fixed point number, the divisions here could be replaced with shifts.

Client Comment *See response to 15.*

```
373 a = (a * ONE_20) / a2;
378 a = (a * ONE_20) / a3;
383 a = (a * ONE_20) / a4;
388 a = (a * ONE_20) / a5;
393 a = (a * ONE_20) / a6;
398 a = (a * ONE_20) / a7;
403 a = (a * ONE_20) / a8;
408 a = (a * ONE_20) / a9;
413 a = (a * ONE_20) / a10;
418 a = (a * ONE_20) / a11;
```

CVF-16. INFO

- **Category** Documentation
- **Source** FixedPoint.sol

Description A value returned by the function is not the true value rounded down, but rather some estimation guaranteed to not exceed the true value.

Recommendation Consider not using the “rounding” term.

Client Comment *See response to 10.*

```
68 * @dev Returns x^y, assuming both are fixed point numbers, rounding
      ↪ down. The result is guaranteed to not be above
```

CVF-17. INFO

- **Category** Suboptimal
- **Source** FixedPoint.sol

Description Such optimization actually makes the generic case less efficient.

Recommendation Consider implementing a separate function for raising a fixed-point value into an integer power.

Client Comment *See response to 10.*

```
72 // Optimize for when y equals 1.0, 2.0 or 4.0, as those are very
      ↪ simple to implement and occur often in 50/50
```

```
98 // Optimize for when y equals 1.0, 2.0 or 4.0, as those are very
      ↪ simple to implement and occur often in 50/50
```

CVF-18. INFO

- **Category** Documentation
- **Source** FixedPoint.sol

Description A value returned by the function is not the true value rounded up, but rather some estimation guaranteed to not be below the true value.

Recommendation Consider not using the “rounding” term.

Client Comment *See response to 10.*

```
97 function powUp(uint256 x, uint256 y) internal pure returns (uint256)
      ↪  {
```

CVF-19. INFO

- **Category** Overflow/Underflow
- **Source** LogExpMath.sol

Description Overflow is prevented here by limiting the argument, which could be too restrictive.

Recommendation Consider just using checked math. Also, consider using the “muldiv” function to simplify code make make it more resilient.

Client Comment *See response to 10.*

```
125 logx_times_y = ((ln_36_x / ONE_18) * y_int256 + ((ln_36_x % ONE_18)
      ↪ * y_int256) / ONE_18);
```

```
127 logx_times_y = _ln(x_int256) * y_int256;
```

7 Moderate Issues

CVF-20. INFO

- **Category** Unclear behavior
- **Source** BPTWeightedPriceFeed.sol

Description This function return default (i.e. zero) on invalid argument.

Recommendation Consider reverting in such a case.

Client Comment *The function is internal and is never called with invalid argument.*

```
243 function _getPriceFeedParams(uint256 i)
```

CVF-21. FIXED

- **Category** Flaw
- **Source** RedstonePriceFeed.sol

Description This constraint could be bypassed by setting non-unique signer addresses.

Recommendation Consider checking the signer addresses for uniqueness.

```
122 if (_signersThreshold > 10) revert InvalidSignerSetException();
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** BPTStablePriceFeed.sol

Description The “updatedAt” value returned always belong to the price from “priceFeed0” even if the actual price was taken from a different price feed.

Client Comment *”updatedAt” removed across repository for all price feeds where ”skipPriceCheck = true”*

```
68 function getAggregatePrice() public view override returns (int256
   ↪ answer, uint256 updatedAt) {
```

CVF-23. INFO

- **Category** Flaw
- **Source** LogExpMath.sol

Description There are no checks to ensure that the arguments are non zero.

Recommendation Consider adding appropriate checks or use the “ln” function instead of “_ln_36” or “_ln”.

Client Comment *See response to 10.*

```
287 function log(int256 arg, int256 base) internal pure returns (int256)
        ↪ {
```

8 Minor Issues

CVF-24. INFO

- **Category** Procedural
- **Source** YearnPriceFeed.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for: RedstonePriceFeed.sol, WstETHPriceFeed.sol, ERC4626PriceFeed.sol, CurveCryptoLPPriceFeed.sol, CurveStableLPPriceFeed.sol, CurveUSDPriceFeed.sol, CompoundV2PriceFeed.sol, BPTStablePriceFeed.sol, BPTWeightedPriceFeed.sol, WrappedAaveV2PriceFeed.sol, AbstractPriceFeed.sol, LPPriceFeed.sol, SingleAssetLPPriceFeed.sol, BoundedPriceFeed.sol, CompositePriceFeed.sol, ZeroPriceFeed.sol, IPriceFeed.sol, ILPPriceFeed.sol.

```
4 pragma solidity ^0.8.17;
```

CVF-25. INFO

- **Category** Procedural
- **Source** RedstonePriceFeed.sol

Description We didn't review this file.

```
8 "@redstone-finance/evm-connector/contracts/core/
    ↪ RedstoneConsumerNumericBase.sol";
```

CVF-26. INFO

- **Category** Procedural
- **Source** RedstonePriceFeed.sol

Recommendation This interface should be moved into a separate file named "IRedstonePriceFeedExceptions.sol".

Client Comment *We keep small helper interfaces close to where they are used.*

```
14 interface IRedstonePriceFeedExceptions {
```

CVF-27. INFO

- **Category** Procedural
- **Source** RedstonePriceFeed.sol

Recommendation This interface should be moved into a separate file named "IRedstonePriceFeed.sol".

Client Comment *See response to 26.*

```
37 interface IRedstonePriceFeedEvents {
```

CVF-28. FIXED

- **Category** Documentation
- **Source** RedstonePriceFeed.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
39 event PriceUpdated(uint256 price);
```

CVF-29. FIXED

- **Category** Procedural
- **Source** RedstonePriceFeed.sol

Description Making this variable public is redundant, as there is dedicated getter for it.

Recommendation Consider declaring this variable as internal.

```
102 uint8 public immutable signersThreshold;
```

CVF-30. FIXED

- **Category** Bad datatype
- **Source** RedstonePriceFeed.sol

Recommendation The value "10" should be a named constant.

```
122 if (_signersThreshold > 10) revert InvalidSignerSetException();
```

CVF-31. INFO

- **Category** Suboptimal
- **Source** RedstonePriceFeed.sol

Recommendation A binary decision tree would be more efficient, however this would require sorting the signer addresses.

```
152 if (signerAddress == signerAddress0) return 0;
    if (signerAddress == signerAddress1) return 1;
    if (signerAddress == signerAddress2) return 2;
    if (signerAddress == signerAddress3) return 3;
    if (signerAddress == signerAddress4) return 4;
    if (signerAddress == signerAddress5) return 5;
    if (signerAddress == signerAddress6) return 6;
    if (signerAddress == signerAddress7) return 7;
160 if (signerAddress == signerAddress8) return 8;
    if (signerAddress == signerAddress9) return 9;
```

CVF-32. FIXED

- **Category** Documentation
- **Source** RedstonePriceFeed.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or documenting.

Client Comment *"getRoundData" is removed, variables in "latestRoundData" named.*

```
222 function latestRoundData() external view override returns (uint80,
      ↪ int256, uint256, uint256, uint80) {
```

```
233 function getRoundData(uint80) external pure override returns (uint80
      ↪ , int256, uint256, uint256, uint80) {
```

CVF-33. INFO

- **Category** Suboptimal
- **Source** ERC4626PriceFeed.sol

Description The “decimals” property is used by UI to render asset amounts in a human-eadable way. Using this property in smart contract is discouraged.

Recommendation Consider treating all asset amounts as integers.

Client Comment *In this case, there’s no robust way to properly compute prices without relying on decimals.*

```
25 _shareUnit = 10 ** IERC4626(_vault).decimals();
   _assetUnit = 10 ** ERC20(IERC4626(_vault).asset()).decimals();
```

CVF-34. INFO

- **Category** Readability
- **Source** CurveCryptoLPPriceFeed.sol

Description Defining a top-level constant in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the constant definition into the contract or moving it into a separate file.

```
13 uint256 constant USD_FEED_SCALE = 10 ** 8;
```

CVF-35. INFO

- **Category** Suboptimal
- **Source** CurveCryptoLPPriceFeed.sol

Description Branching execution based on an immutable variable is waste of gas.

Recommendation Consider implementing two versions of the contract: one for two feeds and another for three feeds.

Client Comment *See response to 4.*

```
67 if (nCoins == 3) {
```

CVF-36. INFO

- **Category** Unclear behavior
- **Source** CurveStableLPPriceFeed.sol

Description This check shouldn't be performed in case in the previous check (nCoins > 2) failed.

Client Comment *True, but because this statement initializes an immutable variable, rewriting it would clutter the code, and the check is cheap enough to forego it.*

57
```
skipCheck3 = nCoins > 3 ? _validatePriceFeed(priceFeed3,
    ↪ stalenessPeriod3) : false;
```

CVF-37. INFO

- **Category** Suboptimal
- **Source** CurveStableLPPriceFeed.sol

Description Branching execution based on an immutable variable is waste of gas.

Recommendation Consider implementing thee versions of the contract: for two, three, and four feeds.

Client Comment *See response to 4.*

72
```
if (nCoins > 2) {
```

76
```
    if (nCoins > 3) {
```

CVF-38. INFO

- **Category** Procedural
- **Source** CurveUSDPriceFeed.sol

Recommendation This interface should be moved into a separate file named "IPool.sol".

Client Comment *See response to 26.*

10
```
interface IPool {
```

CVF-39. INFO

- **Category** Documentation
- **Source** WrappedAaveV2PriceFeed.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *Its meaning is rather clear given that "getScale" is right below this function.*

```
22 function getLPExchangeRate() public view override returns (uint256)
      ↪ {
```

CVF-40. FIXED

- **Category** Readability
- **Source** AbstractPriceFeed.sol

Description Defining a top-level structure in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the struct definition into the contract or moving it into a separate file.

Client Comment *Moved to a separate file, the contract itself is removed.*

```
11 struct PriceFeedParams {
```

CVF-41. INFO

- **Category** Bad naming
- **Source** LPPriceFeed.sol

Description This variable names are too generic.

Recommendation Consider making them more specific.

Client Comment *Their meaning is fairly clear from the code.*

```
38 uint256 public override lowerBound;
```

```
41 uint256 public constant override delta = 2_00;
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** LPPriceFeed.sol

Recommendation These checks are redundant as it is anyway possible to pass a dead address.

Client Comment *It allows to prevent some deploy process bugs rather than protects from using dead addresses.*

```
52 nonZeroAddress(_lpToken)
   nonZeroAddress(_lpContract)
```

CVF-43. INFO

- **Category** Documentation
- **Source** LPPriceFeed.sol

Description The semantics of the unnamed return values is unclear.

Recommendation Consider giving descriptive names to all return values, and/or describing them in a documentation comment.

Client Comment *Those values are ignored so it's better to leave them as is, although one can easily lookup Chainlink interface documentation.*

```
70 returns (uint80, int256 answer, uint256, uint256 updatedAt, uint80)
```

CVF-44. FIXED

- **Category** Bad datatype
- **Source** LPPriceFeed.sol

Recommendation The values 998 and 1000 should be named constants.

```
136 _setLimiter(reserveExchangeRate * 998 / 1000, getLPExchangeRate());
```

```
144 _setLimiter(exchangeRate * 998 / 1000, exchangeRate);
```

CVF-45. FIXED

- **Category** Bad datatype
- **Source** LPPriceFeed.sol

Recommendation The first two conditions could be checked earlier.

```
150 if (lower == 0 || current < lower || current > upper) revert
    ↪ IncorrectLimitsException();
```

CVF-46. INFO

- **Category** Documentation
- **Source** SingleAssetLPPriceFeed.sol

Description The number format of the “answer” return value is unclear.

Recommendation Consider documenting.

Client Comment *Documented in the base class.*

```
29 function getAggregatePrice() public view override returns (int256
    ↪ answer, uint256 updatedAt) {
```

CVF-47. INFO

- **Category** Procedural
- **Source** BoundedPriceFeed.sol

Recommendation This interface should be moved into a separate file named “ChainlinkReadableAggregator.sol”.

Client Comment *See response to 26.*

```
11 interface ChainlinkReadableAggregator {
```

CVF-48. INFO

- **Category** Documentation
- **Source** CompositePriceFeed.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or describing in the documentation comment.

Client Comment *Documented in the base class.*

```
65 returns (uint80, int256 answer, uint256, uint256 updatedAt, uint80)
```

CVF-49. INFO

- **Category** Overflow/Underflow
- **Source** CompositePriceFeed.sol

Description Phantom overflow is possible here.

Recommendation Consider using the muldiv function.

Client Comment *Unlikely to occur in practice unless Chainlink goes off, in which case we don't mind if this reverts.*

```
69 answer = (answer * answer2) / targetFeedScale;
```

CVF-50. INFO

- **Category** Bad naming
- **Source** ZeroPriceFeed.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving descriptive names to the returned values and/or describing in the documentation comment.

```
18 function latestRoundData() external view override returns (uint80,
   ↪ int256, uint256, uint256, uint80) {
```

CVF-51. INFO

- **Category** Suboptimal
- **Source** LogExpMath.sol

Description Multiplication after division could lead to precision degradation.

Recommendation Consider refactoring.

Client Comment *See response to 10.*

```
280 return (((product * seriesSum) / ONE_20) * firstAN) / 100;
```

CVF-52. INFO

- **Category** Procedural
- **Source** ILPPriceFeed.sol

Recommendation This interface should be moved into a separate file named "ILPPrice-FeedEvents.sol".

Client Comment *See response to 26.*

```
8 interface ILPPriceFeedEvents {
```

CVF-53. INFO

- **Category** Bad naming
- **Source** ILPPriceFeed.sol

Recommendation Events are usually named via nouns, such as "Bounds".

Client Comment *The convention for events naming in our codebase is action verb + object*

```
10 event SetBounds(uint256 lowerBound, uint256 upperBound);
```

```
13 event SetUpdateBoundsAllowed(bool allowed);
```

CVF-54. INFO

- **Category** Documentation
- **Source** ILPPriceFeed.sol

Description It is unclear how the upper bound is determined.

Recommendation Consider documenting.

Client Comment *The function is documented in the contract that implements this interface.*

```
37 function setLimiter(uint256 newLowerBound) external;
```

CVF-55. INFO

- **Category** Documentation
- **Source** ILPPriceFeed.sol

Description The semantics of this function is unclear.

Recommendation Consider documenting.

Client Comment *The function is documented in the contract that implements this interface.*

```
38 function updateBounds() external;
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting