Zellic

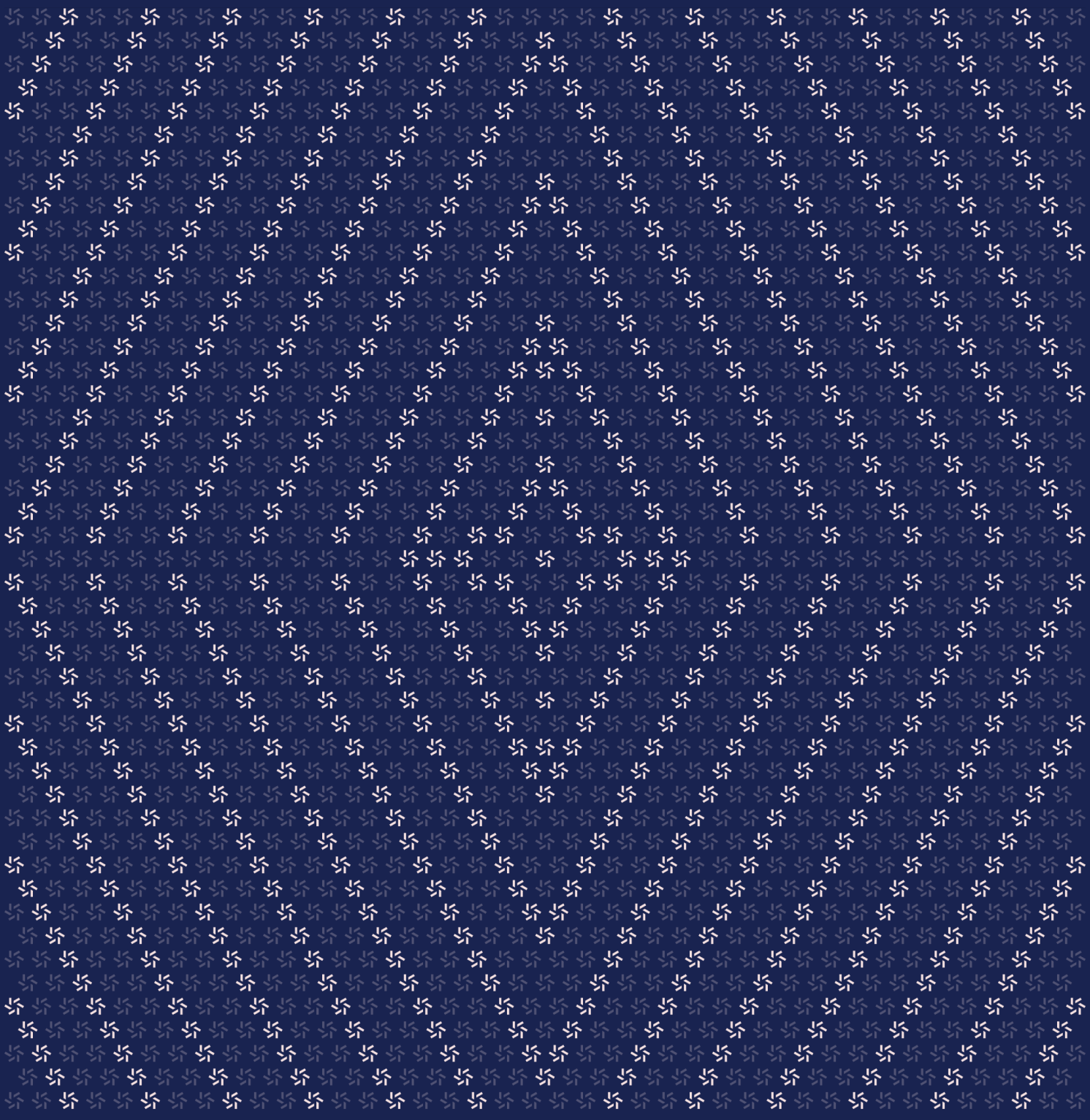
Prepared for
Dan Oved
Iain Nash
Zora

Prepared by
Seunghyeon Kim
Jinheon Lee
Zellic

March 17, 2025

Zora Token

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 Zora from March 10th to March 11th, 2025. During this engagement, Zellic reviewed Zora Token's 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 there a mismatch between the allocation amounts and the amount claimed?
- Are there any errors or issues with setting allocations?
- Are the tokens implemented safely so they cannot be drained?
- Are the signature-verification steps reasonable?

1.3. Non-goals and Limitations

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

- 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 Zora Token contracts, we discovered one finding, which was informational in nature.

Breakdown of Finding Impacts

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

2. Introduction

2.1. About Zora Token

Zora contributed the following description of Zora Token:

> The Zora token (ZORA) is an ERC20 token with voting capabilities and permit functionality for gasless approvals. It uses a delayed initialization pattern where all tokens are minted in a single event after deployment, creating a fixed supply. The ZoraTokenCommunityClaim contract enables gas-efficient token claiming to community members through direct storage of allocations rather than merkle proofs. It features time-gated claims, and signature-based delegation for claiming on behalf of others (including smart wallet support).

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.

2.3. Scope

The engagement involved a review of the following targets:

Zora Token Contracts

Type	Solidity
Platform	EVM-compatible
Target	zora-token
Repository	https://github.com/ourzora/zora-token ↗
Version	`ba75438808458efce67eb32947e339edcc547634`
Programs	`Zora` `ZoraTokenCommunityClaim`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of three 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:

Seunghyeon Kim
Engineer
seunghyeon@zellic.io

Jinheon Lee
Engineer
jinheon@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

March 10, 2025	Kick-off call
March 10, 2025	Start of primary review period
March 11, 2025	End of primary review period

3. Detailed Findings

3.1. The allocation amount could be incorrect

Target	ZoraTokenCommunityClaim		
Category	Protocol Risks	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The function `setAllocations` only uses `bytes32` for a pair of addresses and amounts. The design expects that 160 bits will be used for the address and the leftover 96 bits will be the amount. Generally, `uint256` is used for manipulating a token of which its decimals are 18, but it is used with `uint96`. Therefore, we are concerned for situations where `uint96` is not enough to deal with the amount the user wants to allocate.

```
function setAllocations(bytes32[] calldata packedData) external override {
    require(msg.sender == admin, OnlyAdmin());
    require(!claimIsOpen(), ClaimOpened());

    for (uint256 i = 0; i < packedData.length; i++) {
        // Extract address from first 160 bits
        address user = address(uint160(uint256(packedData[i])));

        // Extract allocation from remaining bits (shift right by 160 bits)
        uint96 amount = uint96(uint256(packedData[i]) >> 160);

        // Store the allocation
        compactAllocations[user] = amount;
    }

    emit AllocationsSet(packedData);
}
```

Impact

A user cannot allocate more tokens than the range of `uint96`.

Recommendations

We recommend expanding the room for the `packedData`.

Remediation

This issue has been acknowledged by Zora. Zora provided following response:

> Our token decimals is 18 and the max amount fits with plenty of room for `uint96`.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Module: ZoraTokenCommunityClaim.sol

Function: `claim(address _claimTo)`

This function is for claiming the allocation.

Inputs

- `_claimTo`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The address to send the tokens to.

Branches and code coverage

Intended branches

- Try to claim the allocation to the given address.
  - ☑ Test coverage

Negative behavior

- Revert if the caller has no allocation.
  - ☑ Negative test
- Revert if the claim is not opened.
  - ☑ Negative test
- Revert if the caller has already claimed.
  - ☑ Negative test

4.2. Module: Zora.sol

Function: `initialize(address[] tos, uint256[] amounts, string contractURI_)`

This function is for initializing this contract.

Inputs

- `tos`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the same length as the input `amount`.
  - **Impact**: Array of recipient addresses.
- `amounts`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must be the same length as the input `tos`.
  - **Impact**: Array of amounts to mint.
- `contractURI_`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Must not be null.
  - **Impact**: Contract URI to set for the token.

Branches and code coverage

Intended branches

- Mint the given amount of tokens to the given address.
  - ☑ Test coverage

Negative behavior

- Revert if the length of the inputs `tos` and `amounts` are not the same.
  - ☑ Negative test
- Revert if the caller is not an admin.
  - ☑ Negative test
- Revert if the input `contractURI_` is null.
  - ☑ Negative test

5. Assessment Results

At the time of our assessment, the Zora Token code was deployed to the Base network without initialization, but the ZoraTokenCommunityClaim contract was not yet deployed.

During our assessment on the scoped Zora Token contracts, we discovered one finding, which was informational in nature.

5.1. 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.