Three Sigma

# Code Audit

LAYER3

## Layer3 Season3 Airdrop

# Disclaimer

Code Audit

**Layer3** Season3 Airdrop

# Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

# Table of Contents

Code Audit

**Layer3** Season3 Airdrop

Table of Contents

**Disclaimer** **3**
**Summary** **8**
**Scope** **10**
**Methodology** **12**
**Project Dashboard** **14**
**Code Maturity Evaluation** **17**
**Findings** **20**
3S-Layer3-L01 20
3S-Layer3-L02 21
3S-Layer3-N01 22
3S-Layer3-N02 23

# Summary

Code Audit

**Layer3** Season3 Airdrop

# Summary

Three Sigma audited Layer3 in a 0.4 person week engagement. The audit was conducted from 10/03/2025 to 11/03/2025.

## Protocol Description

Layer3's token distribution in Season 3 will allow users to claim tokens on Ethereum mainnet or Base through smart wallets, with options to stake for full rewards, claim at a reduced rate, earn a 5% staking bonus, and accumulate liquid rewards for using L3 in transactions.

# Scope

Code Audit

**Layer3** Season3 Airdrop

# Scope

| Filepath | nSLOC |
|---|---|
| src/contracts/L3Distributor.sol | 78 |
| src/contracts/LiquidReward.sol | 94 |
| src/contracts/StakingReward.sol | 98 |
| **Total** | **266** |

## Assumptions

Backend keys are securely stored and used as intended.

# Methodology

Code Audit

**Layer3** Season3 Airdrop

# Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

## Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

| Severity / Likelihood | LOW | MEDIUM | HIGH |
|---|---|---|---|
| NONE | None | | |
| LOW | Low | | |
| MEDIUM | Low | Medium | Medium |
| HIGH | Medium | High | High |
| CRITICAL | High | Critical | Critical |

# Project Dashboard

Code Audit

**Layer3** Season3 Airdrop

# Project Dashboard

## Application Summary

| | |
|---|---|
| Name | Layer3 |
| Commit | 94a129e8191f8249c23e08f76eb315be570e5a48 |
| Language | Solidity |
| Platform | Ethereum, Base |

## Engagement Summary

| | |
|---|---|
| Timeline | 10/03/2025 to 11/03/2025 |
| № of Auditors | 1 |
| Review Time | 0.4 person weeks |

## Vulnerability Summary

| Issue Classification | Found | Addressed | Acknowledged |
|---|---|---|---|
| Critical | 0 | 0 | 0 |
| High | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 |
| Low | 2 | 0 | 2 |

| | | | |
|---|---|---|---|
| None | 2 | 1 | 1 |

## Category Breakdown

| | |
|---|---|
| Suggestion | 2 |
| Documentation | 0 |
| Bug | 2 |
| Optimization | 0 |
| Good Code Practices | 0 |

# Findings

Code Audit

**Layer3** Season3 Airdrop

# Findings

## 3S-Layer3-L01

Signatures do not have expiration

| | |
|---|---|
| Id | 3S-Layer3-L01 |
| Classification | Low |
| Severity | Low |
| Likelihood | Medium |
| Category | Bug |
| Status | Acknowledged |

### Description

Signatures in the protocol do not have an expiration time. This might have detrimental business consequences. For example, a signature may be generated for a specific LiquidReward version, but following a contract logic upgrade it can lead to unexpected behaviour.

Likewise, because both **`LiquidReward`** and **`StakingReward`** can be called by someone different from the beneficiary, the generated signature might actually end up being posted even if the user had decided not to use it after all (e.g. if they don't want to claim tokens yet due to some legal liabilities).

### Recommendation

Consider adding an expiration deadline to all signatures to avoid unexpected behaviour or unpleasant user experience.

# 3S-Layer3-L02

## `L3Distributor` hashes do not use EIP712

| Id | 3S-Layer3-L02 |
|---|---|
| Classification | Low |
| Severity | Low |
| Likelihood | Low |
| Category | Bug |
| Status | Acknowledged |

---

### Description

The `L3Distributor` contract uses EIP-191 to encode the signing data in the _claim function. The lack of EIP712 in a non-message data type will lead to unreadable data to be signed by the signer wallet. However, that's generally fine for this situation, considering the signer will be a backend key.

The greater issue lies in the lack of chain ID and verifying contract as part of the hashed data. This allows signature replays in the event of a chain fork. It could also allow replay attacks if the contract were to be deployed again (on another chain or on the same one) with the same Merkle root, but that's very unlikely for the presented use case.

---

### Recommendation

Consider adopting the EIP712 standard. Given that this EIP is already being used in the other contracts in scope, it also brings more standardisation across the protocol contracts.

# 3S-Layer3-N01

**`collectDust`** is only present in **`L3Distributor`**

| | |
|---|---|
| Id | 3S-Layer3-N01 |
| Classification | None |
| Category | Suggestion |
| Status | Addressed in #94a129e. |

## Description

The L3Distributor.collectDust function is used by the owner to withdraw any amount of non-L3 tokens which have been mistakenly sent to the contract. If this is a concern, then the same logic applies to **`StakingReward`** and **`LiquidRewards`**.

## Recommendation

Consider including the same **`collectDust`** function in the other two contracts, adapting to its **`AccessControl`** logic.

# 3S-Layer3-N02

**`L3Distributor`** is the only contract where a third-party cannot sponsor a claim transaction

| | |
|---|---|
| Id | 3S-Layer3-N02 |
| Classification | None |
| Category | Suggestion |
| Status | Acknowledged |

---

## Description

Each contract in scope has a way to claim tokens/rewards, and **`L3Distributor`** is the only one where a sponsor cannot pay for a user's transaction. This is due to the fact that the L3Distributor._claim function uses **`msg.sender`** as part of the hashed data.

---

## Recommendation

Because the Merkle Tree leaf can be used in one of two different functions (**`claim`** and **`claimAndStake`**), which have drastically different outcomes for the user, it does not make sense for a sponsor to be able to decide the used endpoint. A way to allow sponsorship would require that the user signs their desire to make sure the right endpoint is used.

If the team is aware of this or do not value this feature, then the added complexity is unnecessary and nothing should be done about it.