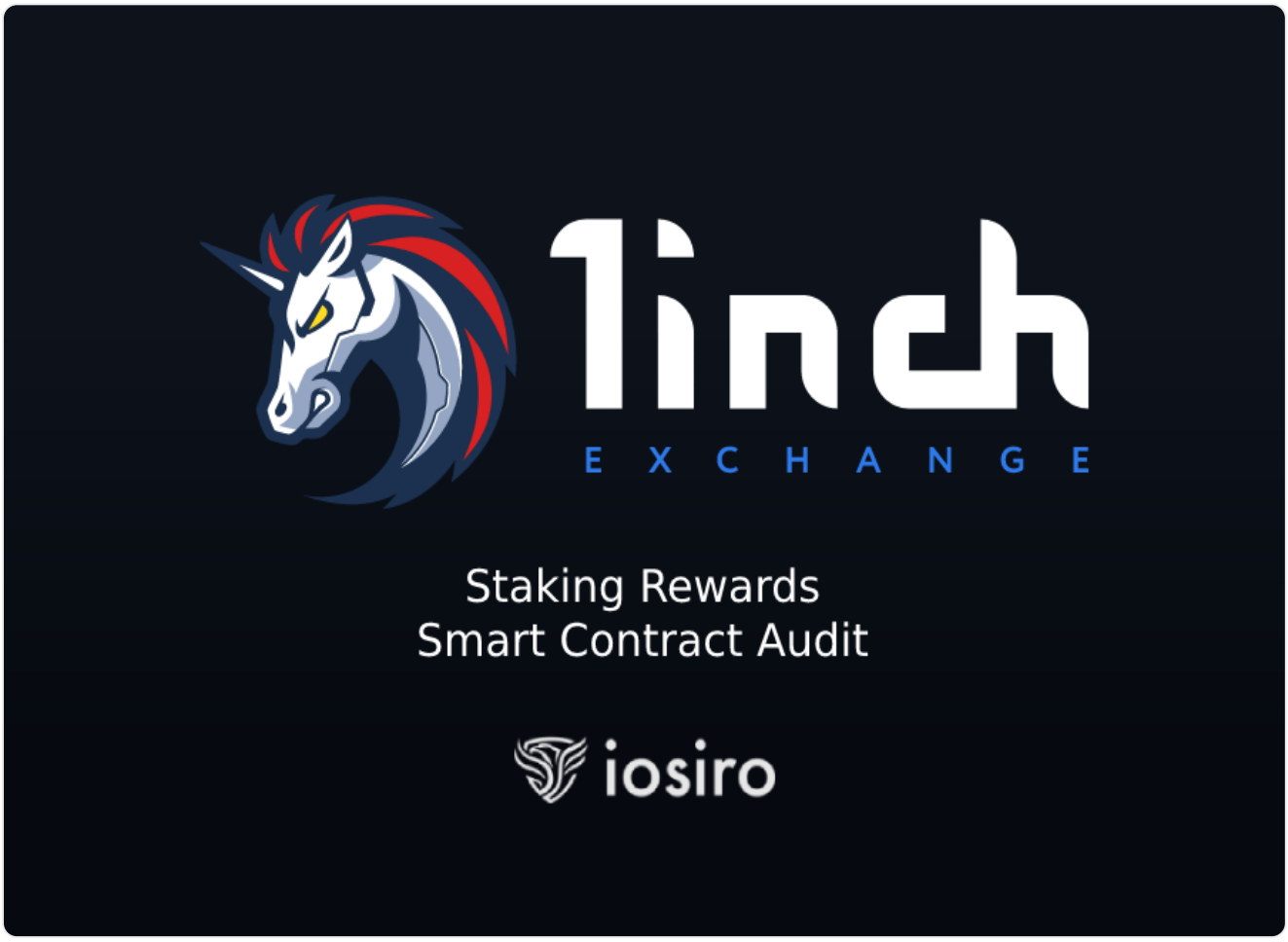
1inch Exchange Staking Rewards Smart Contract Audit



1. Introduction

iosiro was commissioned by 1inch Exchange to conduct an audit on their Staking Rewards smart contracts. The audit was performed between 1 March and 5 March 2021, consuming a total of 5 person days. A review of the remediations to audit findings was performed on 11 March 2021.

This report is organized into the following sections.

- **Section 2 - Executive summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit details:** A description of the scope and methodology of the audit.
- **Section 4 - Design specification:** An outline of the intended functionality of the smart contracts.

- **Section 5 - Detailed findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to better understand the risk exposure of the smart contracts, and as a guide to improving the security posture of the smart contracts by remediating issues identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

The purpose of this audit was to achieve the following:

- Identify potential security flaws.
- Ensure that the smart contracts functioned according to the documentation provided.

Assessing the economics, game theory, or underlying business model of the platform were strictly beyond the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. Strategies that should be used to encourage secure code development include:

- Security should be integrated into the development lifecycle and the level of perceived security should not be limited to a single code audit.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed where possible.

2. Executive summary

This report presents the findings of an audit performed by iosiro on an upgraded version of Synthetix's StakingRewards.sol contract. The upgraded smart contract allowed for multiple staking rewards programs to run simultaneously.

Several low risk and informational issues were identified, some of which related to access control and an overlap in the staking rewards programs. The majority of these findings were addressed following the initial audit, leaving a single low risk and three informational issues open. Comments provided by 1inch Exchange have been added to each open issue. Overall, the code was found to be of a high standard and to accord with the specification provided.

3. Audit details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files is considered to be out-of-scope. Out-of-scope code that interacts with in-scope code is assumed to function as intended and introduce no functional or security vulnerabilities for the purposes of this audit.

3.1.1 1inch smart contracts

Project name: 1inch-liquidity-protocol
Original audit commit: c9c8bc6
Final review commit: 46f35fd
Files: BaseRewards.sol, BalanceAccounting.sol

3.2 Methodology

A variety of techniques were used while conducting the audit. These techniques are briefly described below.

3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high risk areas of the system.

3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a Ganache test environment. Manual analysis was used to confirm that the code operated at a functional level, and to verify the exploitability of any potential security issues identified.

3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. The static analysis results were manually analysed to remove false positive results. True positive results would be indicated in this report. Static analysis tools commonly used include Slither, MythX, as well as Securify. Furthermore, the Remix IDE, compilation output, and linters are also used to identify potential areas of concern.

3.3 Risk ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk** - The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk** - The issue resulted in the code specification being implemented incorrectly.
- **Low risk** - A best practice or design issue that could affect the security of the contract.
- **Informational** - A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

4. Design specification

The following section outlines the intended functionality of the system at a high level.

4.1 Staking rewards

The `BaseRewards` contract was extended to allow for multiple rewards programs to run simultaneously. The original contract facilitated only one staking rewards program, which allowed users to receive staking rewards based on the number of tokens staked and on the duration the tokens were staked. The contract was extended to support multiple programs, allowing the user to receive different staking rewards for the same tokens staked.

The owner had the ability to add these staking reward programs at their own discretion, while the contracts providing the rewards could extend the duration of their rewards program. No functionality to remove rewards programs was implemented.

5. Detailed findings

The following section includes in depth descriptions of the findings of the audit.

5.1 High risk

No high risk issues were present at the conclusion of the audit.

5.2 Medium risk

No medium risk issues were present at the conclusion of the audit.

5.3 Low risk

5.3.1 Owner can change the duration of any `TokenReward`

BaseRewards.sol#L105, *BaseRewards.sol#L110*

Description

Through normal use of the `BaseRewards` contract, the owner could change the `rewardDistribution` address of any of the `tokenRewards`. This means the owner could change the duration of any `TokenReward` instance through the `setDuration` function by bypassing its `onlyRewardDistribution()` access control.

The risk of this issue was limited due to the trust assumptions with respect to the owner role within the contract.

Recommendation

Removing the `setRewardsDistribution()` function will tighten the access control of the `onlyRewardDistribution()` function. Alternatively, the `setRewardDistribution()` function could be modified to only allow the owner to change the `rewardDistribution` address once.

1inch Exchange response

1inch Exchange indicated that they prefer to have the option of changing the `rewardDistribution` address for rewards programs in use.

5.4 Informational

5.4.1 Missing reentrancy protection

BaseRewards.sol#L75

Description

The `getReward` function did not make use of a modifier to protect against potential reentrancy attacks. If a token were to implement a callback (e.g. ERC-223 or ERC-777), the function could in theory be targeted through a reentrancy attack. However, as the checks-effects pattern was used the potential for exploitation was mitigated.

Recommendation

A ReentrancyGuard could be used to protect against reentrancy attacks as a defense in-depth measure.

1inch Exchange response

1inch Exchange noted this finding and committed to bearing it in mind when choosing tokens to use as gifts.

5.4.2 Missing inline comments

BaseReward.sol

Description

The `BaseReward` contract did not include any in-line comments describing the function's purpose, parameters and return values. This information is helpful to developers and auditors who read and interact with the contract's code.

Recommendation

Solidity's NatSpec format for inline code comments should be used to document the contract's functionality.

1inch Exchange response

1inch Exchange noted this finding.

5.4.3 Unbounded for-loop anti-pattern

BaseReward.sol

Description

The `tokenRewards` array stored an array of `TokenReward` structs, each containing information about a certain staking rewards programs. The owner was able to append additional entries to the `tokenRewards` array, but could not remove the entries for completed staking reward programs. This means the `tokenRewards` array would continually grow in the future, which will impact the gas cost of any functions that loop through it, such as the `updateReward()` function.

This issue may impact the overall number of rewards programs that can be run on the contract. The impact of this issue was limited, as the function to add additional reward programs was protected by the owner.

Recommendation

To prevent the for-loop from growing too large, implement functionality to allow the owner to remove certain entries from the `tokenRewards` array. As an example, the `tokenRewards` array could be re-implemented with a mapping and an array of indexes, indicating which entries from the mapping are still in use.

1inch Exchange response

1inch Exchange plans to have three token reward programs at most, which should greatly mitigate this issue.

5.5 Closed

5.5.1 Low reward amount and long duration leads to zero `rewardRate` (low risk)

BaseRewards.sol#L90, *BaseRewards.sol#L94*

Description

When calling the `notifyRewardAmount()` function, the `rewardRate` is instantiated with the amount or updated to reflect the additional reward. Since the reward rate is effectively the total rewards available divided by the duration, if the overall reward amount is lower than the duration, the reward rate will update to zero. This means no users will accrue rewards, and the rewards will remain in the contract.

The risk of this issue was limited as it was unlikely for this issue to occur in the usual lifecycle of the rewards program, provided standard tokens (i.e. with a reasonable number of decimal places), amounts and durations are used.

Recommendation

A require statement should be added to ensure that the reward is more than the duration. This should be done when the reward's `periodFinish` has already been reached, as well as while the reward is still in progress. As an example, this could be achieved by adding `require(reward > duration)` and `require(reward.add(leftover) > duration)` respectively.

Update

This issue was remediated in accordance with the above recommendation in 2164271.

5.5.2 Potential conflict between `TokenRewards` with the same gift (low risk)

BaseRewards.sol#L116

Description

The contract owner could add multiple `TokenReward` programs with the same ERC-20 gift. If the `rewardDistribution` of one `TokenReward` program sent tokens and only distributed a portion of the tokens through the `notifyRewardAmount` function, the remaining tokens would be available for the other `TokenReward` instances to claim.

As an example, if a reward program transferred 1000 USDC as rewards, but only distributed 500 USDC through the `notifyRewardAmount()` function, the remaining 500 USDC could be distributed through another program also rewarding USDC. While the USDC would still be distributed to users if an appropriate duration was set, it would be through a separate program that did not transfer the rewards.

This risk of this issue was limited as the scenario was considered unlikely.

Recommendation

Additional validation should be added to ensure that no two `TokenReward` programs have the same gift. This could be done by looping through the `tokenRewards` array and requiring that the new gift does not match any existing gifts.

Update

This issue was remediated in accordance with the above recommendation in 7415ffb.

5.5.3 Not using `safeTransfer` (low risk)

Description

BaseRewards.sol#L80

The `getReward` function did not make use of the `safeTransfer` function from the OpenZeppelin SafeERC20 library. The `safeTransfer` function abstracts the standard ERC-20 `transfer` function and throws an exception if the transfer returns `false`. This

adds an additional layer of validation in case the ERC-20 gift returns `false` on failed transfers instead of reverting.

Recommendation

The OpenZeppelin SafeERC20 library should be used to wrap all important ERC-20 functions as a defense in-depth measure.

Update

This issue was remediated in accordance with the above recommendation in 853b640.

5.5.4 SafeMath multiplication overflow (low risk)

Description

BaseRewards.sol#L98

The `rewardPerToken` function was subject to a multiplication overflow bug. If a `rewardsDistribution` contract provides a reward amount greater than `uint(-1)/10**18`, the `rewardPerToken` function would overflow. Validation to prevent this bug was added in line 98; however, the validation was insufficient to fully prevent the bug in all circumstances.

This issue was unlikely to be encountered through normal use of the contract, as the value required to trigger the overflow was very high.

Recommendation

To prevent the multiplication overflow, additional validation should be added to ensure that the reward provided is not more than `uint(-1)/10**18`. An example of such a require statement is `require(reward < uint(-1).div(10**18))`.

Update

This issue was remediated in accordance with the above recommendation in 2164271.

5.5.5 Missing function events (informational)

BaseRewards.sol#L105, *BaseRewards.sol#L110*, *BaseRewards.sol#L116*

Description

The newly added functions `setRewardDistribution()`, `setDuration()` and `addGift()` did not emit any events when executed. Events aid in the visibility of contract state changes. This information can be used on the dApp frontend, and could also be useful for users.

Recommendation

The following events should be emitted:

1. The `setRewardDistribution()` function should emit the `tokenReward` index and the new reward distribution address.
2. The `setDuration()` function should emit the `tokenReward` index and the new duration.
3. The `addGift()` function should emit the new gift parameters `gift`, `duration` and `rewardDistribution`.

Update

This issue was remediated in 46f35fd. Note that multiple events are emitted by the `addGift()` function to record the recommended parameters.