Report

v. 2.0

Customer

RedStone

Smart Contract Audit

Oracles

13th April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	09.04.24	A. Zveryanskaya	Initial Draft
0.2	09.04.24	A. Zveryanskaya	Minor revision
1.0	09.04.24	A. Zveryanskaya	Release
1.1	13.04.24	A. Zveryanskaya	CVF-1, 2, 3 downgraded
2.0	13.04.24	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.

RedStone is an Oracle that delivers frequently updated, reliable, and diverse data feeds for dApp and smart contracts on multiple L1s and L2s.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

```
/
    SinglePrice FeedAdapter.sol
    LayerBank OracleAdapterV1.sol
    LayerBank OracleAdapterBase.sol
    RedstonePrimary ProdWithout RoundsERC7412.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 Recommendations

We found several minor issues and provided Client with recommendations for consideration.

CVF-1 INFO

- **Category** Suboptimal
- **Source** LayerBankOracleAdapterV1.sol

Description Hardcoding mainnet addresses is a bad practice as it makes the code harder to test and harder to deploy on different networks.

Recommendation Consider passing the addresses as constructor argument and storing in immutable variables.

Client Comment *This is "configured" contract which is not directly used in tests. This allows us to save gas, because we don't have to use SLOAD to read variables. We cant pass arguments via constructor cause we are using upgradable contracts.*

```
22 if (asset == 0xb73603C5d87fA094B7314C74ACE2e64D165016fb) {
24 } else if (asset == 0x6Fae4D9935E2fcb11fC79a64e917fb2BF14DaFaa) {
26 } else if (asset == 0x20A512dbdC0D006f46E6cA11329034Eb3d18c997) {
28 } else if (asset == 0x2FE3AD97a60EB7c79A976FC18Bb5fFD07Dd94BA5) {
30 } else if (asset == 0xEc901DA9c68E90798BbBb74c11406A32A70652C3) {
32 } else if (asset == 0xbdAd407F77f44F7Da6684B416b1951ECa461FB07) {
34 } else if (asset == 0x95CeF13441Be50d20cA4558CC0a27B601aC544E5) {
```

CVF-2 INFO

- **Category** Suboptimal
- **Source** SinglePriceFeedAdapter.sol

Recommendation Consider using the hash expression instead of a hardcoded hash value. Solidity compiler is smart enough to calculated hashes of constant data at compile time.

Client Comment *TypeError: Only direct number constants and references to such constants are supported by inline assembly.*

```
22 bytes32 internal constant DATA_FROM_LATEST_UPDATE_STORAGE_LOCATION =
   0x632f4a585e47073d66129e9ebce395c9b39d8a1fc5b15d4d7df2e462fb1fccfa;
      ↪ // keccak256("RedStone.singlePriceFeedAdapter");
```

CVF-3 INFO

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

Description The returned value has type ”uint128” while actually they are 48-bit numbers.

Recommendation Consider using ”uint256” or ”uint48” type instead.

Client Comment *TypeError: Overriding function return types differ.*

```
92 function getTimestampsFromLatestUpdate() public view virtual
      ↪ override returns (uint128 dataTimestamp, uint128
      ↪ blockTimestamp) {
```

CVF-4 INFO

- **Category** Procedural
- **Source** LayerBankOracleAdapterBase.sol

Description Consider specifying as ”^0.8.0” unless there is something special regarding this particular version.

Recommendation Also relevant for: LayerBankOracleAdapterV1.sol, RedstonePrimaryProdWithoutRoundsERC7412.sol, SinglePriceFeedAdapter.sol.

Client Comment *Version 0.8.14 is used across all .sol files.*

```
3 pragma solidity ^0.8.14;
```

CVF-5 INFO

- **Category** Procedural
- **Source** LayerBankOracleAdapterBase.sol

Recommendation We didn't review these files.

```
import {ILToken} from "./ILToken.sol";
import {IPriceCalculator} from "./IPriceCalculator.sol";
import {PriceFeedsAdapterWithRoundsPrimaryProd} from "../../price-feeds/data-services/PriceFeedsAdapterWithRoundsPrimaryProd.sol";
```

CVF-6 INFO

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

Recommendation The parameter type should be more specific.

Client Comment *This is enforced by external Interface IPriceCalculator.*

```
error UnsupportedAsset(address asset);
```

CVF-7 INFO

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

Recommendation The argument types should be more specific.

Client Comment *I think we can't do it because we are implementing IPriceCalculator interface which uses a generic "address". This external interface we can't modify it.*

```
13 function getDataFeedIdForAsset(address asset) public view virtual
   ↪ returns(bytes32);
```

```
21 function _uncheckedPriceOf(address asset) internal view virtual
   ↪ returns (uint256) {
```

```
27 function priceOf(address asset) public view virtual returns (uint256
   ↪ ) {
```

```
33   address[] memory assets
```

CVF-8 FIXED

- **Category** Documentation
- **Source** LayerBankOracleAdapterBase.sol

Description The exact semantics of this function is unclear.

Recommendation Consider documenting.

```
14 function convertDecimals(bytes32 dataFeedId, uint256
   ↪ valueFromRedstonePayload) public view virtual returns (uint256
   ↪ );
```

CVF-9 INFO

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

Recommendation The return type should be more specific.

Client Comment *We can't do it because we are implementing IPriceCalculator interface which uses a generic "address". This external interface we can't modify it.*

```
17 function getUnderlyingAsset(address gToken) public view virtual
   ↪ returns(address) {
```

CVF-10 INFO

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

Recommendation The argument type should be "ILToken".

Client Comment *We can't do it because we are implementing IPriceCalculator interface which uses a generic "address". This external interface we can't modify it.*

```
17 function getUnderlyingAsset(address gToken) public view virtual
   ↪ returns(address) {
```

```
42 function getUnderlyingPrice(address gToken) public view returns (
   ↪ uint256) {
```

CVF-11 FIXED

- **Category** Documentation
- **Source** LayerBankOracleAdapterBase.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

```
21 function _uncheckedPriceOf(address asset) internal view virtual
   ↪ returns (uint256) {
```

```
27 function priceOf(address asset) public view virtual returns (uint256
   ↪ ) {
```

```
34 ) external view returns (uint256[] memory values) {
```

```
42 function getUnderlyingPrice(address gToken) public view returns (
   ↪ uint256) {
```

CVF-12 INFO

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

Recommendation The argument type should be "ILToken[]".

Client Comment *We can't do it because we are implementing IPriceCalculator interface which uses a generic "address". This external interface we can't modify it.*

```
47 address[] memory gTokens
```

CVF-13 FIXED

- **Category** Suboptimal
- **Source** LayerBankOracleAdapterV1.sol

Recommendation Here "address(0)" would be more compact.

```
9 address internal constant ETH_ASSET = 0
  ↪ x0000000000000000000000000000000000000000;
```

CVF-14 INFO

- **Category** Bad datatype
- **Source** LayerBankOracleAdapterV1.sol

Recommendation The argument type should be more specific.

```
21 function getDataFeedIdForAsset(address asset) public view virtual
   ↪ override returns(bytes32) {
```

CVF-15 FIXED

- **Category** Bad datatype
- **Source** LayerBankOracleAdapterV1.sol

Recommendation These addresses should be named constants.

```
22 if (asset == 0xb73603C5d87fA094B7314C74ACE2e64D165016fb) {
24 } else if (asset == 0x6Fae4D9935E2fcb11fC79a64e917fb2BF14DaFaa) {
26 } else if (asset == 0x20A512dbdC0D006f46E6cA11329034Eb3d18c997) {
28 } else if (asset == 0x2FE3AD97a60EB7c79A976FC18Bb5fFD07Dd94BA5) {
30 } else if (asset == 0xEc901DA9c68E90798BbBb74c11406A32A70652C3) {
32 } else if (asset == 0xbdAd407F77f44F7Da6684B416b1951ECa461FB07) {
34 } else if (asset == 0x95CeF13441Be50d20cA4558CC0a27B601aC544E5) {
```

CVF-16 FIXED

- **Category** Bad datatype
- **Source** LayerBankOracleAdapterV1.sol

Recommendation The value ”1e10” should be a named constant.

```
57 return valueFromRedstonePayload * 1e10;
```

CVF-17 FIXED

- **Category** Procedural
- **Source** RedstonePrimaryProdWithoutRoundsERC7412.sol

Description This version requirement is inconsistent with other files.

Recommendation Consider using consistent version requirements across the code base.

```
2 pragma solidity ^0.8.12;
```

CVF-18 INFO

- **Category** Procedural
- **Source** RedstonePrimaryProdWithoutRoundsERC7412.sol

Description We didn't review this file.

```
4 import {IERC7412} from './IERC7412.sol';
  import {MergedSinglePriceFeedAdapterWithoutRoundsPrimaryProd} from '
    ↪ ../price-feeds/data-services/
    ↪ MergedSinglePriceFeedAdapterWithoutRoundsPrimaryProd.sol';
```

CVF-19 FIXED

- **Category** Bad datatype
- **Source** RedstonePrimaryProdWithoutRoundsERC7412.sol

Recommendation These values should be named constants.

```
25 maxDataAheadSeconds = 120;
   maxDataDelaySeconds = 120;
```

CVF-20 FIXED

- **Category** Readability
- **Source** SinglePriceFeedAdapter.sol

Recommendation Should be "else revert".

```
55 revert DataFeedIdNotFound(dataFeedId);
```

CVF-21 INFO

- **Category** Suboptimal
- **Source** SinglePriceFeedAdapter.sol

Recommendation This check could be optimized like this: if (valueForDataFeed » 20 != 0) ...

Client Comment *Using hardhat benchmarks it seems to cost more 7 gas diff. Also it should be 160 not 20.*

```
67 if (valueForDataFeed > MAX_VALUE_WITH_20_BYTES) {
```

CVF-22 FIXED

- **Category** Suboptimal
- **Source** SinglePriceFeedAdapter.sol

Recommendation The type conversion here is redundant, as Solidity compiler would anyway do it automatically.

```
83 return uint256(dataFeedValueCompressed);
```

CVF-23 INFO

- **Category** Suboptimal
- **Source** SinglePriceFeedAdapter.sol

Recommendation These checks should be optimized like this: if (x » 48 != 0) ...

Client Comment *Using hardhat benchmarks it seems to cost more.*

```
126 if (dataPackagesTimestamp > MAX_NUMBER_FOR_48_BITS) {
```

```
130 if (blockTimestamp > MAX_NUMBER_FOR_48_BITS) {
```

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