Audit Report

Loop v2 Staking Contract

v0.5

December 14, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by Loop Blockchain Pty Ltd to perform a security audit of the Loop staking smart contract.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/Loop-Protocol/Loop_protocol_col5

Commit hash: `0f902abc1981be446445427d1fe2a75a2f28e161`

This audit only covers the staking contracts in the following directory:

- `contracts/loopswap_staking`

As well as imported code from:

- `packages`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The submitted contracts implement the staking functionality of Loop Protocol, a DEX built on the Terra blockchain.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	Transferable reward tokens may block token owner from unstaking and claiming rewards	Critical	Resolved
2	Rewards are lost if user does not claim them during unstake	Critical	Resolved
3	Partial reward calculation in staking contract leads to loss of rewards	Critical	Resolved
4	Users can steal all rewards from the staking contract with very little cost	Critical	Resolved
5	Reward distribution in staking contract may run out of gas and be blocked forever	Major	Resolved
6	User reward query of staking contract returns wrong pending reward amount	Major	Resolved
7	Native tokens support is partially implemented, which could cause inconsistent state and failures	Minor	Resolved
8	Treating LUNA as a special case for tax calculation may lead to problems with Terra protocol updates	Minor	Resolved
9	The owner key being comprised would result in funds locked forever	Minor	Resolved
10	Canonical address transformations are inefficient	Informational	Resolved
11	Storing unused data is inefficient	Informational	Resolved
12	Storing duplicated data is inefficient	Informational	Resolved
13	Use of magic numbers can be error-prone	Informational	Resolved
14	Overflow checks not enabled for release profile in contracts/loopswap_staking/Cargo.toml	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium	We recommend using more idiomatic Rust, such as `unwrap_or_default` instead of defining state variables for unwrapping values. The codebase contains many typos and several comments that are contradicting the implementation.
Level of Documentation	Medium	-
Test Coverage	Low-Medium	The codebase exhibits only sparse unit tests and a lack of integration tests.

Detailed Findings

1. Transferable reward tokens may block token owner from unstaking and claiming rewards

Severity: Critical

The current architecture of the staking contract emits reward tokens (`uLS`), which are fungible and transferable. However, in `contracts/loopswap_staking/src/contract.rs:332`, there is a check that verifies whether the sent token amount is equal to the original amount minted by the sending user. If the current owner of the tokens was not the minter, the check would fail, preventing the user from unstaking and claiming rewards. Additionally, the condition for the user staked time in line `314` will fail if the current owner has not staked tokens before.

Consequently, the current architecture implies that tokens are not fully fungible.

Recommendation

If the intention is to prevent users from transferring reward tokens, we recommend not minting and sending reward tokens, and re-architect the logic to keep track of user balances with a map of accounts in storage. Otherwise, if the intention is that reward tokens should be fungible, we recommend removing any restrictions on fungibility. That implies removing storage of balances and staked tokens as well as time locks on a per user basis. Time locks could be tracked in the token contract itself, where any transfer of tokens could reset the time a user has held tokens.

Status: Resolved

2. Rewards are lost if user does not claim them during unstake

Severity: Critical

If the `execute_unstake_and_claim` function of the staking contract is called with `is_reward_claimed = false`, the user's stake and the total rewards will be reduced independent of a user claiming the rewards or not in `contracts/loopswap_staking/src/contract.rs:417` and `421`, which implies that the user can never again retrieve the reward. Additionally, no one else will ever be able to retrieve the reward, so it will be lost.

Recommendation

We recommend either always sending rewards to users, storing them as pending rewards, or distributing them to other users or a dedicated account.

Status: Resolved

3. Partial reward calculation in staking contract leads to loss of rewards

Severity: Critical

When unstaking from the staking contract, the rewards earned until the current block time are calculated in `contracts/loopswap_staking/src/contract.rs:412` using the amount unstaked, not the total amount staked. Whenever the amount unstaked is less than the total amount staked, the user will lose the rewards on the difference. Those lost tokens are locked forever in the contract.

Recommendation

We recommend using the total amount staked for the reward calculation.

Status: Resolved

4. Users can steal all rewards from the staking contract with very little cost

Severity: Critical

If a user stakes tokens in the staking contract, and then unstakes a part of those without claiming rewards, the user's `USER_REWARD_INFO` storage map entry will be deleted in `contracts/loopswap_staking/src/contract.rs:439`. The user can then unstake again with claiming rewards. Now the user will get an empty `RewardInfo`, which means that the user can claim a proportion of the reward independent of the duration the user has staked, sinche the user's `reward_index` will be zero.

Recommendation

We recommend setting the `reward_index` in line `402` to the `current_reward_index`.

Status: Resolved

5. Reward distribution in staking contract may run out of gas and be blocked forever

Severity: Major

In `contracts/loopswap_staking/src/contract.rs:459`, a loop is used to process daily reward distribution with one iteration per day. Depending on how many days have not had rewards distributed, this loop could run out of gas, with no mechanism to recover.

Recommendation

We recommend removing the loop and instead simply calculating the amount of passed days, and then distributing the full amount at once.

Status: Resolved

6. User reward query of staking contract returns wrong pending reward amount

Severity: Major

The staking contract's `query_user_reward` function returns the user's share of the currently unclaimed rewards, but does not consider historic staking/unstaking. That is caused by the fact that the query does not take the user's `reward_index` into account.

Recommendation

We recommend considering the user's `reward_index` when calculating a user's pending rewards.

Status: Resolved

7. Native tokens support is partially implemented, which could cause inconsistent state and failures

Severity: Minor

In multiple places in the codebase, the `Asset` are `AssetInfo` types are used, which indicates that native tokens may be supported. Support for native tokens is not handled properly though, which would lead to issues. There is also no error returned in most cases, so a call will not revert, but rather lead to an inconsistent state. Instances are:

- If a native token was supported as stakeable, unstaking a native token would lead to no state updates and no return of the staked tokens to the user, and the liquidity/LS token would be kept in the contract due to the condition in line `contracts/loopswap_staking/src/contract.rs:325`.

- Native staked tokens would be skipped during reward distribution due to the condition in line `461`.
- Claiming reward tokens would fail due to the CW20 transfer in line `432`.

This issue is classified as minor since it does not pose a problem currently since native tokens cannot be sent to the contract. The only way to stake is through a CW20 receive message at the moment, see line `84`. Still, the admin could already set a native token as a stakeable and a distribution token, causing the issues described above.

Recommendation

We recommend either adding support for native tokens, or replacing the `Asset` and `AssetInfo` with a simple `Addr` to only support CW20 tokens (and a `Uint128` type if an amount is needed) in the contract.

Status: Resolved

8. Treating LUNA as a special case for tax calculation may lead to problems with Terra protocol updates

Severity: Minor

In `packages/loopswap/src/asset.rs:35`, LUNA is treated as a special case for tax calculations with a hard-coded zero tax. However, this might lead to inconsistencies if Terra changes the LUNA tax policy in a future protocol update. In such a case, the contract would pay the tax, leading to liquidity being used in the case of the pair contracts or operations failing in the router contract.

Recommendation

We recommend treating LUNA the same as other native tokens and querying the tax rate from Terra.

Status: Resolved

9. The owner key being comprised may result in funds locked forever

Severity: Minor

In `contracts/loopswap_staking/src/contract.rs:306`, the contracts check if they are in a `freeze` state. If that's the case, any unstake and claim actions are blocked. In the case that the owner key is lost or compromised, the funds in the contract would be locked forever.

Recommendation

We recommend using a time-lock instead of a freeze and ensuring that the owner key is properly protected, e. g. by using a multi-sig.

Status: Resolved

10. Canonical address transformations are inefficient

Severity: Informational

While previously recommended as a best practice, usage of canonical addresses for storage is no longer encouraged. The background is that canonical addresses are no longer stored in a canonical format, so the transformation just adds overhead without much benefit. Additionally, the codebase is more complicated with address transformations.

Recommendation

We recommend removing any transformation from human to canonical addresses and using the new `Addr` type for validated addresses instead.

Status: Resolved

11. Storing unused data is inefficient

Severity: Informational

The `contract_addr` field of the `StakeableInfoRaw` struct in `packages/loopswap/src/asset.rs:329` does not need to be stored, it can instead be queried within the contract using `env.contract.address`. Storing unused data increases gas consumption and reduces maintainability.

Recommendation

We recommend removing unused stored data.

Status: Resolved

12. Storing duplicated data is inefficient

Severity: Informational

The `token` field of the staking contract's `Config` struct in `contracts/loopswap_staking/src/state.rs:12` contains the address of the stakeable LP token. That same address is redundantly stored in the `StakeableInfoRaw`

struct in the `asset_infos` field of the `STAKEABLE_INFO` storage item. Storing duplicated data increases gas consumption and reduces maintainability.

Recommendation

We recommend removing duplicated stored data.

Status: Resolved

13. Use of magic numbers can be error-prone

Severity: Informational

In several places of the codebase such as in `contracts/loopswap_staking/src/contract.rs:269`, `412`, `452`, `475`, `477`, `487`, `577`, and `578`, magic numbers are used, which can be error-prone and decrease maintainability.

Recommendation

We recommend defining constants and replacing the use of magic numbers by them, to produce more reliable code.

Status: Resolved

14. Overflow checks not enabled for release profile in `contracts/loopswap_staking/Cargo.toml`

Severity: Informational

While set in other packages, `contracts/loopswap_staking/Cargo.toml` does not enable overflow-checks for the release profile.

We only classify this issue as informational since overflow checks are implicitly enabled through the workspace `Cargo.toml`.

Recommendation

We recommend enabling overflow checks in every package, even if no calculations are currently performed in those packages. That prevents unintended consequences when features are added in the future or when the project is refactored.

Status: Resolved