Topia Staking Security Review

Pashov Audit Group

Conducted by: pashov

July 25th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Topia Staking** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Topia Staking

The contracts in this review allow for liquidity providers to lock their liquidity for a fixed duration by staking their LP tokens and receive some yield (rewards) for doing so after the lock period ends.

Use case scenario:

1. Alice provides liquidity to the `TOPIA/ETH` Uniswap V2 liquidity pool, which mints LP tokens to her wallet
2. Alice stakes her LP tokens in `TopiaLpStaking`, choosing a duration (lock time) of the stake
3. A special formula calculates the reward rate & duration based on the weight of each position and the number of positions
4. Alice can only claim rewards after the staking lock time has ended - if she unstakes her LP tokens early she would get 0 rewards
5. A staking position won't yield rewards after its staking duration ends, even if user hasn't unstaked
6. Alice can hold multiple staking positions at the same time

Privileged Roles & Actors

- Staking owner - can manage rewards, the reward/staking tokens and add lockup intervals.
- LP staker - can stake his LP tokens and expect a yield in `rewardToken`s

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **-** **fac6a6a3c751fb01bb3125fce14065099792a579**

fixes review commit hash **-** **277815bcf6150be906fc8006b945c1736f2f71ab**

Scope

The following smart contracts were in scope of the audit:

- `ITopiaLpStaking`
- `TopiaLpStaking`

7. Executive Summary

Over the course of the security review, pashov engaged with Topia Staking to review Topia Staking. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Topia Staking
Date	July 25th, 2023

Findings Count

Severity	**Amount**
Critical	1
Medium	3
Low	1
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Rewards calculation error will result in 0 rewards for users	Critical	Resolved
[M-01]	Multiple centralization vulnerabilities can break the protocol	Medium	Resolved
[M-02]	Rewards can possibly be left stuck in contract	Medium	Resolved
[M-03]	Staking won't work correctly with non-standard ERC20 tokens	Medium	Resolved
[L-01]	Precision loss due to division before multiplication	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Rewards calculation error will result in 0 rewards for users

Severity

Impact: High, as users will lose on their rewards

Likelihood: High, as it will happen almost always

Description

The formula to calculate rewards in `getUserStakeReward` is the following:

```
rewardsPerWeight_.accumulated =
  (
    rewardsPerWeight_.accumulated +
    (
        unaccountedTime * rewardsPerWeight_.rate
    ) / rewardsPerWeight_.totalWeight
  ).toUint96();
```

It is the same in `updateRewardsPerWeight`. The problem with this code is that `(unaccountedTime * rewardsPerWeight_.rate) / rewardsPerWeight_.totalWeight` will round down to zero almost always. Since both `totalWeight` and `rate` are measured in 18 decimals tokens (expected), then as more users stake it is highly likely that the `totalWeight` will grow much more than the static `rate`. The `unaccountedTime` variable just holds how many seconds have passed since the last stake/unstake event, which will always be a pretty small number (1 day is 86400 seconds, which is a small, 5 digit number). Now when `(unaccountedTime * rewardsPerWeight_.rate)` is smaller than `rewardsPerWeight_.totalWeight` this math will round down to zero and `rewardsPerWeight_.accumulated` will stay the same value, meaning no new rewards will be accumulated to be distributed to users anymore.

Recommendations

In both `getUserStakeReward` and `updateRewardsPerWeight` change code like:

```
- (unaccountedTime * rewardsPerWeight_.rate) / rewardsPerWeight_.totalWeight
+ (unaccountedTime * rewardsPerWeight_.rate).divWadDown
+ (rewardsPerWeight_.totalWeight)
```

And also in `getUserStakeReward` change code like:

```
- return getUserStakeWeight(userStake) *
- (rewardsPerWeight_.accumulated - userStake.checkpoint);
+ return getUserStakeWeight(userStake).mulWadDown
+ (rewardsPerWeight_.accumulated - userStake.checkpoint);
```

By using FixedPointMathLib from Solmate.

Discussion

pashov: Resolved.

8.2. Medium Findings

[M-01] Multiple centralization vulnerabilities can break the protocol

Severity

Impact: High, as users might never withdraw their stake or claim their rewards

Likelihood: Low, as it requires a compromised or malicious owner

Description

Multiple methods in `TopiaLpStaking` are centralization vulnerabilities and some can be used to break the protocol for users:

- `setRewardsToken` can be used to update the `rewardsToken` address to a random one, making all reward claims revert, leading to stuck funds
- `setUniswapPair` can be used to update the `uniswapPair` address to a random one, making all stakes/unstakes revert, leading to stuck funds
- `addLockupInterval` can be used to add strange lockup intervals with huge multipliers

The `setRewards` method has multiple problems in itself:

- can be called multiple times, pushing the reward period away with every call
- the `_start` value can be too further away in the future
- the `_end` value can already have passed
- the duration between `_start` and `_end` might be too large (for example 70 years)
- the contract balance of reward tokens is not validated - this can mean users won't be guaranteed to be able to claim their rewards

Recommendations

Make the `rewardsToken`, `uniswapPair` and `lockupIntervals` immutable variables, there shouldn't be a need to change them. Also make sure `setRewards` is callable just once.

Discussion

pashov: Resolved.

[M-02] Rewards can possibly be left stuck in contract

Severity

Impact: High, as it is a value loss for the protocol/its users

Likelihood: Low, as it needs a special scenario to be present

Description

Currently in `TopliaLpStaking::setRewards` we have this comment:

> // This must be called AFTER some LP are staked (or ensure at least 1 LP position is staked before the start timestamp)

While the issue is pointed out here, it is not enforced in a smart contract native manner and the code is still vulnerable. The problem is that if the `rewardsPeriod.start` timestamp has passed and no one has staked, the rewards accumulated until the first stake will be forever stuck in the contract, due to the `stake` method calling `updateRewardsPerWeight` before actually setting the staker's checkpoint.

Recommendations

Add a mechanism to ensure that at least 1 user has staked before `rewardsPeriod.start` - one possible solution is enforcing that there was at least one stake before calling `setRewards` and that `_start >= block.timestamp`.

Discussion

pashov: Resolved.

[M-03] Staking won't work correctly with non-standard ERC20 tokens

Severity

Impact: High, as it can lead to a loss of value

Likelihood: Low, as such tokens are not so common

Description

Some tokens do not revert on failure in `transfer` or `transferFrom` but instead return `false` (example is ZRX). While such tokens are technically compliant with the standard it is a common issue to forget to check the return value of the `transfer`/`transferFrom` calls. With the current code, if such a call fails but does not revert it can result in users unstaking without claiming their rewards, even though they wanted to. Those rewards will be forever stuck in the contract.

Some tokens also implement a fee-on-transfer mechanism, meaning on `stake`, the actual value transferred to the contract's balance won't be `_lpAmount` but `_lpAmount - fee`. This will be problematic on `unstake` as the last users to call it will get their transactions reverted because of insufficient balance in the contract.

Low decimals tokens won't work with `setRewards`, as the method requires at least 10e18 worth of the reward token as a reward per second, which in the case of just a stable coin would be a crazy daily reward rate, which is close to impossible to fulfill for a prolonged period of time. Using highly valued tokens as ETH or BTC would make it even worse.

While those are expected to not be a problem since the README suggests the staking token will be `TOPIA/ETH` Uniswap V2 LP tokens and the reward token will be `TOPIA`, currently the contract has a mechanism to update both tokens and it opens up the attack vector to use ones that are not compatible with the staking contract.

Recommendations

Use OpenZeppelin's `SafeERC20` library and its `safe` methods for ERC20 transfers. For fee-on-transfer tokens, check the balance before and after the deposit (`stake`) and use the difference between the two as the actual transferred value. Consider allowing a lower rewards rate in `setRewards`.

Or you can just remove the `setRewardsToken` and `setUniswapPair` methods.

Discussion

pashov: Resolved.

8.3. Low Findings

[L-01] Precision loss due to division before multiplication

The `estimateStakeReward` method does division before multiplication, which would lead to unnecessary precision loss in Solidity. This will result in incorrect estimation on the front-end for users that want to see a reward projection, showing less than it should have. Make the code so that multiplication comes before division.

Discussion

pashov: Resolved.