HecoDAO Audit Report

Version 1.0.0

Serial No. 20211129000222022

Presented by Fairyproof

November 29, 2021

FAIRYPROOF

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the HecoDAO project, at the request of the HecoDAO team.

Audit Start Time:

November 25, 2021

Audit End Time:

November 26, 2021

Audited Source Files:

The calculated SHA-256 values for the audited files when the audit was done are as follows:

```
HecoDAOProposal.sol:
0x4ff2f31962d2356f03f44fea50d318f98226ba35f09202fbe35d2a22172026ce
```

The goal of this audit is to review HecoDAO's solidity implementation for its governance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the HecoDAO team for specified versions. Whenever the code, software, materials, settings, enviroment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— The HecoDAO Team's Consent/Acknowledgement:

The audited materials of the project including but not limited to the documents, home site, source code, etc are all developed, deployed, managed, and maintained outside Mainland CHINA.

The members of the team, the foundation, and all the organizations that participate in the audited project are not Mainland Chinese residents.

The audited project doesn’t provide services or products for Mainland Chinese residents.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.

2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

For this audit, we used the following sources of truth about how the HecoDAO system should work:

https://docs.hecochain.com/

whitepaper

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the HecoDAO team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2021112900022022	Fairyproof Security Team	November 25, 2021 - November 26, 2021	Passed

Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. No risks have been discovered at the time of writing.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to Heco

HECO Chain (Heco) is a decentralized, high-efficiency and energy-saving public chain. It is compatible with smart contracts and supports high- performance transactions. The endogenous token of Heco is HT and it adopts the HPoS consensus mechanism. Heco will continue to improve the efficiency of Ethereum by Layer2, which will supplement and empower the Ethereum ecosystem.

04. Major functions of audited code

The audited code mainly implements the following functions:

- the admin can add or remove a validator
- any validator can propose a proposal
- a proposal will go to the voting stage after it is verified by validators excluding the proposing validator
- all validators can participate in voting and the outcome will be saved in the contract
- the contract is upgradeable

The admin has the following privilege(s):

- adding or removing a validator

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack

- Reordering Attack
- DDos Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Neutral is not an issue or risk but a suggestion for code improvement.

07. List of issues by severity

- N/A

08. Issue descriptions

- N/A

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Add Validation for the Value of An `option`

The `vote` function allowed each `validator` to vote only once on a `proposal`. And the value of an `option` couldn't be modified. If somehow the value of an `option` were to be invalid, the vote associated with the `option` would be invalid and the validator that had cast the vote would lose this chance to vote.

Recommendation:

Consider adding a conditional check to validate the value of `option` to prevent a voter from being losing his/her chance to cast a valid vote.

Status: it has been fixed by the HecoDAO team.

- Add An Event for Address Update by Admin

The `setOperator` function defined in line 111 of the `HecoDAOProposal.sol` file could be used to change the admin's address. This was a sensitive and important status update which should be recorded, traced and monitored.

Recommendation:

Consider defining an event such as `OperatorChange` in the `setOperator` function to allow this update conveniently traced and monitored.

Status: it has been fixed by the HecoDAO team.

- Remove Unused Function Inherited From `ReentrancyGuardUpgradeable`

Line 11 of the `HecoDAOProposal.sol` file has the following code section:

```
contract HecoDAOProposal is ReentrancyGuardUpgradeable, OwnableUpgradeable
```

In the above code, `HecoDAOProposal` inherited all the functions of `ReentrancyGuardUpgradeable`. However `HecoDAOProposal` didn't use the `nonReentrant` function.

Recommendation:

Consider removing the inheritance from `ReentrancyGuardUpgradeable` if the `nonReentrant` function is never used.

Status: it has been fixed by the HecoDAO team.

- Refine Implementation of `auditProposal` to Reduce `gas` Consumption

From line 170 to line 178 of the `HecoDAOProposal.sol` file, the `auditProposal` function had a constraint `auditOperator[_proposalId].length < 2`. This constraint resulted in two conditions: `auditOperator[_proposalId].length == 0` or `auditOperator[_proposalId].length == 1`. Therefore there was no need to have an additional constraint `auditOperator[_proposalId].length == 1`. Removing this additional constraint could remove two `sload` operations to reduce gas consumption and enhance the code's readability.

Recommendation:

Consider removing `auditOperator[_proposalId].length == 1` .

Status: it has been fixed by the HecoDAO team.