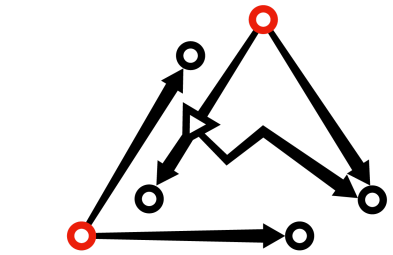

Least Authority

PRIVACY MATTERS

ERC20 and Claim Smart Contracts

Security Audit Report

Aligned Layer

Updated Final Audit Report: 23 December 2024

Table of Contents

Overview

Background

Aligned Layer has requested Least Authority perform a security audit of their ERC20 and Claim Smart Contracts.

Project Dates

- **December 16, 2024 - December 18, 2024:** Initial Code Review *(Completed)*
- **December 19, 2024:** Delivery of Initial Audit Report *(Completed)*
- **December 23, 2024:** Verification Review *(Completed)*
- **December 23, 2024:** Delivery of Final Audit Report *(Completed)*
- **December 23, 2024:** Delivery of Updated Final Audit Report *(Completed)*

Review Team

- Nikos Iliakis, Security Researcher and Engineer
- Will Sklenars, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the ERC20 and Claim Smart Contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Claim Contracts: https://github.com/yetanotherco/aligned_layer/tree/ee2efe276cdd21ec702e79d8970125ef77a09681/claim_contracts/src
    - ERC20 + the claim contract

Specifically, we examined the Git revision for our initial review:

- `E26c62f38c472ba5c4cfc9f87a37db88243f5316`

For the verification, we examined the Git revision:

- `1636bb23ccbc1a99fefe96389012fb514c154993`

For the review, this repository was cloned for use during the audit and for reference in this report:

- Aligned Layer: https://github.com/LeastAuthority/Aligned-Layer

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Website:
  https://alignedlayer.com

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the network;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use of the smart contracts or disrupt their execution;
- Vulnerabilities in the smart contracts' code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team performed a security audit of Aligned Layer's ERC20 token contract for the Aligned Token $ALIGN. This audit also included the `ClaimableAirdrop` contract for token distribution and the deployment process.

We examined the contracts for possible attack vectors due to re-entrancy and found that the Aligned Layer team has used OpenZeppelin's `ReentrancyGuard` to mitigate this risk. We investigated ways malicious third-party contracts could be called to execute arbitrary logic but did not identify any areas of concern within this context. While it does not pose an immediate vulnerability, we found one case where the Checks-Effects-Interactions pattern was not followed (Suggestion 1).

We additionally reviewed the contracts for scenarios that could result in the accidental loss of funds, or loss of control of the contracts, and did not identify any issues with this area of investigation. The contracts use zero address checks whenever addresses are supplied as arguments to public functions, and the ERC20 contract contains a custom function to override the OpenZeppelin `renounceOwnership` function, eliminating the possibility that the contract's ownership could be accidentally renounced.

In addition we found a few opportunities for improvement and reported some suggestions related to testing (Suggestion 3), best practices (Suggestion 2), and gas efficiency (Suggestion 4).

System Design

Our team examined the design of the ERC20 and Claim Smart Contracts and found that security has been taken into consideration as demonstrated by adherence to best practices, as well as the utilization of the well-audited Open Zeppelin libraries. Our team also noted that due to the minimal amount of custom logic in the ERC20 contract, the attack surface, along with the potential implementation of incorrect logic and possible occurrence of bugs, is significantly reduced.

The contracts are upgradable and use the OpenZeppelin proxy pattern. Additionally, ownership is transferable, leveraging OpenZeppelin's `Ownable2StepUpgradeable` library. Having the two-step ownership transfer is a recommended practice, as it eliminates the possibility of transferring ownership to an unknown or inactive account due to user error.

Furthermore, Aligned Layer's `ClaimableAirdrop` contract leverages a Merkle tree system and a map to keep track of which users are eligible to claim tokens, and which of those users have already claimed their tokens. The `ClaimableAirdrop` contract stores the Merkle root of a Merkle Tree where the leaf nodes are a digest of the eligible claimant's account and claim amount. When an eligible user wishes to claim their tokens, they must submit a Merkle proof that proves their eligibility. The contract builds a Merkle leaf using the claimant's address and token amount, and checks this against the Merkle root stored in the contract, using the Merkle proof supplied by the claimant. The Merkle root is updatable, which allows the Aligned Layer team to update the collection of eligible claimants. If a user's claim is successful, the contract will update the `hasClaimed` map, the purpose of which is to ensure a claimant can only claim once. In our review of the codebase, we identified one opportunity for gas optimization, and recommended using a bitmap in conjunction with an off-chain data structure to keep track of which user has claimed the token, thereby reducing the amount of data that needs to be stored on-chain (Suggestion 4).

Code Quality

We performed a manual review of the repositories in scope and found the contracts to be straightforward and well-written. Additionally, the codebase is well-organized and easy to read and reason about, in that it adheres closely to Solidity best practices.

Tests

During our review, our team found no tests within the codebase. Although the contracts' logic is relatively simple, we recommend writing comprehensive unit tests, as unit tests can help detect implementation errors that could result in unexpected behaviour (Suggestion 3).

Documentation and Code Comments

There was no documentation provided specifically for the smart contracts in scope. However, the lack of documentation was not an issue, as the smart contracts are simple. Additionally, the Aligned Layer system, as a whole, has sufficient documentation, and the code is well-commented.

Scope

The scope of this review was sufficient, as it included both, the smart contracts and the deployment script.

Dependencies

Our team noted that the only dependencies implemented in the codebase are the OpenZeppelin contracts. The OpenZeppelin contracts are the latest stable versions and have no known vulnerabilities.

However, we recommend updating and locking down the Solidity compiler pragma, as newer versions of the Solidity compiler can contain security-relevant fixes (Suggestion 2).

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Suggestion 1: Update Claim Function To Implement Checks-Effects-Interaction Pattern	Resolved
Suggestion 2: Upgrade Solidity Compiler Version and Lock the Pragma	Resolved
Suggestion 3: Add Tests	Unresolved
Suggestion 4: Improve Gas Efficiency	Unresolved

Suggestions

Suggestion 1: Update Claim Function To Implement Checks-Effects-Interaction Pattern

Location

`claim_contracts/src/ClaimableAirdrop.sol#L114`

Synopsis

The storage value `hasClaimed` is updated after an external call (ERC20 transfer). Since the token contract is known and the function implements `nonReentrant`, this does not currently pose a security concern. However, issues may occur in any future updates.

Mitigation

We recommend updating the storage variable before the external call.

Status

The Aligned Layer team has implemented the suggestion as recommended.

Verification

Resolved.

Suggestion 2: Upgrade Solidity Compiler Version and Lock the Pragma

Location

`claim_contracts/src/ClaimableAirdrop.sol#L2`

`claim_contracts/src/AlignedToken.sol#L2`

Synopsis

The pragma on the contracts is version `^0.8.19x` floating. A floating pragma is an error-prone practice that could lead to deployment issues in the case that an incorrect compiler version is used. The older compiler versions have known and fixed issues and can be used maliciously.

Mitigation

We recommend upgrading to the most recent compiler version (Solidity `0.8.28`), as it may include features and bug fixes for issues that were present in previous versions, and locking the pragma.

Status

The Aligned Layer team updated the compiler version to Solidity `0.8.28`.

Verification

Resolved.

Suggestion 3: Add Tests

Location

Throughout the codebase.

Synopsis

While there are some sanity/integration tests for after the contracts are deployed, there are no unit tests. Having good test coverage can help with the early identification of bugs and also facilitate refactoring.

Mitigation

We recommend writing comprehensive tests for the smart contracts.

Status

The Aligned Layer team has acknowledged this suggestion and does not plan to immediately address it.

Verification

Unresolved.

Suggestion 4: Improve Gas Efficiency

Location

`claim_contracts/src/ClaimableAirdrop.sol#L31`

Synopsis

The current implementation of the `Airdrop` uses a mapping to store a Boolean for each address, which can incur additional costs as the number of claimers increases.

Mitigation

We recommend using a bitmap, which will occupy only one bit for each user (as a reference, we recommend checking how Uniswap used it in its `MerkleDistributor`). This, however, will require an index that corresponds to each user off-chain and that is used in the claim and Merkle leaves.

Status

The Aligned Layer team has acknowledged this suggestion and does not plan to immediately address it.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.