FilDA Audit Report

Version 1.0.0

Serial No. 2021021900012020

Presented by Fairyproof

Feburary 19, 2021

FAIRYPROOF

01. Introduction

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

The audited code can be found in three public Github repositories:

the FilDA's CandyDispenser Github repository, and the version used for this report is commit 45fa502e849c6a7d2ec754c733127372bc0c41cb ,

the FilDA's AssetManager Github repository, and the version used for this report is commit 85b08f23a19b9a02fce0c4bd44423ea7f41ff106 and

the FilDA's Quicksilver Github repository, and the version used for this report is commit 9aecf2e1edb18eaecc8f39c705975414f3388d17 .

The goal of this audit is to review FilDA's solidity implementation for a lending application, 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.

— 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 smart contract 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. 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

FilDA's codebase 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 smart contracts.
   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 smart contracts 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 contract files under the directories https://github.com/fildaio/CandyDispenser, https://github.com/fildaio/AssetManager and https://github.com/fildaio/Quicksilver. 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 FilDA system should work:

https://docs.filda.io/

https://www.yuque.com/cheyenne-i6vw0/xq2fzw

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

— Comments from Auditor

No vulnerabilities with critical or medium severities were found in the FilDA's codebase. Two vulnerability with high severity were fixed by the team. Four vulnerabilities with low severity were acknowledged by the team, and the team doesn't think they will trigger issues or risks and may make changes in future upgrades.

The FilDA's codebase passed the audit performed by the Fairyproof team.

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 smart contract systems.

03. 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.

04. List of issues by severity

A. Critical

- N/A

B. High

- CandyDispenser/tree/main/contracts/AirDropPool.sol

Logic Error

- AssetManager/tree/main/contracts/Governable.sol

Risky Shared Slot

C. Medium

- N/A

D. Low

- CandyDispenser/tree/main/contracts/AirDropPool.sol

Unsafe Function Call

- CandyDispenser/tree/main/contracts/BlackList.sol

Missing Address Check

- CandyDispenser/tree/main/contracts/PoolManager.sol

Missing Address Check

- AssetManager/tree/main/contracts/PoolHandler.sol

Missing Address Check

05. List of issues by contract file

- CandyDispenser/tree/main/contracts/AirDropPool.sol

Logic Error: High

Unsafe Function Call: Low

- CandyDispenser/tree/main/contracts/BlackList.sol

Missing Address Check: Low

- CandyDispenser/tree/main/contracts/PoolManager.sol

Missing Address Check: Low

- AssetManager/tree/main/contracts/PoolHandler.sol

Missing Address Check: Low

- AssetManager/tree/main/contracts/Governable.sol

Risky Shared Slot: High

06. Issue descriptions and recommendations by contract file

- CandyDispenser/tree/main/contracts/AirDropPool.sol

Logic Error: High

Source and Description:

Line 382: the function `earned` gets a wrong calculation value in certain circumstances, thus causing the function `getReward` in line 390 to be exploited by allowing unauthorized users to claim rewards.

Recommendation:

Consider changing the order of the conditional checks in the function `earned`.

Update: Fixed in c2fd3da3f5e5be973d3658ecf0ff218120068c66 by the team adopting the recommended change.

Unsafe Function Call: Low

Source and Description:

Line 402: the function `notifyRewardAmount` calls different functions to transfer tokens in different circumstances. It is possible that after the function `notifyRewardAmount` is called multiple times the variable `totalSupply` is calculated incorrectly by taking an incorrect input (e.g. when the actual `reward` is greater than 0 it is incorrectly assigned a value equal to 0, thus causing the variable `totalSupply` incorrectly calculated)

Recommendation:

Consider uniformly using `safeTransferFrom` to transfer tokens in all circumstances and adding a check `require(reward > 0)`.

Update: Acknowledged by the FilDA team. The team doesn't think this will cause potential issues or risks and therefore prefers to keep it for now, and may make a change in a future upgrade.

- CandyDispenser/tree/main/contracts/BlackList.sol

Missing Address Check: Low

Source and Description:

Lines 11 and 15: neither the function `addToBlackList` nor the function `addToBlackList` checks the parameter `_target`, thus causing a zero address which is invalid for the function call can be used to call the function.

Recommendation:

Consider adding a check `require(_target != address(0))` for both functions.

Update: Acknowledged by the FilDA team. At the time of writing the team stated that this function would no longer be called, therefore the team doesn't think it will cause any issues or risks thus preferring to keep it for now, and may make a change in a future upgrade.

- CandyDispenser/tree/main/contracts/PoolManager.sol

Missing Address Check: Low

Source and Description:

Line 26: the function `register` doesn't check its parameter `pool`, thus causing a zero address which is invalid for the function call can be registered.

Recommendation:

Consider adding a check `require(pool != address(0))` before the statement `require(!exist(pool), "aready exist!");`.

Update: Acknowledged by the FilDA team. At the time of writing the team stated that this function would no longer be called, therefore the team doesn't think it will cause any issues or risks thus preferring to keep it for now, and may make a change in a future upgrade.

- AssetManager/tree/main/contracts/PoolHandler.sol

Missing Address Check: Low

Source and Description:

Line 33: the function `register` doesn't check its parameter `pool`, thus causing a zero address which is invalid can be registered.

Recommendation:

Consider adding a check `require(pool != address(0))` before the statement `require(!exist(pool), "aready exist!");`.

Update: Acknowledged by the FilDA team. The team doesn't think this will cause potential issues or risks and therefore prefers to keep it for now, and may make a change in a future upgrade.

- AssetManager/tree/main/contracts/Governable.sol

Risky Shared Slot: High

Source and Description:

This contract is inherited by the `HandlerProxy` contract as a proxy contract. The contract defines an address variable `governance` in line 5 but doesn't specify a slot for this variable, thus causing its implementation contracts sharing slot 0 with it and causing the variable `governance` to be modified unexpectedly.

Recommendation:

Consider specifying a slot for the variable `governance` .

Update: Fixed in 63c0767ab1dda45156d4ed7f96213e7d8db6c58f by the team removing this contract file, adding a contract file `GovernableInitiable.sol` and making changes accordingly in the `HandlerProxy` contract.