FLR Finance

Stake Helper Smart Contract Audit

February 20, 2023

Common Prefix

Overview

Introduction

Common Prefix was commissioned to perform a security audit on FLR Finance's Stake Helper smart contracts, at commit hash df8430a03875d6e896cfaec446ede3d719b01170. The files inspected are the following:

`PoolHelperBase.sol`

`PoolStakeHelper.sol`

`PoolWithdrawHelper.sol`

Description of the protocol

FLR Finance's Stake Helper contracts allow the users to interact with the Farming Pools and FLRX products in a simple, unified way. Users can provide liquidity for an FLRX pair, get the corresponding LP tokens and then deposit them into a farming pool to earn rewards, in a single transaction. Similarly, they can withdraw their deposits from the farming pool, return the LP tokens and get back the pair of tokens they provided as liquidity and swap these tokens to get the tokens they want. The contracts, except for LP tokens, allow the users to deposit and withdraw also to regular (single token) farming pools.

Disclaimer

Note that this audit does not give any warranties on the bug-free status of the given smart contracts, i.e. the evaluation result does not guarantee the nonexistence of any further findings of security issues. This audit report is intended to be used for discussion purposes only. Functional correctness should not rely on human inspection but be verified through thorough testing. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the project.

Findings Severity Breakdown

The findings are classified under the following severity categories according to the impact and the likelihood of an attack.

Level	Description
Critical	Logical errors or implementation bugs that are easily exploited and may lead to any kind of loss of funds
High	Logical errors or implementation bugs that are likely to be exploited and may have disadvantageous economic impact or contract failure
Medium	Issues that may break the intended contract logic or lead to DoS attacks
Low	Issues harder to exploit (exploitable with low probability), issues that lead to poor contract performance, clumsy logic or seriously error-prone implementation
Informational	Advisory comments and recommendations that could help make the codebase clearer, more readable and easier to maintain

Findings

Critical

No critical issues found.

High

No high issues found.

Medium

MEDIUM-1	No minimum output amount set on swaps, therefore they can suffer from high slippage
Contract(s)	*`PoolStakeHelper.sol`*, *`PoolWithdrawHelper.sol`*
Status	**Resolved**

Description

Functions *`PoolStakeHelper::_swapTokens`* and *`PoolWithrawHelper::_swapTokens`* swap user's tokens on FLRX pools using the specified path. The minimum amount out on all these swaps is set equal to the bare minimum, just 1. An attacker could frontrun these transactions and the result would be that the user gets a much smaller output amount than expected. This could also happen not only via an attack, but by an unfortunate ordering of the transactions on the corresponding FLRX pools.

Recommendation

We suggest allowing the user to pass the minimum acceptable output amount to mitigate the frontrunning issue.

Alleviation

The team fixed this issue at commit hash 916cb0096980379da4c6422ec949e01baa8a7229.

Low

LOW-1	Missing case on `_getTokensAndCheckCorrectPath()`
Contract(s)	*`PoolStakeHelper.sol`*
Status	**Resolved**

Description

On `_getTokensAndCheckCorrectPath()` it is checked for each one of `token0path` and `token1path` that the first address of the path is `inToken` and the last the corresponding token of the FLX pair only in the cases that `inToken` is not equal to `token0` or `token1`, but this check should be done every time.

Recommendation

We suggest replacing _gettokensAndCheckCorrectPath() with the following which checks also the missing cases:

```
  function _getTokensAndcheckCorrectPath(address inToken, address pair, address[] memory token0Path, address[] memory token1Path)
       private view returns (address token0, address token1)
   {
         token0 = IPair(pair).token0();
         token1 = IPair(pair).token1();
         require(inToken == token0Path[0]);
         require(token0 == token0Path[token0Path.length - 1]
         require(inToken == token1Path[0]);
         require(token1 == token1Path[token1Path.length - 1]);
    }
```

Alleviation

The team fixed this issue at commit hash [2ad9ae0a171b1abb3a1ac48cfc886ca1ede24697](2ad9ae0a171b1abb3a1ac48cfc886ca1ede24697).

Informational/Suggestions

INFO-1	Gas optimization on `PoolWithdrawHelper::_swapTokens` when `path.length==1`
Contract(s)	*`PoolWithdrawHelper.sol`*
Status	**Resolved**

Description

The call to the FLRX router contract could be avoided in the case of *`path.lenght==1`*, because in this case the expected output token is just the staking token, therefore no swap is needed.

Recommendation

We suggest adding an extra if, before the else branch, to cover this special case and minimize the gas costs.

Alleviation

The team fixed this issue at commit hash 47602aa901d2171638897d17f297ee3af0d12c5f.

INFO-2	For extra flexibility, the protocol could allow the user to provide liquidity using two different tokens
Contract(s)	*`PoolStakeHelper.sol`*
Status	**Resolved**

Description

The function `stakeLpFarm` has an argument inToken, the token the user sends to the contract.The amount of this token is splitted in half and it is exchanged for `token0` and `token1`. With the current implementation, if the user holds both tokens0 and tokens1, he cannot use them directly, without swaps, to provide liquidity via the Stake Helper contracts. Except of the inconvenience, being able to directly deposit the tokens (if he holds them) it could be more efficient than using swap, because swaps, due to slippage, could in principle result in a smaller final amount.

Recommendation

We suggest adding an extra argument on `stakeLpFarm` (instead of just one `inToken`, use `inToken0` and `inToken1`) and allowing the users to send up to two different tokens (choosing inToken0==inToken1 the user gets the current implementation).

Alleviation

The team added this extra functionality at commit hash e31fbb8283130cd84377e69f5762cc39767cbd10.

INFO-3	`withdrawLpToken()` and `withdrawSingleToken()` could also return to the user his rewards from depositing on the farming pool
Contract(s)	*PoolWithdrawHelper.sol*
Status	**Resolved**

Description

Depositors on the Farming Pools collect rewards depending on the total time they kept their tokens on the pool and the reward rate of the pool. `FarmingPool::withdrawAndGetReward()` withdraws the requested amount and also gives his rewards to the user. Stake Helper does not provide this functionality, so the user has to manually collect his reward, by calling `FarmingPool::getReward()`.

Recommendation

We suggest adding this extra feature to make the protocol even more appealing for the users.

Alleviation

The team fixed this issue at commit hash
449c06421d2921b2dcc9675601005b1d6d009186.

INFO-4	Asymmetric implementation of stake and withdraw of LP tokens
Contract(s)	*`PoolStakeHelper.sol`*, *`PoolWithdrawHelper.sol`*
Status	**Dismissed**

Description

Staking and withdrawing of LP tokens are symmetric operations: upon staking the tokens of the user are deposited as liquidity in a FLRX pool, he gets LP tokens and then these are deposited to a farming pool. On withdrawal we follow the exact opposite direction. But this symmetry is not reflected in the implementation of these functions. `PoolStakeHelper::stakeLpFarm` calls the addLiquidity function of FlareXRouter, but the `PoolWithdrawHelper::withdrawLpToken` instead of using `FlareXRouter::removeLiquidity` calls directly `FlareXPair::burn`.

Recommendation

There are no any security (or any other) issues with this implementation, but it is not clear to us why would someone take such an approach (except for some minor gas optimization). We think using the removeLiquidity function of the FlareXRouter would make the code easier to understand, much safer and would ensure that staking and withdraing will remain symmetric operations, even after an update of the FlareXRouter contract.

Alleviation

The team prefers not to change the current implementation, because it is more gas efficient.

INFO-5	Typo in the comment of `_checkApprove()`
Contract(s)	*`PoolHelperBase.sol`*

Status	**Resolved**

Description

There is an extra “be” in the comment

```
if (allowance < type(uint256).max / 2) { //allowance should be enough to support
different tokens, so check to type(uint256).max / 2
```

Recommendation

We suggest correcting this typo.

Alleviation

The team corrected the typo at commit hash cdaff60fc11927feab518e395ca08bbcb40b240e.

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.