Zellic

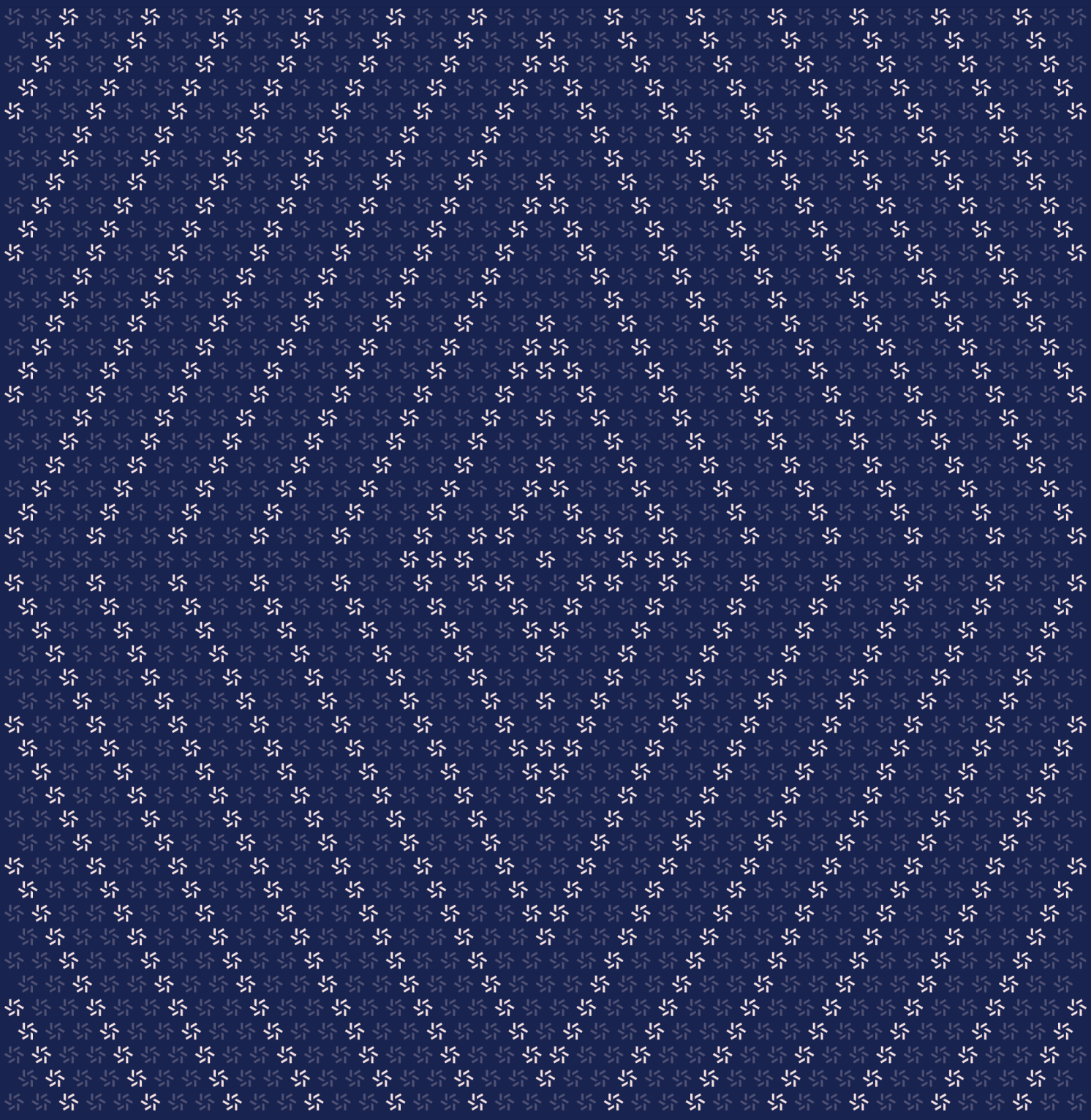
Prepared for
Matthew Katz
Parker Jou
Nathan Leung
Caldera

Prepared by
Junghoon Cho
Jaeeu Kim
Zellic

October 28, 2024

Rollup Token Contracts

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Caldera from October 24th to October 25th, 2024. During this engagement, Zellic reviewed the Rollup Token contracts' code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is it possible to claim unauthorized Airdrop allocations?
- Is the cryptographic system implemented for Airdrop secure?
- Are functions appropriately access controlled?
- Are the Rollup Token contracts secure against common smart contract vulnerabilities?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Backend server for GitHub claim
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

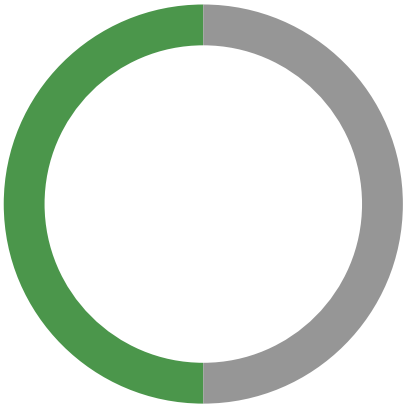
1.4. Results

During our assessment on the scoped Rollup Token contracts, we discovered two findings. No critical issues were found. One finding was of low impact and the other finding was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Caldera in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	1



2. Introduction

2.1. About Rollup Token Contracts

Caldera contributed the following description of the Rollup Token contracts:

> Caldera is building a set of contracts for 1-click deployment of native tokens on customer rollups with ownability / upgradeability / distribution so partners can launch tokens in one click.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Rollup Token Contracts Contracts

Type	Solidity
Platform	EVM-compatible
Target	rollup-token-contracts-zellic
Repository	https://github.com/nate-caldera/rollup-token-contracts-zellic ↗
Version	`3a8e004d3b2edbab6d21997cd6ecb429b8c23461`
Programs	`Airdrop.sol` `RollupToken.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-days. The assessment was conducted by two consultants over the course of two calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Junghoon Cho
Engineer
junghoon@zellic.io ↗

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

2.5. Project Timeline

October 24, 2024	Start of primary review period
October 25, 2024	End of primary review period

3. Detailed Findings

3.1. Lack of parameter validation in initialization

Target	Airdrop.sol		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `initialize` function of the Airdrop contract does not perform validation checks on its input parameters. Critical addresses like `airdropVault_`, `githubSigner_`, and `token_` could be set to the zero address. Additionally, `startTime_` could be set after `endTime_`, contradicting the logical flow of the airdrop period.

```
function initialize(
[...]
    __UUPSUpgradeable_init();
    __Pausable_init();
    __Ownable_init(initialOwner);
    airdropVault = airdropVault_;
    githubSigner = githubSigner_;
    token = token_;
    addressClaimMerkleRoot = addressClaimMerkleRoot_;
    addressClaimMessage = addressClaimMessage_;
    githubClaimMerkleRoot = githubClaimMerkleRoot_;
    startTime = startTime_;
    endTime = endTime_;
[...]
```

Impact

Initializing with zero addresses can prevent essential functions from executing properly. Since the `token` address cannot be updated later due to the absence of a setter function, an incorrect `token` address during initialization would permanently disable token-related operations.

Also, the incorrect timestamps can render the airdrop inaccessible; however, they can be modified later with the setter functions.

Recommendations

Add validation checks within the `initialize` function to ensure all parameters are valid.

Remediation

This issue has been acknowledged by Caldera, and a fix was implemented in commit 4b13e87a ↗.

3.2. Incorrect `MintCapLowered` event emission

Target	RollupToken.sol		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `lowerMintCap` function decreases the value of the `mintCapBips` variable, which limits the number of tokens that can be minted at once. When the value of `mintCapBips` changes, a `MintCapLowered` event is emitted that includes both the previous and updated values.

```
function lowerMintCap(uint256 newMintCapBips) external onlyOwner {
    if (newMintCapBips > mintCapBips) {
        revert MintCapTooHigh();
    }
    mintCapBips = newMintCapBips;
    emit MintCapLowered(mintCapBips, newMintCapBips);
}
```

However, since `mintCapBips` has already been overwritten with the new value, both `mintCapBips` and `newMintCapBips` are emitted with the same value.

Impact

Since both `mintCapBips` and `newMintCapBips` are emitted with the same value, it becomes challenging to distinguish between the previous and updated values, reducing transparency in the change history.

Recommendations

Modify the code as follows.

```
function lowerMintCap(uint256 newMintCapBips) external onlyOwner {
    if (newMintCapBips > mintCapBips) {
        revert MintCapTooHigh();
    }
    uint256 oldMintCapBips = mintCapBips;
    mintCapBips = newMintCapBips;
```

```
    emit MintCapLowered(oldMintCapBips, newMintCapBips);
}
```

Remediation

This issue has been acknowledged by Caldera, and a fix was implemented in commit 2b275bfc ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Recommendation on GitHub claim procedure

Users authenticate their GitHub credentials through OAuth on a separate backend server, obtaining a signed message that includes their GitHub username and Ethereum wallet address. They can then call the `githubClaim` function with this message as an argument to claim an airdrop.

However, since GitHub usernames can be changed arbitrarily, they may not be suitable as a unique identifier for identifying users. For this reason, it is recommended to use the GitHub user ID rather than the username.

4.2. Centralization risks

This section documents the roles defined in the Caldera contracts and highlights the potential centralization risks associated with them according to the contract and the specifications:

RollupToken

- `owner`
  - **Privileges:** Minting tokens and setting the mint cap.
  - **Restrictions:** None.
  - **Key-custody solution:** Assigned to a multi-sig wallet.

Airdrop

- `owner`
  - **Privileges:** Pausing/unpausing the contract, setting the airdrop duration, updating the blocklist, setting claim-Merkle trees and `githubSigner` address, and managing the `airdropVault`.
  - **Restrictions:** None.
  - **Key-custody solution:** Assigned to a multi-sig wallet.
- Additional notes:
  - The `airdropVault` is considered to have approved the assets for the Airdrop contract in order to proceed with `safeTransferFrom` during distribution.

Backend

- `githubSigner`
  - **Privileges:** Signing the GitHub username for the GitHub-based claim in the backend of Caldera.
  - **Restrictions:** Backend access only.

The above introduces centralization risks that users should be aware of, as it grants a single point of control over the system. We recommend that these centralization risks be clearly documented for users so that they are aware of the extent of the owner's control over the contract. This can help users make informed decisions about their participation in the project. Additionally, clear communication about the circumstances in which the owner may exercise these powers can help build trust and transparency with users.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. RollupToken

The RollupToken contract is an upgradable ERC-20 token that inherits OpenZeppelin's ERC20Upgradeable, ERC20PermitUpgradeable, and ERC20VotesUpgradeable contracts, allowing for voting and delegation functionalities. Additionally, the RollupToken contract inherits OpenZeppelin's Ownable contract, enabling the designation of an admin to manage the contract. This admin has the authority to call the mint function and set the mint cap.

During the deployment of the RollupToken contract, a specific amount of tokens is minted to the `airdropVault`, within the limit of the total supply.

Rate limit on mint

The RollupToken contract implements several restrictions on token minting. First, it limits the number of tokens an admin can mint in a single transaction through the `mintCapBips` variable. Specifically, the number of tokens minted at once cannot exceed a fraction of `mintCapBips / 10000` of the current total supply. While the admin can adjust the `mintCapBips` value by calling the `lowerMintCap` function, this new value cannot exceed the existing one, allowing only for reductions.

Additionally, to prevent bypassing the aforementioned minting restrictions through multiple mint calls, the RollupToken contract enforces an interval between each mint operation. When a mint is called, the timestamp for the next allowable mint is updated, and this interval is defined by the constant `MINIMUM_MINT_INTERVAL` variable. In the current version of the contract, the `MINIMUM_MINT_INTERVAL` is set to one year.

5.2. Airdrop

The Airdrop contract is responsible for distributing tokens to eligible addresses through address-based and GitHub-based claims. The contract inherits OpenZeppelin's Ownable contracts, allowing for the designation of an admin to manage the contract. This admin has the authority to set the airdrop duration, update the blocklist, set claim-Merkle trees, and manage the `airdropVault`.

Airdrop duration

The Airdrop contract sets a duration during which tokens can be claimed via the airdrop. This duration is defined by the `startTime` and `endTime` variables, requiring that the timestamp of the transac-

tion block in which a user calls `claim` falls between `startTime` and `endTime`. The contract admin can modify these variables using the `setStartTime` and `setEndTime` functions, which enforce an invariant preventing `startTime` from being set after `endTime` or `endTime` from being set before `startTime`.

Blocklist system

The Airdrop contract manages a `blocklist`, which is a boolean mapping for addresses. This blocklist can be modified by the admin through the `updateBlocklist` function, allowing the admin to arbitrarily set the blocked status of specific addresses. Addresses marked as blocked are unable to claim tokens through either address-based or GitHub-based claims. Given the centralization risk posed by the admin's ability to block specific users from claiming, this system should be used cautiously and only in critical situations, such as emergency responses to hacking incidents.

Address-based claim

The Airdrop contract allows eligible addresses to claim their airdrop using a signature for verification. The address-based claim includes the following fundamental checks:

- The contract must not be paused.
- The current timestamp must fall within the valid airdrop duration.
- The caller must not be blocked.

A user attempting to claim must provide a signed message as a parameter, which includes content such as the user's acceptance of the airdrop's terms and conditions. The signer of this message is defined in the contract by the `addressClaimMessage` variable. If the signature of the message is verified, the contract then uses a Merkle tree to check whether the caller is eligible to claim. It verifies eligibility by querying the Merkle tree with `msg.sender` and the amount to be claimed, and it records the claim to prevent duplicate claims through replay attacks. Once all validations are complete, the specified amount of tokens is transferred to the user from the `airdropVault`.

GitHub-based claim

The Airdrop contract also supports GitHub-based claims, which allow users to claim tokens by submitting a GitHub username that meets the airdrop conditions. The GitHub-based claim includes the following fundamental checks:

- The contract must not be paused.
- The current timestamp must fall within the valid airdrop duration.
- The caller must not be blocked.

A user attempting to claim must provide a signed message that is same as the address-based claim. However, the message should contain the GitHub username additionally. A user has to verify a signature with the GitHub username signed by `githubSigner`, which is a backend signer of Caldera. If the signature of the message is verified, the contract then uses a Merkle tree to check whether the caller is eligible to claim. It verifies eligibility by querying the Merkle tree with `githubUsername`

and the amount to be claimed, and it records the claim to prevent duplicate claims through replay attacks. Once all validations are complete, the specified amount of tokens is transferred to the user from the `airdropVault`.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to Ethereum Mainnet.

During our assessment on the scoped Rollup Token contracts, we discovered two findings. No critical issues were found. One finding was of low impact and the other finding was informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.