Security Audit Report

ICN Fairdrop

v1.0

May 22, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Impossible Cloud Network Foundation to perform a security audit of Audit of the ICN Fairdrop distribution smart contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/ICN-Protocol/icn-fairdrop-distribution-smart-contract
Commit	`b3321aec5c2036b2a4db48a076480fb339fa5f87`
Scope	All contracts were in scope.
Fixes verified at commit	`5183e996ea52c2660db2ae134d38b408fd7a6c80` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The ICN Fairdrop contract manages a token distribution system for ICN tokens with two vesting schedule options.

It implements role-based access control for administrative functions like importing user data and setting activation time.

Users are categorized into three allocation tiers (`800`, `525`, or `250` tokens) with either a default (7-step) or fast (2-step) vesting schedule. Once activated, eligible users can claim their tokens during specific time windows for each vesting step.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	The client provided the documentation of the contract.
Test coverage	**High**	`forge coverage` reports a test coverage of `99.30%`.

Summary of Findings

No	Description	Severity	Status
1	Claim window boundary edge case condition leads to lost claims	Minor	Resolved
2	Misleading event parameter in `VestingFinalized`	Minor	Resolved
3	Centralization risks	Minor	Acknowledged
4	Repeated activation configuration enables retroactive vesting manipulation	Minor	Resolved
5	Incorrect EIP-7201 storage slot pointer	Informational	Resolved
6	ETH sent during the implementation deployment would be stuck	Informational	Resolved
7	Lack of funding validation allows activation without sufficient allocated funds	Informational	Acknowledged
8	Use of magic numbers decreases maintainability	Informational	Resolved
9	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Claim window boundary edge case condition leads to lost claims

Severity: Minor

In `src/ICNFairdropDistribution.sol:160-161`, the `claimStep` function validates eligibility by checking whether `block.timestamp` falls within the inclusive range `[startTime, endTime]`, where `endTime` is calculated as `startTime + MAX_STEP_CLAIM_PERIOD`. This inclusive condition implies that at the block where `block.timestamp` is equal to `endTime`, the current step remains claimable.

However, this creates a scenario where two consecutive steps may simultaneously satisfy the eligibility check. Specifically, if `block.timestamp` is equal to the previous step unlock time plus `MAX_STEP_CLAIM_PERIOD`, both the current step (step = `last`) and the preceding step (step = `last - 1`) pass the time check.

If the last step is claimed first in this edge case, the `userData` is cleared, causing the subsequent step to become unclaimable, even though it was technically eligible, resulting in potential loss of claim rights.

Recommendation

We recommend revising the steps' time boundary logic to enforce non-overlapping claim periods.

Status: Resolved

2. Misleading event parameter in `VestingFinalized`

Severity: Minor

In `src/ICNFairdropDistribution.sol:123-137`, the `finalizeVesting` function emits the `VestingFinalized` event with the contract's full balance at the time of execution.

However, this does not accurately reflect the actual `_amount` parameter, which represents the precise value transferred to the treasury.

As a result, the emitted event can falsely suggest that the entire contract balance was vested, even if only a partial transfer occurred. This discrepancy may mislead off-chain event log analyses and could result in misinterpretations.

Recommendation

We recommend modifying the `finalizeVesting` function to ensure the `VestingFinalized` event emits the exact `_amount` value that is transferred to the treasury.

Status: Resolved

3. Centralization risks

Severity: Minor

The smart contracts under review rely on specific roles to perform critical administrative actions, creating centralized points of control that may be exploited if misused or compromised. The following roles hold elevated privileges:

- `DEFAULT_ADMIN_ROLE`
    - Can invoke `finalizeVesting` to recover all tokens.
    - Has the authority to grant and revoke all other roles.
- `UPGRADES_OPERATOR_ROLE`
    - Authorized to upgrade the contract implementation, potentially introducing arbitrary logic.
- `UPLOADER_ROLE`
    - Can import and remove user allocations at will.
    - Capable of removing users immediately before `setActivateAt`, preventing user response or mitigation.
- `BUSINESS_OPERATOR_ROLE`
    - Controls the activation of vesting schedules, thereby influencing token distribution timing.

This centralized design grants a small set of entities full operational control over contract behavior and user entitlements, exposing the contract to risks in case the aforementioned accounts are compromised.

Recommendation

We recommend enforcing strict key management and the usage of multi-signature accounts.

Status: Acknowledged

4. Repeated activation configuration enables retroactive vesting manipulation

Severity: Minor

In `src/ICNFairdropDistribution.sol:112-122`, the `setActivateAt` function allows an account holding the `BUSINESS_OPERATOR_ROLE` to set the `dateTime` when the vesting will be activated.

However, this function can be executed multiple times, with subsequent invocations overwriting the existing schedule, including past deadlines.

This retroactive modification may arbitrarily re-enable or disable claims or indefinitely delay them, potentially undermining the trust in the vesting mechanism and misleading participants.

Recommendation

We recommend enforcing immutability of the vesting calendar once it has been activated by preventing any further calls to `setActivateAt` after the initial invocation.

Status: Resolved

5. Incorrect EIP-7201 storage slot pointer

Severity: Informational

In `src/ICNFairdropDistributionStorage.sol:39-41`, the contract utilizes a dedicated storage slot for its main data struct `ICNFairdropDistributionStorageData`, following the EIP-7201 namespaced storage pattern.

However, the constant `ICN_TOKEN_DISTRIBUTION_STORAGE_SLOT` holds a value `(0x9afe....6000)` that does not match the value derived from the standard EIP-7201 calculation using the namespace "`icnFairdropDistribution.storage`".

Recommendation

We recommend changing the value to `0x5ddccec0f95808cd4366a65720a39599b50cad94662220621e68977d11acbf00`.

Status: Resolved

6. ETH sent during the implementation deployment would be stuck

Severity: Informational

In `src/ICNFairdropDistribution.sol:25`, the constructor is marked as `payable`.

While this is a gas optimization that saves approximately 20-30 gas by avoiding EVM checks for ETH transfers, any ETH mistakenly sent during the implementation contract deployment would be permanently locked, as there is no mechanism to retrieve these funds from the implementation contract.

Recommendation

We recommend removing the payable modifier from the constructor unless the minimal gas savings are critical to the deployment process.

Alternatively, if keeping the payable modifier, add a clear comment warning that ETH should not be sent during the implementation deployment.

Status: Resolved

7. Lack of funding validation allows activation without sufficient allocated funds

Severity: Informational

In `src/ICNFairdropDistribution.sol:112-122`, the `setActivateAt` function can be executed by the `BUSINESS_OPERATOR_ROLE` to activate the contract after user data has been loaded via `batchImportUserData`.

However, there is no verification that the contract holds the necessary funds corresponding to the total user allocations.

This omission could result in a scenario where users are unable to claim their allocated tokens due to insufficient contract balance.

Recommendation

We recommend introducing a cumulative variable that tracks the total allocated amounts during `batchImportUserData`. The `setActivateAt` function should then enforce a check to ensure the contract's balance is at least equal to this cumulative allocation before proceeding with activation.

Status: Acknowledged

8. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such "magic numbers" goes against best practices as they reduce code

readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `src/ICNFairdropDistribution.sol:85,237,264,268,280` (`0x7F` - claim mask)
- `src/ICNFairdropDistribution.sol:256,274` (`0x0003` - claim mask start)
- `src/ICNFairdropDistribution.sol:237,264` (`3` - claim mask bit shifting)

Recommendation

We recommend defining constants with descriptive variable names and comments.

Status: Resolved

9. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- Use `!= 0` instead of `> 0` for non-zero checks to save gas across the multiple instances where this pattern appears.
- In `README.md`, the documentation does not explicitly state that users permanently lose their tokens if they do not claim them within the 30-day window for each vesting step. We recommend updating the `README.md` to explicitly state that unclaimed tokens for a given step are permanently lost after the 30-day claim window expires and will eventually be transferred to the treasury via the `finalizeVesting` function.
- The `userData` mapping in `src/ICNFairdropDistributionStorage.sol:15` uses the traditional anonymous syntax. We recommend refactoring the definition to use named keys and values to `mapping(address user => uint16 data) userData` for improved code clarity.
- Remove custom error `NoTokensToTransfer` defined in `src/interfaces/IICNFairdropDistributionErrors.sol:55`, which is unused.
- The `for` loops in `src/ICNFairdropDistribution.sol` use `<counter>++` syntax. We recommend using unchecked `{ ++i; }` and `unchecked { ++step; }` for gas savings by avoiding unnecessary overflow checks.

- The expression `step - 1` is calculated multiple times in the `claimStep` and `getVestingStepsInfo` functions. We recommend calculating it once and storing it in a local variable within the relevant scope to save gas.
- The `getVestingStepsInfo` function in `src/ICNFairdropDistribution.sol:205-206` reverts if vesting is not activated or the user data is not found. We recommend returning an empty array or an additional boolean to signify nothing to claim instead of reverting in these cases for a smoother off-chain integration experience.
- Calls to `calculateStepAmount` in `src/ICNFairdropDistribution.sol:247` and `isStepUnclaimed` in `src/ICNFairdropDistribution.sol:233` within the `getVestingStepsInfo` loop cause redundant `SLOAD`s of `userData`. We recommend creating internal function variants that accept the pre-loaded `userData` as a parameter or inlining the logic to avoid repeated `SLOAD`s within the loop.

Status: Resolved