Report

v. 2.0

Customer

RedStone

Smart Contract Audit

Oracle. Phase II

16th June 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	15.06.23	A. Zveryanskaya	Initial Draft
0.2	15.06.23	A. Zveryanskaya	Minor revision
1.0	15.06.23	A. Zveryanskaya	Release
1.1	16.06.23	A. Zveryanskaya	CVF-4,5 are removed
1.2	16.06.23	A. Zveryanskaya	CVF-7,8 are downgraded
2.0	16.06.23	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.

RedStone builds a novel oracle that delivers data feeds for 1,100+ assets, available on Ethereum, Polygon, zkEVM, Avalanche, Arbitrum, and 30+ chains with 10 second update time. Thanks to RedStone's StorageLess architecture, which bypasses expensive EVM-storage, projects can integrate it once and use on all EVM chains.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

```
/
```

CallDataExtractor.sol	PriceFeedBase.sol	PriceFeedsAdapterBase.sol
PriceFeedsAdapterWithoutRounds.sol	PriceFeedsAdapterWithRounds.sol	PriceFeedWithoutRounds.sol
PriceFeedWithRounds.sol	RedstoneAdapterBase.sol	SinglePriceFeedAdapter.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 2 major, and a few less important issues.

	Active	Fixed
Major	1	1
Moderate	2	4
Minor	7	11

Fixed 16 out of 26 issues

6 Major Issues

CVF-1. INFO

- **Category** Suboptimal
- **Source** CalldataExtractor.sol

Description Performing this check on every loop iteration is suboptimal.

Recommendation Consider extracting the first timestamp before the loop, and then looping through the remaining timestamps.

Client Comment *It would significantly complicate the code and thanks to the optimiser, the gas gains would not be significant.*

```
40 if (extractedTimestamp == 0) {
```

CVF-9. FIXED

- **Category** Procedural
- **Source** SinglePriceFeedAdapter.sol

Description This code assumes that both values fit into 48 bits.

Recommendation Consider explicitly asserting this fact.

```
66 uint256 timestampsPacked = dataPackagesTimestamp << 208; // 48 first
   ↪  bits for dataPackagesTimestamp
timestampsPacked |= (block.timestamp << 160); // 48 next bits for
   ↪ block.timestamp
```

7 Moderate Issues

CVF-2. FIXED

- **Category** Procedural
- **Source** PriceFeedBase.sol

Recommendation These functions should probably be made pure virtual, as their implementation in this contract doesn't look real.

```
11 function getDataFeedId() public view virtual returns (bytes32) {
```

```
46 function latestRound() public view virtual returns (uint80) {
```

CVF-7. INFO

- **Category** Suboptimal
- **Source** RedstoneAdapterBase.sol

Recommendation Linear search could be replaced with more efficient binary search here, if the "getDataFeedIds" function would be required to return a sorted array.

Client Comment *Binary search would complicate the code and would not really improve the gas costs for arrays with small length (we benchmarked it before) and we expect these arrays to be quite small, usually containing only one element. Additionally, this function is virtual and we can always override it in real contracts (we do it currently this way), which can really improve gas costs.*

```
57 for (uint256 i = 0; i < dataFeedIds.length; i++) {
     if (dataFeedIds[i] == dataFeedId) {
       return i;
60   }
   }
```

CVF-8. INFO

- **Category** Suboptimal
- **Source** RedstoneAdapterBase.sol

Description While making off-chain usages more convenient, this logic increases cost of on-chain usages.

Recommendation Consider removing this logic.

Client Comment *We can not get rid of this due to the nature of our evm-connector smart contracts. This function is called few times per tx, so the gas costs are not really higher because of this (there are just few opcodes).*

```
84 // It means that we are in the special view context and we can skip
   ↪ validation of the
   // timestamp. It can be useful for calling view functions, as they
   ↪ can not modify the contract
   // state to pass the timestamp validation below
   if (msg.sender == address(0)) {
     return;
   }
```

CVF-10. FIXED

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

Description Overflow is possible during type conversion.

Recommendation Consider using safe conversion.

```
43 return int256(getPriceFeedAdapter().getValueForDataFeed(dataFeedId))
   ↪ ;
```

CVF-11. FIXED

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

```
23 answer = int256(dataFeedValue);
```

CVF-12. FIXED

- **Category** Procedural
- **Source** RedstoneAdapterBase.sol

Description This logic assumes that both values fit into 128 bits.

Recommendation Consider explicitly asserting this.

```
188 let timestamps := or(shl(BITS_COUNT_IN_16_BYTES,
       ↪ dataPackagesTimestamp), blockTimestamp)
```

8 Minor Issues

CVF-3. FIXED

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

Description Despite the name, this function doesn't contain anything regarding validating the data feeds values.

Recommendation Consider renaming the "_updateDataFeedValue" function into "_validateAndUpdateDataFeedValue".

```
13 function validateAndUpdateDataFeedsValues(
```

CVF-13. INFO

- **Category** Suboptimal
- **Source** CalldataExtractor.sol

Description Maintaining a negative offset counter and converting it to normal offset at every iteration is suboptimal.

Recommendation Consider maintaining a normal offset counter instead.

Client Comment *Thanks to optimiser it should not be more expensive, but we can not easily switch to normal offset, because inside the loop we use the '_getDataPackageByteSize' function, which receives calldataNegativeOffset as an argument, so we need to maintain it anyway.*

```
31 uint256 timestampOffset = msg.data.length - timestampNegativeOffset;
```

CVF-14. FIXED

- **Category** Procedural
- **Source** PriceFeedBase.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special about this particular version. Also relevant for: PriceFeedsAdapterBase.sol, PriceFeedsAdapterWithoutRounds.sol, PriceFeedsAdapterWithRounds.sol, PriceFeedWithoutRounds.sol, PriceFeedWithRounds.sol, RedstoneAdapterBase.sol, SinglePriceFeedAdapter.sol.

Client Comment *We use custom errors, that are available starting from 0.8.4: Custom Errors But we'll actually start requiring ^0.8.14, because of the following bug in evm assembly solidity optimiser in 0.8.13: Inline Assembly memory side effects bug and Overly Optimistic optimizer certora bug*

```
2 pragma solidity ^0.8.4;
```

CVF-15. INFO

- **Category** Procedural
- **Source** PriceFeedBase.sol

Description We didn't review this file.

```
5 import "../core/IRedstoneAdapter.sol";
  import "./interfaces/IPriceFeed.sol";
```

CVF-16. FIXED

- **Category** Procedural
- **Source** PriceFeedBase.sol

Recommendation It's a good practice to put a comment into an empty block to explain why the block is empty.

```
9 function initialize() public initializer {}
```

CVF-17. FIXED

- **Category** Documentation
- **Source** PriceFeedBase.sol

Description The difference between “roundId” and “answeredInRound” is unclear.

Recommendation Consider documenting.

```
38 answeredInRound = roundId;
```

CVF-18. FIXED

- **Category** Procedural
- **Source** PriceFeedsAdapterBase.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
9 function initialize() public initializer {}
```

CVF-19. FIXED

- **Category** Suboptimal
- **Source** PriceFeedsAdapterBase.sol

Description This variable is redundant, as its value is used only once.

Recommendation Consider removing it.

```
18 bytes32 dataFeedId = dataFeedsIdsArray[i];
```

CVF-20. INFO

- **Category** Procedural
- **Source** PriceFeedsAdapterWithoutRounds.sol

Description Solidity compiler is smart enough to calculate constant hash expressions at compile time.

Recommendation Consider specifying the value as a hash expression rather than a hardcoded value.

Client Comment *It would cause errors if we try to use this constant in evm assembly. Only direct number constants and references to such constants are supported by inline assembly.*

```
8  bytes32 constant VALUES_MAPPING_STORAGE_LOCATION =
   0x4dd0c77efa6f6d590c97573d8c70b714546e7311202ff7c11c484cc841d91bfc;
      ↪ // keccak256("RedStone.oracleValuesMapping");
```

CVF-21. INFO

- **Category** Procedural
- **Source** PriceFeedsAdapterWithRounds.sol

Description Solidity compiler is smart enough to calculate constant hash expressions at compile time.

Recommendation Consider specifying the value as a hash expression rather than a hardcoded value.

Client Comment *It would cause errors if we try to use this constant in evm assembly. Only direct number constants and references to such constants are supported by inline assembly.*

```
8  bytes32 constant VALUES_MAPPING_STORAGE_LOCATION =
   0x4dd0c77efa6f6d590c97573d8c70b714546e7311202ff7c11c484cc841d91bfc;
      ↪ // keccak256("RedStone.oracleValuesMapping");
10 bytes32 constant ROUND_TIMESTAMPS_MAPPING_STORAGE_LOCATION =
   0x207e00944d909d1224f0c253d58489121d736649f8393199f55eecf4f0cf3eb0;
      ↪ // keccak256("RedStone.roundTimestampMapping");
   bytes32 constant LATEST_ROUND_ID_STORAGE_LOCATION =
   0xc68d7f1ee07d8668991a8951e720010c9d44c2f11c06b5cac61fbc4083263938;
      ↪ // keccak256("RedStone.latestRoundId");
```

CVF-22. FIXED

- **Category** Suboptimal
- **Source** PriceFeedWithRounds.sol

Recommendation Consider using safe conversion here.

```
14 return uint80(getPriceFeedAdapterWithRounds().getLatestRoundId());
```

CVF-23. INFO

- **Category** Procedural
- **Source** RedstoneAdapterBase.sol

Description We didn't review this file.

```
6 import "./IRedstoneAdapter.sol";
```

CVF-24. INFO

- **Category** Procedural
- **Source** RedstoneAdapterBase.sol

Description Solidity compiler is smart enough to calculate constant hash expressions at compile time.

Recommendation Consider specifying the value as a hash expression rather than a hardcoded value.

Client Comment *It would cause errors if we try to use this constant in evm assembly. Only direct number constants and references to such constants are supported by inline assembly.*

```
25 bytes32 internal constant LATEST_UPDATE_TIMESTAMPS_STORAGE_LOCATION
   ↪ =
0x3d01e4d77237ea0f771f1786da4d4ff757fcba6a92933aa53b1dcef2d6bd6fe2;
   ↪ // keccak256("RedStone.lastUpdateTimestamp");
```

CVF-25. FIXED

- **Category** Procedural
- **Source** RedstoneAdapterBase.sol

Recommendation It's a good practice to put a comment into an empty block to explain why the block is empty.

```
48 function requireAuthorisedUpdater(address updater) public view
     ↪ virtual {}
```

CVF-26. FIXED

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

Description This function should probably be made pure virtual as its implementation doesn't look real.

```
50 function getDataFeedIds() public view virtual returns (bytes32[]
     ↪ memory) {
```

CVF-27. FIXED

- **Category** Procedural
- **Source** RedstoneAdapterBase.sol

Description The function "getBlockTimestamp" is called several times.

Recommendation Consider calling once and reusing the returned value.

```
131 if (getBlockTimestamp() < receivedTimestampSeconds) {
      if ((receivedTimestampSeconds - getBlockTimestamp()) >
         ↪ maxDataAheadSeconds) {
        revert RedstoneDefaultsLib.TimestampFromTooLongFuture(
           ↪ receivedTimestampSeconds, getBlockTimestamp());
```

```
135 } else if ((getBlockTimestamp() - receivedTimestampSeconds) >
       ↪ maxDataDelaySeconds) {
      revert RedstoneDefaultsLib.TimestampIsTooOld(
         ↪ receivedTimestampSeconds, getBlockTimestamp());
```

CVF-28. FIXED

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

Description Terms “first” and “last” here are ambiguous.

Recommendation Consider using “higher“ and “lower” or “left” and “right” instead.

```
181 dataTimestamp = uint128(packedTimestamps >> 128); // first 128 bits
    blockTimestamp = uint128(packedTimestamps); // last 128 bits
```

CVF-29. INFO

- **Category** Procedural
- **Source** SinglePriceFeedAdapter.sol

Description Solidity compiler is smart enough to calculate constant hash expressions at compile time.

Recommendation Consider specifying the value as a hash expression rather than a hardcoded value.

Client Comment *It would cause errors if we try to use this constant in evm assembly. Only direct number constants and references to such constants are supported by inline assembly.*

```
9  bytes32 internal constant DATA_FROM_LATEST_UPDATE_STORAGE_LOCATION =
10 0x632f4a585e47073d66129e9ebce395c9b39d8a1fc5b15d4d7df2e462fb1fccfa;
      ↪ // keccak256("RedStone.singlePriceFeedAdapter");
```

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