AUDITONE

Audit Report

RedStone

07.06.2023

Table of Contents

CONTENTS

INTRODUCTION

Smart Contract Security Analysis Report

Note: This report may contain sensitive information on potential vulnerabilities and exploitation methods. This must be referred internally and should be only made available to the public after issues are resolved (to be confirmed prior by the client and AuditOne).

INTRODUCTION

Csanuragjain, Defsec and Gracious, who are auditors at AuditOne, successfully audited the smart contracts (as indicated below) of Redstone. The audit has been performed using manual analysis. This report presents all the findings regarding the audit performed on the customer's smart contracts. The report outlines how potential security risks are evaluated. Recommendations on quality assurance and security standards are provided in the report.

01-PROJECT DESCRIPTION

RedStone builds a novel oracle that delivers price feeds for over 1,100 assets, available on Ethereum, Polygon, Avalanche, Arbitrum, and 30+ chains with 10-second update time.

Get price data feeds covering long tail tokens, Forex, commodities, on-chain identity, and much more, in a truly scalable and cross-chain manner. The novel product design ensures resistance to network congestion and enables the delivery of data to projects every few seconds.

Moreover, projects can integrate RedStone Oracles once and use it on any L1 or L2 chain they would like to expand to, thanks to our storageless architecture (data is transferred with meta transactions which bypasses expensive EVM-storage).

The company is backed by some of the most renowned Web3 Entrepreneurs such as Stani Kulechov (Aave), Sandeep Nailwal (Polygon), Alex Gluchovski (zkSync) and many more. In Aug 2022 RedStone raised their $7M Seed Round led by Lemniscap with participation from Coinbase Ventures, Blockchain Capital, Arweave, and others.

02-Project and Audit Information

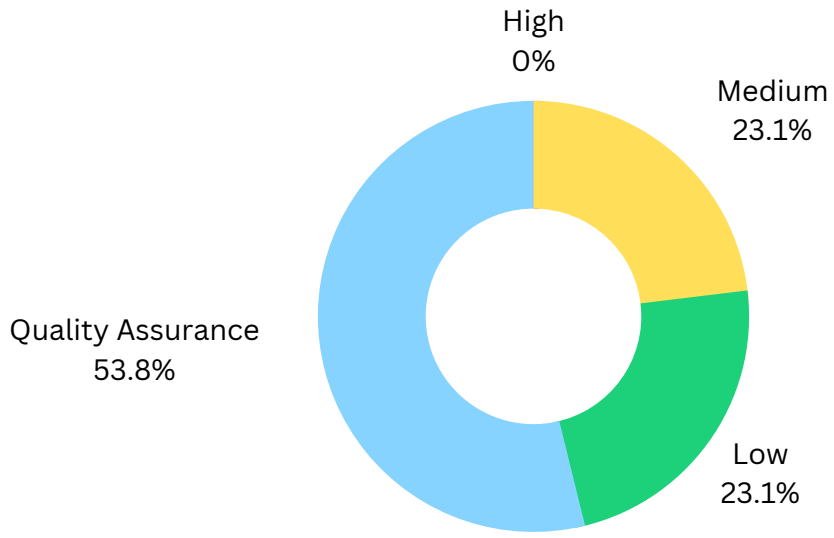
Term	Description
Auditor	Csanuragjain, Defsec and Gracious
Reviewed by	Raja Thota
Type	Oracle
Language	Solidity
Ecosystem	Avalanche, Arbitrum, Ethereum, and Celo
Methods	Manual Review
Repository	https://github.com/redstone-finance/redstone-oracles-monorepo
Commit hash (at audit start)	3f466fb7d6b74acc4fd94457c6aca666f0410bc5
Commit hash (after resolution)	f2cfb6490dbfe53b752f721a052687e6ae97d78a
Documentation	https://docs.redstone.finance/docs/introduction
Website	https://redstone.finance/
Submission Time	25-04-2023
Finishing Time	07-06-2023

03-Contracts in Scope

- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/core/RedstoneAdapterBase.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/PriceFeedsAdapterBase.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/PriceFeedBase.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/with-rounds/PriceFeedsAdapterWithRounds.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/with-rounds/PriceFeedWithRounds.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedWithoutRounds.sol
- https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedsAdapterWithoutRounds.sol

04-Executive summary

Redstone plugin's smart contracts were audited between 2022-12-06 and 2023-03-14 by Csanuragjain, Defsec and Gracious. Manual analysis was carried out on the code base provided by the client. The following findings were reported to the client. For more details, refer to the findings section of the report.



Issue Category	Issues Found	Resolved	Acknowledged
High	0	0	0
Medium	3	3	0
Low	3	3	0
Quality Assurance	7	5	2

05-Severity Definitions

Risk factor matrix	Low	Medium	High
Occasional	L	M	H
Probable	L	M	H
Frequent	M	H	H

High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

Medium: The impact of medium issues is less critical than high but still probable with considerable damage. The protocol or its availability could be impacted or leak value with a hypothetical attack path with stated assumptions.

Low: Low issues impose a small risk on the project. Although the impact is not estimated to be significant, we recommend fixing them on a long-term horizon. Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

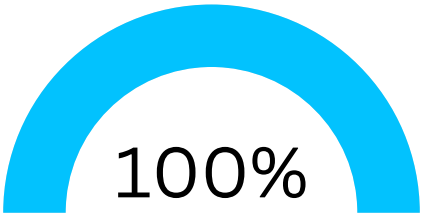
Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events, etc.)

Occasional: This category might represent risks with a moderate chance of occurring. These risks are not common but have happened in similar situations.

Probable: This category represents risks that are likely to happen. There's a high probability based on past experiences, current conditions, or future projections.

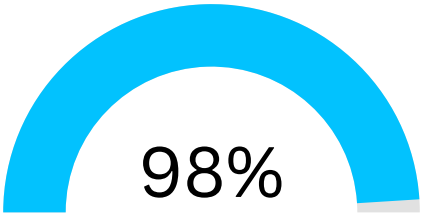
Frequent: This category represents risks that occur regularly. In a security audit, this might refer to common vulnerabilities or threats consistently observed in similar systems or environments.

06-Audit Overview



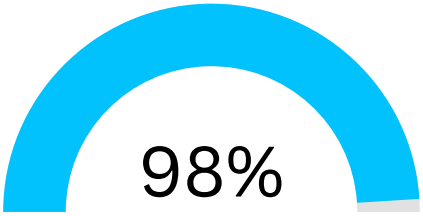
Security score

Security score is a numerical value generated based on the vulnerabilities in smart contracts. The score indicates the contract's security level and a higher score implies a lower risk of vulnerability.



Code quality

Code quality refers to adherence to standard practices, guidelines, and conventions when writing computer code. A high-quality codebase is easy to understand, maintain, and extend, while a low-quality codebase is hard to read and modify.



Documentation quality

Documentation quality refers to the accuracy, completeness, and clarity of the documentation accompanying the code. High-quality documentation helps auditors to understand business logic in code well, while low-quality documentation can lead to confusion and mistakes.

07-Findings

Finding: #1

Issue: The minimum interval between updates is short and poses risk of front-running attack

Severity: Medium

Where: https://github.com/redstone-finance/redstone-oracles-monorepo/blob/e19d97d0f3bb5d93a59af7a115da42908c2b9777/packages/on-chain-relayer/contracts/core/RedstoneAdapterBase.sol#L26

Impact: An attacker can exploit the time gap between transactions to gain an advantage or manipulate the contract's state.

Description: The **MIN_INTERVAL_BETWEEN_UPDATES** constant, as its name suggests, represents the minimum amount of time that must pass between updates. In this case, a 3-second interval has been set, meaning that the updates cannot occur more frequently than once every 3 seconds.
This suggests that a 3-second interval is considered low, which means that it may not provide enough time for transactions to be processed before a new update occurs. This can create a vulnerability known as front-running, where an attacker can exploit the time gap between transactions to gain an advantage or manipulate the contract's state.

Recommendations:
To mitigate this risk, it is necessary to set a higher minimum interval between updates or implement other additional security measures, to prevent front-running attacks.

Status: Resolved. Project team clarified that the MIN_INTERVAL_BETWEEN_UPDATES constant is only used in the **getMinIntervalBetweenUpdates** function. And that the **getMinIntervalBetweenUpdates** function is virtual and will be overridden to specify custom min interval between updates.

Finding: #2

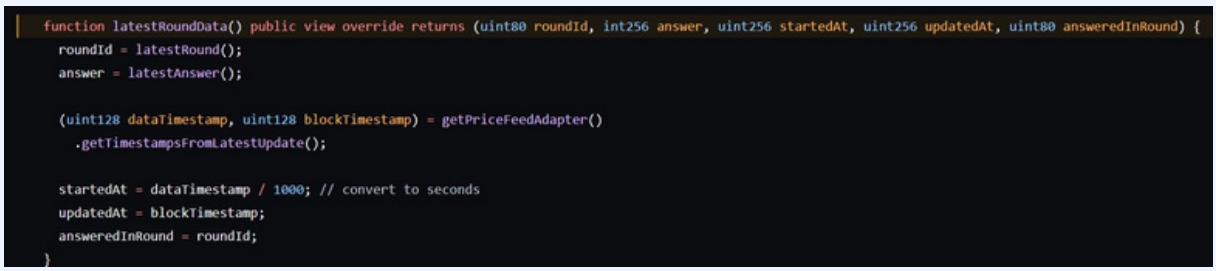
Issue: Stale Prices could be returned

Severity: Medium

Where: https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/PriceFeedBase.sol#L29
https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/price-feeds/without-rounds/PriceFeedWithoutRounds.sol#L15

Impact: Old stale prices could be returned which on using could lead to losses.

Description: It was observed that there is no check to validate that price feed timestamp (in this case blockTimestamp) is recent. If price feed was not updated recently then old prices would be returned by **latestRoundData** function.



```
function latestRoundData() public view override returns (uint80 roundId, int256 answer, uint256 startedAt, uint256 updatedAt, uint80 answeredInRound) {
  roundId = latestRound();
  answer = latestAnswer();

  (uint128 dataTimestamp, uint128 blockTimestamp) = getPriceFeedAdapter()
    .getTimestampsFromLatestUpdate();

  startedAt = dataTimestamp / 1000; // convert to seconds
  updatedAt = blockTimestamp;
  answeredInRound = roundId;
}
```

Recommendations:
Validate that **blockTimestamp** is recent. Something similar to **_assertMinIntervalBetweenUpdatesPassed** at RedstoneAdapterBase.sol#L102 could be helpful.

Status: Resolved.
Project team stated that one of the conditions for triggering value updates is the deviation in value. For instance, if the current market price of ETH deviates by 0.5% from the latest value saved in the contract, they may consider updating the ETH price feed. This approach allows them to have flexibility in determining the maximum time interval between updates. Additionally, the team believes that it should be the responsibility of consumer contracts to check the difference between the latest update block and the current block, if they choose to do so.

Finding: #3

Issue: Long Delays Can Cause Sandwich Attacks

Severity: Medium

Where: RedstoneAdapterBase.sol#LL126C1-L143C4

```
function validateDataPackagesTimestampOnce(uint256 dataPackagesTimestamp) public view virtual {
  uint256 receivedTimestampSeconds = dataPackagesTimestamp / 1000;

  (uint256 maxDataAheadSeconds, uint256 maxDataDelaySeconds) = getAllowedTimestampDiffsInSeconds();

  if (block.timestamp < receivedTimestampSeconds) {
    if ((receivedTimestampSeconds - block.timestamp) > maxDataAheadSeconds) {
      revert RedstoneDefaultsLib.TimestampFromTooLongFuture(receivedTimestampSeconds, block.timestamp);
    }
  } else if ((block.timestamp - receivedTimestampSeconds) > maxDataDelaySeconds) {
    revert RedstoneDefaultsLib.TimestampIsTooOld(receivedTimestampSeconds, block.timestamp);
  }
}

function getAllowedTimestampDiffsInSeconds() public view virtual returns (uint256 maxDataAheadSeconds, uint256 maxDataDelaySeconds) {
  maxDataAheadSeconds = RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_AHEAD_SECONDS;
  maxDataDelaySeconds = RedstoneDefaultsLib.DEFAULT_MAX_DATA_TIMESTAMP_DELAY_SECONDS;
}
```

Impact: Allowing long delays in data package timestamps can expose the contract to the following risks:

- **Price Manipulation**: An attacker can take advantage of the delayed data to manipulate the price or other values within the contract, potentially affecting users and other contracts relying on the data.
- **Front-Running**: Attackers can monitor pending transactions and exploit the outdated data to place their transactions with more favorable conditions, causing potential financial loss to users.

Description: The **validateDataPackagesTimestampOnce()** function in the given contract validates the timestamp of data packages to ensure that they are not too far in the future or too old. However, long delays between the actual data timestamp and the current block timestamp can lead to potential sandwich attacks, where an attacker can manipulate the order of transactions in a block to their advantage.

Recommendations: To mitigate the risk of sandwich attacks due to long delays in data package timestamps, consider implementing the following changes:
Adjust the **maxDataDelaySeconds** value in the **getAllowedTimestampDiffsInSeconds()** function to a shorter time frame, reducing the allowed delay between the data package timestamp and the current block timestamp.

Status: Resolved. Project team introduced several checks (e.g. to disallow too frequent updates and to disallow saving in adapter contract storage prices older than were saved before).

Finding: #4

Issue: Packed timestamp value could change

Severity: Low

Where: https://github.com/redstone-finance/redstone-oracles-monorepo/blob/483-implement-a-roundless-and-a-roundful-version-of-on-chain-relayer-contracts/packages/on-chain-relayer/contracts/core/RedstoneAdapterBase.sol#L179

Impact: Packed timestamp will be incorrectly saved. However it requires privilleged user to use a value>uint128 for dataPackagesTimestamp which is unlikely.

Description: When value of **dataPackagesTimestamp** exceeds uint128, then **packTwoNumbers** will not work properly (since it is expecting **dataPackagesTimestamp** to be max uint128). Due to this a different value of **dataPackagesTimestamp** will be retrieved when **_unpackTimestamps** is used.

```
function _unpackTimestamps(uint256 packedTimestamps) internal pure returns (uint128 dataTimestamp, uint128 blockTimestamp) {
  dataTimestamp = uint128(packedTimestamps >> 128); // first 128 bits
  blockTimestamp = uint128(packedTimestamps); // last 128 bits
}

function _saveTimestampsOfCurrentUpdate(uint256 dataPackagesTimestamp) internal virtual {
  uint256 blockTimestamp = block.timestamp;
  assembly {
    let timestamps := packTwoNumbers(dataPackagesTimestamp, blockTimestamp)
    sstore(LATEST_UPDATE_TIMESTAMPS_STORAGE_LOCATION, timestamps)

    function packTwoNumbers(num1, num2) -> resultNumber {
      resultNumber := or(shl(BITS_COUNT_IN_16_BYTES, num1), num2)
    }
  }
}
```

Recommendations: dataPackagesTimestamp should be uint128 instead of uint256.

Status: This was resolved by adding the corresponding assertions for both block timestamp and data package timestamp.

Finding: #5

Issue: Inline assembly bug in solidity 0.8.13.

Severity: Low

Where: RedstoneConsumerBase.sol#L190

Impact: There is a known medium severity bug which affects memory writing in Solidity version 0.8.13. It would remove unused writes to memory and storage.

Description: There is a known medium severity bug which affects memory writing in Solidity version 0.8.13. It would remove unused writes to memory and storage.

- https://github.com/ethereum/solidity-blog/blob/499ab8abc19391be7b7b34f88953a067029a5b45/_posts/2022-06-15-inline-assembly-memory-side-effects-bug.md
- https://medium.com/certora/overly-optimistic-optimizer-certora-bug-disclosure-2101e3f7994d

Since pragmas are floating in the contracts. the contracts may suffer from the issue. For example, the following contract has the issue. It only uses **mstore** in the inline assembly block. So it will be removed in Solidity 0.8.13.

Recommendations: Use **>= 0.8.14** instead of **^0.8.4**.

Status: Resolved. Project team switched to **^0.8.14.**

Finding: #6

Issue: Potential Gas Problem in **validateAndUpdateDataFeedsValues()**.

Severity: Low

Where: [PriceFeedsAdapterBase.sol#L13]

```
function validateAndUpdateDataFeedsValues(
  bytes32[] memory dataFeedsIdsArray,
  uint256[] memory values
) internal virtual override {
  for (uint256 i = 0; i < dataFeedsIdsArray.length; i++) {
    bytes32 dataFeedId = dataFeedsIdsArray[i];
    _updateDataFeedValue(dataFeedId, values[i]);
  }
}
```

Impact: High gas costs and out-of-gas issues can lead to the following problems:

- **Transaction Failures:** Transactions calling the **validateAndUpdateDataFeedsValues()** function may fail due to insufficient gas, causing potential disruption to the contract's operation.

Description: The **validateAndUpdateDataFeedsValues()** function iterates through the **dataFeedsIdsArray** and values arrays and calls the **_updateDataFeedValue()** function for each element. The gas cost of this function will increase linearly with the number of elements in the arrays, potentially leading to high gas costs and out-of-gas issues if the arrays become too large.

Recommendations: To mitigate the potential impact of high gas costs and out-of-gas scenarios in the **validateAndUpdateDataFeedsValues()** function, consider implementing the following changes:

- **Limit the Size of Input Arrays:** Add a check to ensure that the length of the **dataFeedsIdsArray** and values arrays does not exceed a predefined maximum limit.

Status: Resolved. Project team stated that developers who will deploy **PriceFeeds** will manually specify the supported data feeds, so the arrays will never be long.

Finding: #7

Issue: Consider a higher Solidity version

Severity: Quality Assurance

Where: Whole project

Impact: Contract could be susceptible to attacks or hacks, which could be prevented in a higher version.

Description: When a new version of Solidity is released, it typically includes security updates and bug fixes, which can make your contract less vulnerable to attacks from hackers or malicious actors. Additionally, new versions of Solidity can also introduce new language features and improve existing ones, making it easier for developers to write efficient and safe code.

Recommendations: It is recommended to use a higher solidity version primarily for security purposes.

Status: Resolved. Solidity version was switched to 0.8.14.

Finding: #8

Issue: Missing/Insufficient Natspec comments

Severity: Quality Assurance

Where: Whole project

Impact: Natspec issues can lead to misinterpretations, errors, and vulnerabilities in the code, which can have significant negative consequences for the security and functionality of the smart contract.

Description: There are cases of missing or insufficient Natspec comments in the code.

Recommendations: Consider including Natspec comments in the contract code, explaining its purpose and functionality.

Status: Resolved.

Finding: #9

Issue: Loop optimization

Severity: Quality Assurance

Where: Each **for-loop** in the project

Impact: Save gas for each loop interaction

Description: When a loop iterates many times, it causes the amount of gas required to execute the function to increase significantly. In Solidity, excessive looping can cause a function to use more than the maximum allowed gas, which causes the function to fail.
For instance, the **validateAndUpdateDataFeedsValues()** function in the **PriceFeedsAdapterWithRounds.sol** contract loops through an array of data feed values and updates each value in the contract storage. If the number of data feeds to update is too large, it may consume too much gas and exceed the block gas limit, causing the transaction to fail. This could be exploited by an attacker to perform a denial-of-service attack on the contract by sending a transaction with a large array of data feeds to update, causing legitimate transactions to fail.

Recommendations: To reduce gas consumption, it's recommended to find ways to optimize the loop or potentially break the loop into smaller batches. The following pattern can also be used:

```
uint256 cachedLen = array.length;
for(uint i; i < cachedLen;){

  unchecked {
    ++i;
  }
}
```

Status: Resolved. Unchecked blocks were added.

Finding: #10

Issue: Import declarations should import specific identifiers, rather than the whole file

Severity: Quality Assurance

Where: All Contracts

Impact: Optimization

Description: Using import declarations of the form **import {<identifier_name>} from "some/file.sol"** avoids polluting the symbol namespace making flattened files smaller, and speeds up compilation.

Recommendations: Consider using import declarations of the form **import {<identifier_name>} from "some/file.sol".**

Status: Resolved.

Finding: #11

Issue: Redundant Usage of SafeMath Library in Solidity ^0.8.4

Severity: Quality Assurance

Where: RedstoneConsumerBase.sol#L20

Impact: Redundant file

Description: The **RedstoneConsumerBase** contract imports and uses the SafeMath library for performing arithmetic operations. However, as of Solidity version **0.8.0**, the language has built-in overflow and underflow protection for arithmetic operations, rendering the **SafeMath** library redundant and unnecessary. Using the **SafeMath** library in a contract with Solidity version **^0.8.4** increases gas costs, introduces additional complexity, and may lead to confusion among developers who are already familiar with the updated Solidity features.

```
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";

import "./RedstoneConstants.sol";
import "./RedstoneDefaultsLib.sol";
import "./CalldataExtractor.sol";
import "../libs/BitmapLib.sol";
import "../libs/SignatureLib.sol";

/**
 * @title The base contract with the main Redstone logic
 * @author The Redstone Oracles team
 * @dev Do not use this contract directly in consumer contracts, take a
 * look at `RedstoneConsumerNumericBase` and `RedstoneConsumerBytesBase` instead
 */
abstract contract RedstoneConsumerBase is CalldataExtractor {
  using SafeMath for uint256;
}
```

Recommendations: To address this issue, it is recommended to remove the import and usage of the SafeMath library from the RedstoneConsumerBase contract, as the built-in arithmetic checks in Solidity ^0.8.4 are sufficient.

Status: Acknowledged

Finding: #12

Issue: Typo in the comment

Severity: Quality Assurance

Where: PriceFeedWithoutRounds.sol#L10

Impact: While this typo does not directly impact the functionality of the contract, it can cause confusion for developers who are trying to understand the code. Clear and accurate comments are essential for maintaining a high-quality codebase and ensuring smooth collaboration among developers.

Description: There is a typo in the comment within the **PriceFeedWithoutRounds** contract. The comment states "**but still rely on getRoundData or latestRounud functions**", where the correct term should be **latestRound**.

```
// There are possible use cases that some contracts don't need values from old rounds
// but still rely on `getRoundData` or `latestRounud` functions
```

Recommendations: To address this issue, it is recommended to correct the typo in the comment.

Status: Resolved.

Finding: #13

Issue: Missing event logging and emission

Severity: Quality Assurance

Where: Whole project

Impact: It makes it difficult to monitor and reflect critical state changes.

Description: None of the functions in the contracts log nor emit any events to notify external actors of changes to its state, which makes it harder to monitor and detect potential exploits or attacks.

Recommendations: Consider logging events and adding event emissions to functions.

Status: Acknowledged.

08 - Disclaimer

The smart contracts provided to AuditOne have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The ethical nature of the project is not guaranteed by a technical audit of the smart contract. Any owner-controlled functions should be carried out by the responsible owner. Before participating in the project, all investors/users are recommended to conduct due research.

The focus of our assessment was limited to the code parts associated with the items defined in the scope. We draw attention to the fact that due to inherent limitations in any software development process and 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 cannot be free from any errors or failures. These preconditions can impact 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 correctly executing the included third-party technology stack itself. Report readers should also consider that over the life cycle of any software product, changes to the product itself or the environment in which it is operated can have an impact leading to operational behaviors other than initially determined in the business specification.

Contact

auditone.io

@auditone_team

hello@auditone.io

A trust layer of our multi-stakeholder world.