dxDAO Staking Rewards Review No. 2

April, 2021

Chapter 1

Introduction

1.1 Scope of Work

This code review was prepared by Sunfish Technology, LLC at the request of members of dxDAO, an organization governed by a smart contract on the Ethereum blockchain.

This review covers changes to source files implementing "staking rewards" contracts that were originally reviewed by Sunfish Technology, LLC. in March of 2021

1.2 Source Files

This review covers code from the following public git repositories and commits (or commit ranges):

```
github.com/luzzif/erc20-staking-rewards-distribution-contracts
b6f6ff03794c7df29517bfe3048cdc71a7b59b4b -
48a291f924546b183d1c62151d988eb2c37d3216

github.com/luzzif/dxdao-staking-rewards-distribution-contracts
3f70f0147c14d6fd873f068cf14694c2ad4e23fb

github.com/luzzif/dxdao-liquidity-mining-relayer-contracts
80828b50a5c901f1712368a751d8295c645f3a24
```

Within those commit ranges, only the **changes** to the following files were reviewed:

- erc20-staking-rewards-distribution-contracts
  - ERC20StakingRewardsDistribution.sol
  - ERC20StakingRewardsDistributionFactory.sol
- dxdao-staking-rewards-distribution-contracts

  - DXdaoERC20StakingRewardsDistributionFactory.sol
  - SwaprRewardTokensValidator.sol
  - SwaprStakableTokenValidator.sol

- dxdao-liquidity-mining-relayer-contracts
  - DXdaoLiquidityMiningRelayer.sol

This review was conducted under the optimistic assumption that all of the supporting software infrastructure necessary for the deployment and operation of the reviewed code works as intended. There may be critical defects in code outside of the scope of this review that could render deployed smart contracts inoperable or exploitable.

1.3 License and Disclaimer of Warranty

This source code review is not an endorsement of the code or its suitability for any legal/regulatory regime, and it is not intended as a definitive or exhaustive list of defects. This document is provided expressly for the benefit of dxDAO developers and only under the following terms:

Chapter 2

Minor Issues

Issues discussed in this sections are subjective code defects that affect readability, reliability, or performance.

2.1 Threat Model Changes

The most substantial change to the core logic of the `StakingRewardsDistributionFactory` contract is that it now uses OpenZeppelin's `BeaconProxy` to make every `DistributionRewards` contract point to the same code. Although this will make it cheaper to deploy new `DistributionRewards` contracts, it also requires that anyone who launches one of these contracts (or anyone who uses them) also has to trust that dxDAO will not perform an adversarial upgrade. (Since dxDAO controls the code that is executed when someone calls one of the rewards contracts, dxDAO also indirectly controls all of the assets belonging to those contracts.) Previously, the staking rewards logic was immutable, so anyone could determine whether to trust the contract once it had been deployed simply by examining the contract's code and configuration.

Future users of the `DistributionFactory` code may view this change as a bug rather than a feature. It may make sense to allow the eventual `owner` of each `DistributionRewards` contract to opt out of upgrades when the contract is initially deployed.

2.2 Unsimplified Arithmetic

Most of the code covered by this review was changed to require Solidity version `0.8`, which performs arithmetic over- and underflow checks automatically. Consequently, all of the explicitly checked arithmetic was removed from the code. However, that refactoring did not simplify arithmetic expressions as aggressively as it could have. For example, on line 209 of `ERC20StakingRewardsDistribution.sol`:

```
stakedTokensOf[msg.sender] = stakedTokensOf[msg.sender] + _amount;
```

The statement above could be simplified to just:

```
stakedTokensOf[msg.sender] += _amount;
```

There are a number of other places in the code where this simplification can be applied, and it would likely reduce the need for line breaks within individual statements. Reducing the number of inter-statement line breaks and the number of expressions per statement will improve the readability of the code.