Smart Contract Audit
Fluity

coinspect

Fluity

Source Code Audit

Prepared for Fluity Finance • May 2021

v210517

1. Executive Summary

In **May 2021**, Fluity Finance engaged Coinspect to perform a source code review of the changes made on top of a fork of the Liquity Protocol intended to be deployed in the Binance Smart Chain (BSC). The objective of the project was to evaluate the impact of the changes on the security of the smart contracts.

The assessment identified **one medium risk issue and two low risk issues**.

It was found that the immutability of the original `PriceFeed` contract was broken to allow setting the Tellor oracle address in the future once it is available in Binance Smart Chain (see FTY-003). The event `EarningAdd` is never emitted in the contract `LQTYStacking` (see FTY-001). And Coinspect recommends adding new tests to cover the latest functionality introduced by Fluity's changes (see FTY-002).

2. Assessment

The audit started on **May 11**. Fluity Finance provided Coinspect with a diff with all the changes at https://gist.github.com/jfluity/32216082afa16a5420471088df228a37. Coinspect also verified that the diff matches the contracts in the git repository at https://github.com/fluity-finance/fluity-contracts as of commit 172781d2 of **May 1, 2021**.

The following is a summary of the changes.

The `LQTYStacking` contract was modified to keep track of user rewards and penalties. In Liquity Protocol, earnings in LQTY are transferred directly to the user address; in Fluity instead the earnings are sent to the `LQTYStacking` contract where they are locked for a period between 12 and 13 weeks before users can withdraw them without penalties. Earnings can be withdrawn at any time, but withdrawing before the unlocking time incurs a penalty of 50%. State variables `userEarnings`, `userBalance` and `totalBurned` were added to support this new functionality. The changes to the contract also include new events `EarningAdd` and `EarningWithdraw`, and new functions `addEarning`, `withdrawEarning`, `withdrawableEarning` and `earnedBalances`. However, **the event `EarningAdd` that is supposed to be emitted in function `addEarning` is never emitted** (see FTY-001).

Note that the early withdrawal of a given amount of earnings is supposed to result in an equal amount of user tokens burned, but **the tokens are never burned**. In fact, the "burned" tokens remain in the stacking contract's balance.

It is worth mentioning that the array of user earnings (the values of the mapping `userEarnings`) can continue growing if a user does not *fully* withdraw the earnings, and calls to the function `withdrawEarning` become more expensive as the function could potentially iterate over all the items of the array. However the rate of growth is very slow (at most one entry every 3 months) and it is not a problem.

The `LQTYToken` contract's symbol and name were changed from "LQTY" to "FLTY". The initial LQTY allocations were changed to: the allocation for bounties and hackathons was changed from 2M to 5M (and the comment reads for bounties and *marketing* now), the depositor and frontends entitlement was changed from 32M to 35M, and the LP rewards entitlement was changed from 1.33M to 10M.

The `transfer` function of `LQTYToken` originally had a restriction on multisig transfers for the first year, allowing transfers only to `LockupContracts` registered in the `LockupContractFactory`. But **this restriction was removed in Fluity, and it is recommended to consider adding it again**.

Also, in Liquity transfers of `LQTYToken` to `LQTYStacking` were not allowed, but this restriction was also removed in Fluity (this is of course necessary because now rewards are transferred to the stacking contract instead of directly transferred to the user).

The `LQTYToken` contract was also changed to have transfers paused just after deployment, and the deployer address must call the new function `unpause` to start allowing transfers (this can be done only one time, and once unpaused it cannot be paused/unpaused again).

The `LUSDToken` contract's symbol was changed to "FLUSD" and its name to "FLUSD Stablecoin". Like the `LQTYToken` contract, `LUSDToken` was also changed to have transfers paused just after deployment, and in order to start allowing transfers the deployer address must call the new function `unpause` (again, this can only be done once).

Liquidity uses the Chainlink oracle as primary price feed, and the Tellor oracle as fallback. Since the Tellor oracle is not yet available on the BSC network, the `PriceFeed` contract was modified adding a "Tellor admin" (the deployer address) that can call a new function `updateTellor` to set the Tellor oracle address once it is available. This breaks immutability and increases risk, because **if the "Tellor admin" key is compromised this could allow attackers to set their own fake Tellor contract and possibly change the price feed** (see FTY-003).

The `StabilityPool` contract was modified so that the owner can call the new function `setAllowFrontEndKickback` to allow or disallow frontends kickback on the function `registerFrontEnd` (this is a one-time setting done on deployment, the contract's immutability has not been changed).

In the `Unipool` contract a state variable called `lqtyStackingAddress` (that would be better called `'lqtyStacking'` because its not of type `address`) was added and a new event `LQTYStakingAddressChanged`, and an argument were added to the function `setParams` to set the `lqrtStacikingAddress`. The contract Unipool now approves unlimited transfers by the LQTYStacking contract. The function `claimReward` was modified to call `lqtyStackingAddress.addEarning` instead of directly transferring LQTY to `msg.sender`;

In the contract `CommunityIssuance`, the LQTY supply cap was changed from 32 millions to 35 millions. And just like the changes in `Unipool` contract: a state variable called `lqtyStackingAddress` was added and a new event `LQTYStackingAddressSet`, and the function `setAddresses` was added a new argument for the `lqtyStackingAddress`. The contract `CommunityIssuance` now approves unlimited transfers by `LQTYStacking`. The function `sendLQTY` was modified to call `lqtyStackingAddress.addEarning` instead of directly transferring LQTY to `msg.sender`.

Finally, in the `TroveManager` contract, the bootstrap period, during which redemptions are not allowed, was changed from 14 days to 10 days.

3. Summary of Findings

ID	Description	Risk	Fixed
FTY-001	Event EarningAdd never emitted	Low	✘
FTY-002	Insufficient testing of new functionality	Low	✘
FTY-003	Tellor oracle address can be changed multiple times	Medium	✘

4. Detailed Findings

FTY-001	Event EarningAdd never emitted	
Total Risk **Low**	Impact Low	Location LQTYStacking.sol
Fixed ✘	Likelihood High	

Description

The changes that Fluity made on top of the Liquity contracts include a new event EarningAdd that should be emitted in the function addEarning. But the event is never emitted.

Recommendation

Make sure to emit the event EarningAdd in function addEarning.

FTY-002	Insufficient testing of new functionality	
Total Risk **Low**	Impact Medium	Location test/*
Fixed ✘	Likelihood Low	

Description

No new tests have been added for the new functionality on top of the tests originally included in the Liquity Protocol repository.

Recommendation

Create new tests for all the new functionality, including the functions `addEarning`, `withdrawEarning`, `withdrawableEarning` and `earnedBalances` in `LQTYStacking` as well as the `updateTellor` function in `PriceFeed`.

FTY-003	Tellor oracle address can be changed multiple times	
Total Risk **Medium**	Impact High	Location PriceFeed.sol
Fixed ✘	Likelihood Low	

Description

Since the Tellor oracle is not yet available on the BSC network, the `PriceFeed` contract was modified adding a "Tellor admin" (the deployer address) that can call a new function `updateTellor` to set the Tellor oracle address once it is available:

```
function updateTellor(address _tellorCallerAddress) public {
    require(msg.sender == tellorAdmin, "PriceFeed: tellor can't only updated
by tellor admin");
    tellorCaller = ITellorCaller(_tellorCallerAddress);
}
```

This new function increases risk, because **if the "Tellor admin" key is compromised this could allow attackers to set their own fake Tellor contract and possibly change the price feed**.

Recommendation

In order to minimize the risk, the function `updateTellor` can be modified to allow setting the Tellor address only once (as with the unpausing mechanism implemented in `LQTYToken` and `LUSDToken`).

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.