PUBLIC

# Code Assessment

of the SparkRewards Smart Contracts

February 25, 2025

Produced for

Spark

by

CHAINSECURITY

# Contents

# 1 Executive Summary

Dear all,

Thank you for trusting us to help SparkDAO with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of SparkRewards according to Scope to support you in forming an opinion on their security risks.

Spark implements the SparkRewards smart contract to distribute ERC-20 tokens based on a Merkle tree root and epochs.

The most critical subjects covered in our audit are functional correctness, access control and merkle proof verification. Security regarding all the aforementioned subjects is high.

The general subjects covered are gas efficiency and proper documentation.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the SparkRewards repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 10 Feb 2025 | 00478a8c9364124a80cc73f4e12d7acc8ace7a36 | Initial Version |
| 2 | 25 Feb 2025 | d99f9fa227000d97c89209d83b407628cc1a82ab | After Intermediate Report |

For the solidity smart contracts, the compiler version `0.8.21` was chosen.

The following files were in scope:

```
src/SparkRewards.sol
```

### 2.1.1 *Excluded from scope*

All files not listed above including the tests are out of scope.

## 2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Spark implements the SparkRewards smart contract to distribute ERC-20 tokens based on a Merkle tree root and epochs. Users can claim tokens for a specific epoch and token by providing a valid Merkle proof. Privileged roles can update the Merkle root, the wallet (source of the funds), close epochs, and reopen them.

Users claim using the public `claim()` function:

- Allows anyone to claim funds on behalf of an account. Takes the `epoch`, `account`, `token`, `cumulativeAmount`, `expectedMerkleRoot` and `MerkleProof` as parameters and returns the actually claimed amount.
- Verifies that the `expectedMerkleRoot` is active and the epoch is not closed. Checks the presence of the leaf (based on `epoch`, `account`, `token`, `cumulativeAmount`) and tracks the amount already claimed in this epoch.
- Claims the full delta between the `cumulativeAmount` and the already claimed amount. Transfers funds from the wallet contract to the `account` and emits a `Claimed` event.

Privileged Functions:

`setWallet()`: Allows accounts with the `DEFAULT_ADMIN_ROLE` to update the wallet address. Emits the `WalletUpdated` event.

`setMerkleRoot()`: Allows accounts with the `MERKLE_ROOT_ROLE` to update the Merkle root. Emits the `MerkleRootUpdated` event.

`setEpochClosed()`: Allows accounts with the `EPOCH_ROLE` to update the epoch status. Takes the `epoch` and a boolean `isClosed` and emits the `EpochIsClosed` event.

## 2.3 Trust Model

The following roles are present in this system:

- Admin (`DEFAULT_ADMIN_ROLE`)

Trust Assumption: Fully trusted; has ultimate control over all roles. Worst-Case Scenario: A compromised admin can revoke/grant any role, block the contract by changing the wallet address, and claim all funds by assigning the MERKLE_ROOT_ROLE and updating the merkle root.

- Merkle Root Manager (`MERKLE_ROOT_ROLE`)

Trust Assumption: Trusted to update the Merkle root correctly. Worst-Case Scenario: A compromised role can submit a modified Merkle root, enabling unauthorized claims and draining contract funds. This role can also censor rewards claiming by excluding certain accounts, epochs, and tokens from the Merkle tree.

- Epoch Manager (`EPOCH_ROLE`)

Trust Assumption: Trusted to open or close epochs as expected. Worst-Case Scenario: A compromised role can reopen epochs, allowing users to claim rewards again or close epochs prematurely, blocking legitimate claims.

- Wallet

Trust Assumption: Expected to function as a passive token distributor. Worst-Case Scenario: If the SparkRewards contract does not have the required allowance or the balance is insufficient, legitimate rewards cannot be claimed.

- Users (Untrusted)

Trust Assumption: Untrusted; can only claim rewards.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software 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 itself cannot be free from any error or failures. These preconditions can have an impact on 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 the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |
| Informational Findings | 3 |

- Function Visibility Can Be Restricted to External Code Corrected
- Inaccurate Error Message Code Corrected
- Inconsistent Coding Style Code Corrected

## 6.1 Function Visibility Can Be Restricted to External

Informational Version 1 Code Corrected

*CS-SPRWD-003*

Function `setWallet` and `setEpochClosed` have public visibility. Both of them can be restricted to external, since they are not called internally.

---

**Code corrected:**

The visibility of these functions is now set to external.

## 6.2 Inaccurate Error Message

Informational Version 1 Code Corrected

*CS-SPRWD-001*

In function `claim`, in case the input merkle root does not match the merkle root stored in the SparkRewards contract, it will revert with the message `merkle-root-was-updated`. This error message considers the situation when a user submits a stale merkle root that has been updated, while it does not consider the case when a user submits a wrong merkle root that has never been added.

---

**Code corrected:**

The error message has been changed to `merkle-root-mismatch`.

## 6.3 Inconsistent Coding Style

**Informational** **Version 1** **Code Corrected**

*CS-SPRWD-002*

For the privileged functions `setWallet`, `setMerkleRoot`, `setEpochClosed`, the events are emitted before the state change in the first two functions, while the event is emitted after the state change in the last function.

---

**Code corrected:**

All functions now consistently emit the event before the state change.