Report

v. 2.0

Customer

Redstone

Smart Contract Audit

Adapter Update

15th July 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	10.07.24	A. Zveryanskaya	Initial Draft
0.2	11.07.24	A. Zveryanskaya	Minor revision
1.0	11.07.24	A. Zveryanskaya	Release
1.1	15.07.24	A. Zveryanskaya	CVF-10 typo fixed, CVF-17 removed
2.0	15.07.24	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’re asked to review the code in the separate files:

```
/
```

MultiFeedAdapter WithoutRounds.sol	PriceFeedWithout RoundsForMulti FeedAdapter.sol	RedstoneConsumer Base.sol

The fixes were provided in commit 8a60632.

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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 1 critical, and a few less important issues. All identified Critical issues have been fixed.

Fixed 1 out of 1 issues

6 Critical Issues

CVF-1 FIXED

- **Category** Flaw
- **Source** MultiFeedAdapterWithoutRounds.sol

Recommendation In this branch "dataFeed.isValueBigger" should be set to "false". With current implementation, if "newValue" fits into 152 bits, but the previous value didn't fit, then the previous value will be used even after update.

```
115 if (newValue <= type(uint152).max) {
      dataFeed.smallerValue = uint152(newValue);
```

7 Moderate Issues

CVF-2 INFO

- **Category** Procedural
- **Source** MultiFeedAdapterWithoutRounds.sol

Recommendation The "dataFeedId" parameters should be indexed.

Client Comment *We do not really need indexing of this param, but having it unindexed helps us to save some gas.*

```
38 event ValueUpdate(uint256 value, bytes32 dataFeedId, uint256
      ↪ updatedAt);
   event UpdateSkipDueToBlockTimestamp(bytes32 dataFeedId);
40 event UpdateSkipDueToDataTimestamp(bytes32 dataFeedId);
   event UpdateSkipDueToInvalidValue(bytes32 dataFeedId);
```

CVF-3 FIXED

- **Category** Suboptimal
- **Source** MultiFeedAdapterWithoutRounds.sol

Description These assignments all update the same storage slot.

Recommendation Consider merging them into a single storage write operation.

Client Comment *Added a nested struct (also for readbility), but after experimenting with different versions we noticed that our solidity compiler (with 10k steps of optimisation) comes to the same gas costs as a very pure implementation with single sstore and sload operations (even without structs).*

```
112 dataFeed.dataTimestamp = uint48(dataTimestamp);
    dataFeed.blockTimestamp = uint48(block.timestamp);
```

```
116   dataFeed.smallerValue = uint152(newValue);
```

```
118   dataFeed.isValueBigger = true;
```

CVF-4 INFO

- **Category** Suboptimal
- **Source** MultiFeedAdapterWithoutRounds.sol

Description These expressions all read the same storage slot.

Recommendation Consider reading the slot once.

Client Comment *It doesn't change the gas costs, but makes the code a bit more compilcated.*

```
133 lastDataTimestamp = dataFeed.dataTimestamp;
    lastBlockTimestamp = dataFeed.blockTimestamp;
```

```
136 if (dataFeed.isValueBigger) {
```

```
139   lastValue = dataFeed.smallerValue;
```

CVF-5 FIXED

- **Category** Unclear behavior
- **Source** RedstoneConsumerBase.sol (ONLY YELLOW)

Recommendation These calls should return the updated "calldataNegativeOffset" values, so the caller code wouldn't need to known how many bytes were consumed.

```
110 uint256 dataPackagesCount = _extractDataPackagesCountFromCalldata(
      ↪ calldataNegativeOffset);
```

```
122   (uint256 dataPackageByteSize, uint256 dataPackageTimestamp) =
        ↪ _extractDataPackage(
        dataFeedIds,
        uniqueSignerCountForDataFeedIds,
        signersBitmapForDataFeedIds,
        valuesForDataFeeds,
        calldataNegativeOffset
      );
```

8 Minor Issues

CVF-6 INFO

- **Category** Procedural
- **Source** MultiFeedAdapterWithoutRounds.sol

Recommendation Consider specifying as "^0.8.0" unless there is something special regarding this particular version. Also relevant for: PriceFeedWithoutRoundsForMultiFeedAdapter.sol

Client Comment *We use 0.8.14 because we need custom errors (supported from 0.8.4), but we would like to avoid effects of the bug in the version 0.8.13: https://github.com/ethereum/solidity-blog/blob/499ab8abc19391be7b7b34f88953a067029a5b45/_posts/2022-06-15-inline-assembly-memory-side-effects-bug.md*

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

CVF-7 INFO

- **Category** Procedural
- **Source** MultiFeedAdapterWithoutRounds.sol

Description We didn't review these files.

```
6 import {IMultiFeedAdapter} from "../interfaces/IMultiFeedAdapter.sol
    ↪ ";
  import {IPriceCalculator} from "../../custom-integrations/layerbank/
    ↪ IPriceCalculator.sol";
  import {ILToken} from "../../custom-integrations/layerbank/ILToken.
    ↪ sol";
```

CVF-8 INFO

- **Category** Procedural
- **Source** MultiFeedAdapterWithoutRounds.sol

Description We didn't review the implemented interfaces.

```
28 abstract contract MultiFeedAdapterWithoutRounds is
   ↪ RedstoneConsumerNumericBase, IMultiFeedAdapter,
   ↪ IPriceCalculator {
```

CVF-9 INFO

- **Category** Suboptimal
- **Source** MultiFeedAdapterWithoutRounds.sol

Description Solidity compiler is smart enough to precompute hash expressions.

Recommendation We can not use this pre-compiled hash in the assembly blocks

Client Comment *We use this approach in multiple other contracts and we'd like to stay consistent.*

```
29 bytes32 internal constant DATA_FEEDS_STORAGE_LOCATION = 0
   ↪ xf2fa0a38bedf014b27fa765f1a093191b7bb31b38e723aa3f28037722d5b1658
   ↪ ; // keccak256("RedStone.dataFeeds");
```

CVF-10 INFO

- **Category** Suboptimal
- **Source** MultiFeedAdapterWithoutRounds.sol

Recommendation The argument is redundant, as its value always equals to "msg.sender".

Client Comment *We've removed the function, because we want to have more specific logic, which will be located in the _validateBlockTimestamp function. We want to whitelist specific addresses only if less than X seconds haven't passed since the latest update. Otherwise we want to allow anyone to update data.*

```
55 function _requireAuthorisedUpdater(address updater) internal view
   ↪ virtual {
```

CVF-11 FIXED

- **Category** Documentation
- **Source** MultiFeedAdapterWithoutRounds.sol

Recommendation The function mentioned in this comment isn't actually used in the code, so the comment seems incorrect.

```
63 // The getOracleNumericValuesFromTxMsgWithTimestamp function
   ↪ performs many checks
```

CVF-12 INFO

- **Category** Procedural
- **Source** MultiFeedAdapterWithoutRounds.sol

Description The dollar sign as an identifier looks very unusual in Solidity code and could even be incompatible with certain tools.

Recommendation Consider not using the dollar sign in identifiers.

Client Comment *We used the style similar to the openzeppelin contract, where we've noticed the trick with storage location for structs: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/AccessControlUpgradeable.sol*

```
143 function _getDataFeedsStorage() private pure returns (
    ↪ DataFeedsStorage storage $) {
```

CVF-13 FIXED

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

Description This statement actually does nothing.

```
190 return values;
```

CVF-14 FIXED

- **Category** Documentation
- **Source** MultiFeedAdapterWithoutRounds.sol

Description This comment is misleading, as one could think that "20 bytes" means "bytes20", and in Solidity "bytes20" values are right-padded with zeros when represented as a 32-bytes word. However, this function produces left-padded words by first casting as address to "unit160" (which is left-padded), then to "uint256" (preserving left padding) and then to "bytes32".

Client Comment *Updated the comment.*

```
210 // By default, users will be able to use data feed identifiers (
    ↪ casted to 20 bytes) in layerbank functions
```

CVF-15 INFO

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

Recommendation The argument type should be "ILToken".

Client Comment *It would cause compilation error, because of the interface.*

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

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

CVF-16 INFO

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

Recommendation The return type should be more specific.

Client Comment *It would cause compilation error, because of the interface.*

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

```
234   address[] memory assets
```

CVF-18 INFO

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

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

Client Comment *It would cause compilation error, because of the interface.*

```
248 address[] memory gTokens
```

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