1INCH

1inch Farming

Smart Contract Security Review

Version: 2.0

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the 1inch farming smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the 1inch Farming smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the 1inch smart contracts.

Overview

The 1inch Farming repository provides two types of farming systems that enable the participants to receive reward tokens accrued over time. The first type is a new ERC-20 token with a built-in farming system called `ERC20Farmable`, where token holders are able to join any compatible farms by simply calling a `join()` function.

The second farming type called `FarmingPool` maintains compatibility with existing ERC-20 tokens, where joining a farming pool means transferring the ERC-20 tokens to that farming pool.

Security Assessment Summary

This review was conducted on the files hosted on the 1inch Farming repository and were assessed at commit 879448d. Specifically, the files in scope are as follows:

- `ERC20Farmable.sol`
- `Farm.sol`
- `FarmingPool.sol`
- `accounting/FarmAccounting.sol`
- `accounting/UserAccounting.sol`

Note: the OpenZeppelin and 1inch libraries and dependencies were excluded from the scope of this assessment.

Retesting activities targeted commit 2e57d43.

The manual code review section of the report, focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [**?**, **?**].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 4 issues during this assessment. Categorised by their severity:

- Medium: 1 issue.
- Low: 1 issue.
- Informational: 2 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the 1inch smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
1IFM-01	Farming Amount Limit	Medium	**Resolved**
1IFM-02	Open Solidity Pragma	Low	**Closed**
1IFM-03	Token Transfer Gas Cost While Farming	Informational	**Closed**
1IFM-04	Miscellaneous `1inch Farming` General Comments	Informational	**Resolved**

1IFM-01	Farming Amount Limit		
Asset	`ERC20Farmable.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `FarmAccounting` contract is used by the `Farm` contract to handle computation on `Info`. The `Farm` contract works closely with the `ERC20Farmable` contract to create a token system with farming capability.

The `startFarming()` function in `FarmAccounting` contract is used by the reward distributor to start the farming session. This function requires the input `amount` to be at most `uint176` (based on a check on line [**32**]). However, there is an edge case where this requirement is incorrect when calculating `farmedPerToken (FPT)` value computed by `ERC20Farmable.farmedPerToken()`. The edge case occurs when `FarmAccounting.Info.reward` is sufficiently high and the period since the last checkpoint is sufficiently long, causing `FarmAccounting.farmedSinceCheckpointScaled()` to return a value higher than `1e54` and `ERC20Farmable_lazyGetFarmed()` to return zero. As a result, the the accounting of the farming distributions is incorrect and participants receive less than they are owed.

This means that the safe value for farming depends on `elapsed`, `reward`, and `duration` as identified in `FarmAccounting.farmedSinceCheckpointScaled()` (line [**19**]), where the result should not be more than `1e54` at all times, based on the check done in function `ERC20Farmable._lazyGetFarmed()` (line [**153**]). This indicates that the safest maximum value for farming amount to avoid this edge case is `1e36`.

The issue can be prevented by calling `Farm.startFarming()` to enforce a frequent checkpoint update. However, once the farming period passes, this method no longer works.

Recommendations

The check in `FarmAccounting.startFarming()` should be changed from `uint176` to `1e36`. This will mitigate the issue.

Alternatively, make sure this behaviour is understood. Precalculating reward amounts and farming periods before starting the farming session, in addition to regularly enforcing a checkpoint update can help prevent this edge case. The testing team also recommends providing a safety measure such that the reward token can be retrieved if anything goes wrong after the end of a farming session.

Resolution

The issue has been fixed in commit a57060d. The library `FarmAccounting` imposes `_MAX_REWARD_AMOUNT = 1e42` in function `startFarming()`. This way, the maximum farming reward cannot exceed 1e42. The change also reflects in `ERC20Farmable._lazyGetFarmed()` where the maximum amount now increases to `FarmAccounting._MAX_REWARD_AMOUNT * 1e18` which should no longer be an issue.

The comment in `FarmAccounting.sol` line [**16**] should be updated from `10**54` to `10**60`.

1IFM-02	Open Solidity Pragma		
Asset	`*.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

All of the Solidity contracts use an open pragma that accepts any version of Solidity.

```
pragma Solidity ^0.8.0;
```

A future version of Solidity could potentially be released that contains a bug that could introduce a vulnerability in these contracts.

Recommendations

Pick a trusted, battle-tested version of the Solidity compiler and fix that in the pragma statements.

Resolution

The issue has been acknowledged by the development team with the following comment:

As we position this as a library, we want users to be able to use it with any sane solidity version.

1IFM-03	Token Transfer Gas Cost While Farming
Asset	`ERC20Farmable.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

`ERC20Farmable` is an ERC20 token with a capability of allowing users to join farming contracts and earn rewards. As with any standard ERC20 tokens, it also enables token transfer to other accounts regardless of its farming status. However, transferring tokens while farming potentially requires significantly more gas due to the check performed in the `_beforeTokenTransfer()` function.

Table 1 shows the gas spending of the `transfer()` function while the sender is farming. The percentage of increase shows how much the gas cost is increased compared to the base cost, which is a transfer where both parties are not farming. Based on our experiment, the base cost is 38.624 gas.

No. of Farm	Gas spending	Percentage of Increase (%)
1	44,973	16.47
10	182,469	372.42
100	1,983,895	5,036.43
200	3,934,739	10,087.29

Table 1: Gas Cost Evaluation Based on The Number of Farms Joined

The data shows that the more farms the sender joins, the more expensive the transaction cost will be. The results above assume that the receiver is farming on the first farm of the list (best-case search scenario). The measurements slightly decrease when the receiver does not farm and slightly increase when the receiver's farm is on the last item of the list.

Recommendations

The testing team recommends informing users about the farming impact on their token transfer gas cost. Alternatively, improvements can be made to the farming address retrieval process in line [**111-112**]. The current implementation loads all addresses to the memory before iterating, which incurs significant gas costs. The gas spending increases linearly with the number of farms.

Resolution

The issue has been acknowledged by the development team with the following comment:

We expect the common case to be to simultaneously farm 1-3 farms and we do not expect user to farm more than 10 farms.

1IFM-04	Miscellaneous `1inch Farming` General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings in the `contracts` repository discovered by the testing team that do not have direct security implications:

1. **Lack of Documentation and In-line Comments**
   There is currently no documentation about how the code is used. There is also a low number of comments that explain what functions are for.
2. **Unused Return Value**
   The return value of `startFarming()` in `FarmAccounting.sol` might not be used anywhere.
3. **Renounce Ownership Function**
   Contract `FarmingPool` and `Farm` have `renounceOwnership()` derived from OpenZeppelin library. If used, the contract will lose the `setDistributor()` permission.
4. **Incorrect Revert Message**
   The revert message in line [**16**] on `FarmAccounting.sol` describes the desired condition, not the reverting condition.
5. **Zero Address Prevention**
   The following functions do not prevent zero address as an input.
   5a) The function `FarmingPool.setDistributor()` does not prevent zero address as an input.
   5b) The constructor in `Farm.sol` does not check whether `farmableToken_` and `rewardsToken_` are nonzero addresses.
   5c) The function `ERC20Farmable.join()` and `ERC20Farmable.quit()` does not prevent a user from inputting a zero address in `farm_`.
6. **Zero Deposit/Withdrawal on `FarmingPool`**
   Function `deposit()` and `withdrawal()` do not prevent zero amount. This does not create any security issue, however the transaction wastes gas.

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The issues have been acknowledged or fixed with the following details.

1. Acknowledged.
2. The return value is used in `Farm.sol` and `FarmingPool.sol`. The pattern was adopted from Openzeppelin Solidity library.
3. Acknowledged.
4. Fixed in commit 7760b89.
5. 5a) Acknowledged.
   5b) Fixed in commit b03df37. Extra checks are implemented to prevent zero addresses as inputs.
   5c) Fixed in commit b03df37. Extra checks are implemented to prevent zero addresses as inputs.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_farming_onError                              PASSED   [3%]
test_mint_burn                                    PASSED   [7%]
test_onError                                      PASSED   [10%]
test_join_quit                                    PASSED   [14%]
test_join_quit_zero                               PASSED   [17%]
test_join_quit_multi                              PASSED   [21%]
test_farming                                      PASSED   [25%]
test_farming_entries                              PASSED   [28%]
test_farming_single                               PASSED   [32%]
test_farming_multifarms                           PASSED   [35%]
test_farming_single_refarming                     PASSED   [39%]
test_farming_transfers                            SKIPPED  [42%]
test_spam_normal_use                              PASSED   [46%]
test_spam_heavy_rewards                           PASSED   [50%]
test_spam_find_tipping_point                      SKIPPED  [53%]
test_spam_max_rewards                             PASSED   [57%]
test_spam_excess_rewards                          PASSED   [60%]
test_spam_max_rewards_multi                       PASSED   [64%]
test_spam_find_tipping_point_thousand             SKIPPED  [67%]
test_spam_find_tipping_point_billion              SKIPPED  [71%]
test_spam_find_tipping_point_billion_checkpoint   SKIPPED  [75%]
test_farming                                      PASSED   [78%]
test_claimFor                                     PASSED   [82%]
test_initial_values                               PASSED   [85%]
test_setDistributor                               PASSED   [89%]
test_farming_claim                                PASSED   [92%]
test_farming_exit                                 PASSED   [96%]
test_farming_withdraw                             PASSED   [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

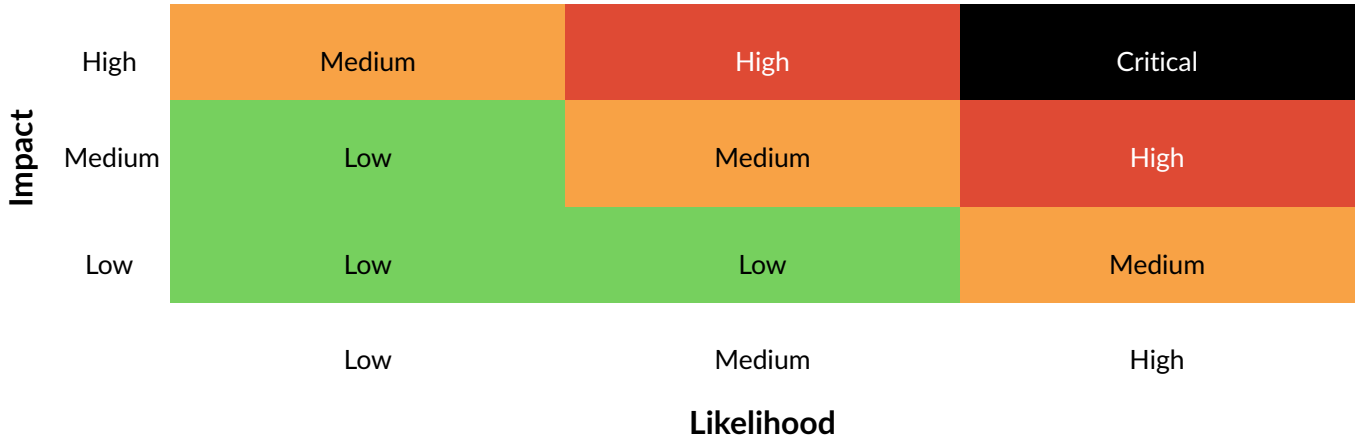

Table 2: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.