HecovotinLoan Audit Report

Version 1.0.0

Serial No. 2021091400022023

Presented by Fairyproof

September 14, 2021

01. Introduction

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

Audit Start Time:

September 7, 2021

Audit End Time:

September 11, 2021

Audited Code's Github Repository:

https://github.com/fildaio/HecovotinLoan/tree/dev/contracts

Audited Code's Github Commit Number When Audit Started:

76ef0c7dbd360e8653c06099cfb7a6d7d8be09fc

Audited Code's Github Commit Number When Audit Ended:

2a04570393270cc81f0fbe99e91302a110802eec

Audited Source Files:

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

```
ComptrollerInterface.sol:
0xe6d721d6d0be5b40b49aab88f3530f9f286a1f0abce7bf17bdc370e4e132b956

GlobalConfig.sol:
0x8226dff0b117d6fff0ac6212fa1fe34f80ba0d9ec0571796ac12b96d86aae999

HTToken.sol:
0x5ab0e4456f26e6ed0b439fded98fe915b34df788268e4da6797aa30e153b0cc7

HTTokenInterface.sol:
0xd5f52abb3a303e7482631c1702c0515b9ac5554b05345163aeb38f438de636a4

LoanInterface.sol:
0xc9c38334bb400459c32ac74ecb1961bf855894b329aaced23e54aa89f4b684dd

LoanViaFilda.sol:
0x1007ad95c6207d572a8071061b23553ba6f0806143c0f19a24d0d521ba634493

Migrations.sol:
0x4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4

Wallet.sol:
0xe00b9661b7c2ff87fc11b808eb2af3f9d554cadc21e82a2168c20689badff294
```

```
WalletFactory.sol:
0x96d255e117f1b96a2a075c61098f12ede3013fce49e7aaaded6e795af1c7deaa

WalletFactoryInterface.sol:
0x03dcb646aa53fc821e80ac820e6514338d0a9a25453f971c0311533c1b4004f4
```

The source files audited include all the files with the extension "sol" as follows:

```
contracts/
├── ComptrollerInterface.sol
├── GlobalConfig.sol
├── HTToken.sol
├── HTTokenInterface.sol
├── LoanInterface.sol
├── LoanViaFilda.sol
├── Migrations.sol
├── Wallet.sol
├── WalletFactory.sol
└── WalletFactoryInterface.sol
```

The goal of this audit is to review HecovotinLoan's solidity implementation for its voting and loan functions, 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 HecovotinLoan 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.

— 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 HecovotinLoan system should work:

https://github.com/fildaio/HecovotinLoan/tree/dev/contracts

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

— Comments from Auditee

No vulnerabilities with critical, high, medium or low-severity were found in the above source code.

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 HecovotinLoan

HecovotinLoan is a Heco based project that implements Heco's node voting function and provides HT based loan services.

04. Major functions of audited code

The audited code implements the following functions:

- Users who deposit crypto assets in the application are qualified to vote and get rewards in HT

- Users who stake the HT tokens in the application can get loans or interests
- When a user's staked assets cannot cover his/her loans, liquidation of his/her staked assets will happen.

05. Key points in audit

During the audit, we worked closely with the HecovotinLoan team and helped fix some issues as follows:

- Adding Constraints to Variables

Source and Description:

In the `setBorrowRate` function defined in the `GlobalConfig.sol` file, the variable `value` should be less than `denominator`, otherwise the `getBorrowLimit` function defined in the `Wallet.sol` file would get an incorrect result. Here was the code section:

```
function setBorrowRate(uint256 value) public {
        _byConfigRole();
        borrowRate = value;
    }
```

In the `setBonusRateForLiquidater` function defined in the `GlobalConfig.sol` file, the variable `value` should be less than `denominator`, otherwise the calculation of `bonus` in both line 251 and line 253 in the `Wallet.sol` file would be greater than `total`, which was incorrect. Here was the code section:

```
function setBonusRateForLiquidater(uint256 value) public {
        _byConfigRole();
        bonusRateForLiquidater = value;
    }
```

In the `setDenominator` function defined in the `GlobalConfig.sol` file, the variable `value` should be greater than `borrowRate` and `bonusRateForLiquidater`, otherwise the `getBorrowLimit` function defined in the `Wallet.sol` file would get an incorrect result. Here was the code section:

```
function setDenominator(uint256 value) public {
        _byConfigRole();
        denominator = value;
    }
```

Recommendation:

Consider adding constraints for the variables, our recommended fixes are as follows:

Change from:

```
function setBorrowRate(uint256 value) public {
        _byConfigRole();
        borrowRate = value;
    }
```

to

```
function setBorrowRate(uint256 value) public {
      require(value < denominator); //a constraint is needed
        _byConfigRole();
        borrowRate = value;
    }
```

Change from:

```
function setBonusRateForLiquidater(uint256 value) public {
        _byConfigRole();
        bonusRateForLiquidater = value;
    }
```

to:

```
function setBonusRateForLiquidater(uint256 value) public {
      require(value < denominator); //a constraint is needed
        _byConfigRole();
        bonusRateForLiquidater = value;
    }
```

Change from:

```
function setDenominator(uint256 value) public {
        _byConfigRole();
        denominator = value;
    }
```

to:

```
function setDenominator(uint256 value) public {
      require(value > borrowRate); //a constraint is needed
      require(value > bonusRateForLiquidater); //a constraint is needed
        _byConfigRole();
        denominator = value;
    }
```

Update: it has been fixed in the latest code.

06. 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
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

07. 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.

08. Major areas that need attention

Based on the provided souce code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that can modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Constraints for Variables

We checked whether or not there were issues with variables' boundaries. We helped fix some issues. For more details please refer to "05. Key points in audit".

- Voting Logic and Reward Mechanism

We checked whether or not the voting logic and the reward mechanism worked correctly.

We didn't find issues or risks in these functions or areas at the time of writing.

- Savings and Loans

We checked whether or not the savings mechanism and loan mechanism worked correctly.

We didn't find issues or risks in these functions or areas at the time of writing.

- Liquidation

We checked whether or not the liquidation mechanism worked correctly.

We didn't find issues or risks in these functions or areas at the time of writing.

- Asset Security

We checked whether or not all the functions that transfer assets are safely hanlded.

We didn't find issues or risks in these functions or areas at the time of writing.

- Contract Migration/Upgrade

We checked whether or not the contract files introduce issues or risks associated with contract migration/upgrade.

We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We didn't find issues or risks in other functions or areas at the time of writing.

09. List of issues by severity

A. Critical

- N/A

B. High

- N/A

C. Medium

- N/A

D. Low

- N/A

10. List of issues by source file

- N/A

11. Issue descriptions

- N/A

12. 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.

- N/A