1Inch Farming Audit

January 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the 1inch's Farming project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Scope

The contracts audited are from the https://github.com/1inch/farming git repository. The audit is based on the commit `d191cc14b526e2eab09c266580b84116b2630da8`. The fixes were added to the commit `1790c23796a0ff630981f5563bf21a312303770a`.

The audited contracts are:

- `contracts/accounting/FarmAccounting.sol`: This library stores and exposes methods for managing farm accounting.
- `contracts/accounting/UserAccounting.sol`: This library stores and exposes method for managing farm's user accounting
- `contracts/BaseFarm.sol:` An abstract contrac implementing the essential farming logic.
- `contracts/ERC20Farmable.sol:` An abstract contract implementing farming functionalities to an ERC20.
- `contracts/Farm.sol:` A contract instantiating and complementing BaseFarm.
- `contracts/FarmingPool.sol:` A contract for creating and deploying farming pools which further allows users to interact.
- `contracts/interfaces/IERC20Farmable.sol`: Interface for IERC20Farmable
- `contracts/interfaces/IFarm.sol`: Interface for Farm
- `contracts/interfaces/IFarmingPool.sol`: Interface for FarmingPool

The scope of the audit is limited to the above-listed files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found a medium issue. Also, several enhancements were proposed.

The medium severity issue was resolved. Also some proposed enhancements were made.

Security Issues

ID	Title	Severity	Status
ME-01	Possible Overflow in StartFarming()	Medium	Resolved

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

Owner

There is only one function that can only be called by the owner and that is `BaseFarm`'s `setDistributor()`. Through inheritance, we find this role in the FarmingPool and Farm contracts.

Distributor

There is only one function that can only be called by the Distributor and that is `BaseFarm's startFarming()`. Through inheritance, we find this role in the FarmingPool and Farm contracts.

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No issues found.

Medium Severity Issues

ME-01 Possible Overflow in StartFarming()

Location:

- `FarmAccounting.sol:38`

An overflow may be triggered in the `startFarming()` function if the sum of the `period` parameter and the `block.timestamp` exceeds 2^{40}, so the `info.finished` entry may have a wrong value because solidity 0.8 checks for overflows in math expressions but not when casting int types. The overflow will trigger in the `uint40(block.timestamp + period)` expression in line 38.

So a distributor in a `BaseFarm` derived contract may corrupt all the farming arithmetic, because the period parameter is passed as is by the `BaseFarm.startFarming()` function. This includes the possibility of triggering a permanent denial of service in the `FarmAccounting._farmedSinceCheckpointScaledMemory`() function, line 24.

Recommendation

Validate if `block.timestamp + period` is less than 2^{40}.

Status

Resolved. The recommended solution was implemented.

Minor Severity Issues

No issues found.

Enhancements

ID	Title	Status
EN-01	Missing Docstrings and References	Not implemented
EN-02	Different and Old Compiler Versions	Not implemented
EN-03	Prefer Defining Constants	Not implemented
EN-04	Details and Best Practices	Implemented

EN-01 Missing Docstrings and References

The reviewed contracts and functions lack documentation. Documentation makes it easier to read the code and helps reviewers to understand its intention. Consider documenting functions using the Ethereum Natural Specification Format (NatSpec).

Similarly, the use of acronyms for naming parameters, as in fpt in `updateBalances()` is confusing and we suggest using full names, e.g., farmedPerToken, instead.

Status

Not implemented. The issue was acknowledged by the development team.

EN-02 Different and Old Compiler Versions

The audited contracts use compiler versions of solidity '^0.8.0' and '^0.8.9'. Since solidity is under heavy development, bug fixes and optimizations are implemented constantly using the latest version is preferred. Considering using the fixed version ‘v0.8.11’.

Status

Not implemented. The development team informed us that as this is a library they expect it to be used with different versions of solidity. They will stick with ^0.8.0.

EN-03 Prefer Defining Constants

Location:

- `FarmAccounting.sol:25,33`
- `UserAccounting.sol:29`
- `ERC20Farmable.sol:120`

Some contracts make use of the uint 1e18 or 1e54. Best coding practices suggest defining constants with these values.

Status

Not implemented. The suggestion was acknowledged by the development team.

EN-04 Details and Best Practices

Location:

- `ERC20Farmable.sol:38`
- `ERC20Farmable.sol:118-132`

The naming of the method `farm()` in `ERC20Farmable` is confusing. Consider using `addFarm`() or other. Also, there is dead code in `ERC20Farmable._getFarmedSinceCheckpointScaled()` function on line 118. The catch and else cases only have code commented out.

Status

Implemented. The `farm()` method was renamed to `join()`. Commented code was reintroduced.

Other Considerations

Abstract Contacts

The `ERC20Farmable.sol` is an abstract contract that is not used elsewhere. A mock included for testing purposes, but a contract instantiating `ERC20Farmable` could be used in some other way. Consider documenting these options and auditing the security of these farm contracts.

Changelog

- 2022-01-24 – Initial report based on commit `d191cc14b526e2eab09c266580b84116b2630da8`.
- 2022-02-03 – Validate fixes and enhancements on commit `1790c23796a0ff630981f5563bf21a312303770a`. We also removed a minor issue. After further analysis with the development team, We came to the conclusion that it is not an issue. At last, one of the other considerations was moved to a new enhancement proposal.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the 1inch Farming project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.