PUBLIC

Code Assessment

of the Enso Token Smart Contracts

October 5, 2021

Produced for

by

Contents

1 Executive Summary

Dear Sir or Madam,

First and foremost we would like to thank EnsoLabs for giving us the opportunity to assess the current state of their Enso Token system. This document outlines the findings, limitations, and methodology of our assessment.

Initially, our code assessment resulted in a number of low severity findings. After the submission of the intermediate reports all findings have been resolved. These have been marked accordingly and can be found in the Resolved Findings section. We want to highlight, that the provided contract implements the Enso Token by extending the OpenZeppelin's `ERC20VotesComp` Token, and `ERC20VotesComp` extends `draft-ERC20Permit` which is in draft state and not audited.

We hope that this assessment provides more insight into the current implementation and provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3
• Code Corrected	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Enso Token repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	September 17 2021	159c5788f82f0f5e67832711e9325b6a89b9398c	Initial Version
2	September 24 2021	fcf17e428c1599fb7ad17bcdbf9272daa59220f6	Version with Fixes

For the solidity smart contracts, the compiler version `0.8.2` was chosen. File `contracts/Enso.sol` is the only contract that was part of this assessment.

2.1.1 Excluded from scope

Imported libraries are assumed to be working correctly according to their specification.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section we have added a version icon to each of the findings to increase the readability of the report.

EnsoLabs offers a token contract which implements the ERC20 token standard with a voting system inherited from `ERC20VotesComp`. The contract makes some requirements for minting including a minimum time between two mints and a cap on the maximum amount per single mint. This amount should be less than a defined percentage of the `TotalSupply` and the percentage can be updated only once.

2.2.1 Trust Model

The minter stored in variable `minter` is considered to be trusted and assumed to act in a non malicious way.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. We draw attention to the fact that due to inherent limitations in any software development process and software product an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third party infrastructure which adds further inherent risks as we rely on the correct execution of the included third party technology stack itself. Report readers should also take into account the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severities. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved, have been moved to the Resolved Findings section. All of the findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Division Before Multiplication Code Corrected
- Outdated Compiler Code Corrected
- Sanity Check Missing Code Corrected

6.1 Division Before Multiplication

Design Low Version 1 Code Corrected

In `Enso.mint` the following snippet is used:

```
require(_amount <= (totalSupply() / 100 ) * mintCap, "Enso#mint: exceeded mint cap");
```

The aim of the snippet is to enforce that no more than `mintCap` percent (currently 2%) of the current total supply is minted. However, due to rounding errors in the division, the enforced cap might be lower.

Code corrected:

The division is performed after multiplication now.

6.2 Outdated Compiler

Design Low Version 1 Code Corrected

The `solc` version is fixed in the hardhat configuration to version `0.8.2`. Please note that this version is outdated. `v0.8.7` is the latest available version for `solc`.

Code corrected:

Solidity compiler version 0.8.9 is used for compilations now.

6.3 Sanity Check Missing

Design Low Version 1 Code Corrected

In `Enso.mint` there is no sanity check for the argument `_amount`. If the value of `_amount` is set to 0 by a mistake, the minting process will block for a year.

Code corrected:

The check that ``_amount > 0 `` was added.