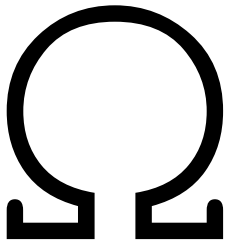
DXdao Staking Rewards Distribution

Final Audit Report

March 29, 2021



Team Omega

teamomega.eth.limo

Summary

DXdao has asked Team Omega to audit the contracts that define the behavior of the DXdao Staking Rewards Distribution contracts.

On November 22, 2021, we delivered an intermediate report. In this report, we described 2 high severity issues - these are issues that can lead to a loss of funds, and are essential to fix. We classified 1 issue as “medium” - these are issues we believe should be definitely addressed. In addition, 16 issues were classified as “low”, and 7 issues were classified as “info” - we believe the code would improve if these issues were addressed as well.

In March, 2022, DXdao provided us with a list of proposed fixes, which we reviewed. All High and Medium issues were resolved.

Severity	**Number of issues**	**Number of resolved issues**
High	2	2
Medium	1	1
Low	15	12
Info	9	2

Scope of the Audit

The audit concerns the contracts committed here:
https://github.com/luzzif/erc20-staking-rewards-distribution-contracts/commit/ee0605eee011394af82b90f407a9393e17e358fc

Specifically, we audited the following contracts:

1. ERC20StakingRewardsDistribution.sol
2. ERC20StakingRewardsDistributionFactory.sol

And the interfaces:

1. IERC20StakingRewardsDistribution.sol
2. IERC20StakingRewardsDistributionFactory.sol

Resolution

The issues mentioned in this report were addressed with commit:
https://github.com/luzzif/erc20-staking-rewards-distribution-contracts/commit/7ce94a917d9bbc6a43bd9c4bec962b4358c4facf

We reviewed these changes, and report the results below under each issue.

Methods Used

Code Review

We manually inspected the source code to identify potential security flaws.

The contracts were compiled, deployed, and tested in a test environment.

Automatic analysis

We have used static analysis tools to detect common potential vulnerabilities. No high severity issues were identified with the automated processes. Some low severity issues, concerning mostly the solidity version setting and functions visibility, were found and we have included them below in the appropriate parts of the report.

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Severity definitions

High	Vulnerabilities that can lead to loss of assets or data manipulations.
Medium	Vulnerabilities that are essential to fix, but that do not lead to assets loss or data manipulations
Low	Issues that do not represent direct exploit, such as poor implementations, deviations from best practice, high gas costs, etc
Info	Matters of opinion

Findings

General

G1. Command for running test coverage is broken [low] [resolved]

Running, as the README instructs, the `yarn test:coverage` command fails due to the fact that the solidity-coverage package is outdated and is missing support for some Solidity 0.8.4 features.
Severity: Low
Recommendation: Update the solidity-coverage package from version 0.7.13 to version 0.7.17 to fix the command. Also configure continuous integration on the github repository (cf G5), so that test and coverage scripts are guaranteed to pass, and the results are always available.
Resolution: The coverage command was fixed.

G2. Test coverage is incomplete [low] [not resolved]

Test coverage is not complete:

```
-----------------------------------------------|----------|----------|----------|----------|----------------|
File                                           |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
-----------------------------------------------|----------|----------|----------|----------|----------------|
 contracts/                                    |    43.67 |    43.13 |    53.19 |    44.27 |                |
  ERC20StakingRewardsDistribution.sol          |    79.46 |    85.53 |    82.61 |    79.89 |... 456,460,461 |
  ERC20StakingRewardsDistributionFactory.sol   |      100 |      100 |      100 |      100 |                |
  StandaloneERC20StakingRewardsDistribution.sol |        0 |        0 |        0 |        0 |... 444,445,446 |
-----------------------------------------------|----------|----------|----------|----------|----------------|
All files                                      |    43.67 |    43.13 |    53.19 |    44.27 |                |
-----------------------------------------------|----------|----------|----------|----------|----------------|
```

Specifically, in `ERC20StakingRewardsDistribution.sol`, a number of error conditions (SRD09, SRD11, SRD16, SRD22), and the functions `renounceOwnership`, `getClaimedRewards`, `exit`, and most of `claimableRewards`, are not or incompletely tested.
Severity: low
Recommendation: Add `StandaloneERC20StakingRewardsDistribution.sol` to the list of files to ignore in the coverage report. Extend tests to cover the rest of the codebase.
Resolution: The issue was not resolved - test coverage of `ERC20StakingRewardsDistribution.sol` remains below 80%.

G3. Contracts do not inherit their own interfaces [low] [resolved]

Both `ERC20StakingRewardsDistribution` and `ERC20StakingRewardsDistributionFactory` contracts do not inherit from their interfaces. This means deviations from the interfaces will not be

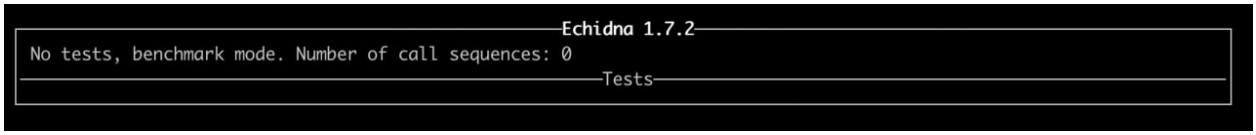
caught automatically by the compiler, which increases the chance of making mistakes, such as those mentioned in issues ID1 and IF1.
Severity: Low
Recommendation: Update the contracts to inherit from their interfaces.
Resolution: The issue was resolved.

G4. Lack of documentation - inline or otherwise [info] [not resolved]

Across the whole codebase, there is very little documentation of the code, and no separate documentation of the code was made available. This makes it hard to know if the code is properly implementing the intentions of the developers, and makes the code harder to reason about, and would make modifications or use of the code by others more difficult.
Severity: Info
Recommendation: Add proper documentation for each function and in lines where non-straightforward operation/ calculation is being done.
Resolution: This issue was not resolved.

G5. Fuzzing tests give empty output [info] [not resolved]

After following the instructions in the README file, the fuzzing test seems not to be working, and no tests are being executed.



Severity: Info
Recommendation: Fix the tests or the documentation for running them.
Resolution: This issue was not resolved.

G6. Configure continuous integration [info] [not resolved]

There is no continuous integration configured - i.e. test and code quality scripts are not run automatically on each deployment.
Severity: Info. Even if this issue does not directly affect the quality of the code itself, having continuous integration would have avoided issues G1, G2 and G5.
Recommendation: Configure automatic tests, using github actions for example
Resolution: This issue was not addressed.

G7. A floating pragma is set instead of a fixed pragma [info] [resolved]

The Solidity pragma version used in the contracts is set as floating: `pragma solidity ^0.8.4`. Setting it to a fixed version will make the compilation step more deterministic, and make it easier for third parties to verify the deployed bytecode.
Severity: Info
Recommendation: Remove the `^` symbol in the pragma definition to make the solidity version fixed and ensure bytecode consistency.
Resolution: The Solidity files now have a fixed compiler version set.

G8. Use the latest solidity version [info] [resolved]

The contracts use Solidity version 0.8.4, while the latest version is 0.8.10.
Severity: Info
Recommendation: Use the latest Soldity version, currently 0.8.10.
Resolution: The contracts now use solidity version 0.8.12

G9. Set the owner of the ERC20StakingRewardsDistribution directly [info] [not resolved]

The current way the owner of the `ERC20StakingRewardsDistribution` contract is set is by first making the `msg.sender`, i.e. the factory, as the owner, and then in the factory, transferring the ownership in the line after. Gas costs for duplicating the owner setting plus the extra costs of calling the `transferOwnership` function could be saved by directly setting the owner on initialization.
Severity: Info
Recommendation: Set the correct owner while initializing by either passing the owner as a parameter to the initialize function of the `ERC20StakingRewardsDistribution` contract.
Resolution: DXdao has decided to not make any changes to the Factory contract, which was already deployed.

ERC20StakingRewardsDistribution.sol

D1. Unbounded iteration over rewards could result in functions running out of gas [high] [resolved]

The length of the rewards array is unlimited. This means that if the rewards array is large, functions that iterate over all items in the array can run out of gas.
This problem is particularly relevant for those functions that do call the functions `balanceOf` and `transfer` on each of the ERC20 contracts in the rewards list, as there is no way of knowing *a priori* how much gas these calls will consume.

The vulnerable functions are:

- `stake`
- `withdraw`
- `addRewards` (does external calls)
- `cancel` (does external calls)
- `recoverUnassignedRewards` (does external calls)
- `claim` (does external calls)
- `claimAll` (does external calls)
- `consolidateReward` (an internal function that is called by all of the functions above)

Severity: High - if a rewards distribution is deployed with too many reward tokens, or with a reward token that uses more gas than expected, funds sent to the contract will be irretrievably lost.
Recommendation: Enforce, in the initializer, a cap on the maximum number of rewards allowed. This resolves the issue for those functions that have calculable gas costs (i.e. stake and withdraw, and consolidateReward). Given the the fact that the gas consumption of external calls cannot be known in advance, we recommend that either:

- Add an argument to the functions `recoverUnassignedRewards`, `claim`, `claimAll` and `cancel`, that allows users to control which tokens should be transferred.
- Implement a payment "pull" mechanism: i.e. keep, for each user, balances of "withdrawable tokens" that are updated in the functions above, and implement a separate function with which users can withdraw their balance.

Resolution: In the initializer, a hard limit of no more than 5 reward tokens is enforced. This resolves the issue for those functions that have deterministic gas costs, such as `stake` and `withdraw`. In addition, the `claim` function now skips tokens which were not requested to be redeemed (i.e. amount specified is 0) and two new functions were added that operate on specific tokens:
`recoverSpecificUnassignedRewards(address _token)`, `claimAllSpecific(address _token)`

D2. A non-standard, malicious, or disabled token in the rewards list can prevent users from withdrawing their rewards [high] [resolved]

The functions `cancel`, `recoverUnassignedRewards`, `claim`, and `claimAll` each iterate over all token addresses in the rewards array, and call the `transfer()` function on these contracts (using the `SafeERC20` wrapper from OpenZeppelin). There are many ways such a call can fail - for a trivial example, the contract could be paused. If that happens, users will not be able to claim their rewards for *any* of the tokens.
Severity: High - even if the rewardTokens are chosen by the contract deployer and so, in theory, should be checked for the above problems, there is a real risk here that funds will be locked in the contract.
Recommendation: See issue D1 for two ways of avoided these problems
Resolution: This issue was resolved by skipping the `transfer` of tokens on `claim` when a 0 amount is specified, as well as by adding the following new functions that operate on specific tokens: `recoverSpecificUnassignedRewards(address _token)`, `claimAllSpecific(address _token)`

D3. Check that each reward token is used only once [medium] [resolved]

It is possible to initialize the contract with an array that contains the same token address multiple times. Adding multiple instances of the same contract in the rewards array may lead to problems in other places of the code, where it is assumed that reward tokens will occur only once in the array.
For example, when there are duplicate rewards contracts, the check on line 121 in the initializer does not guarantee that the contract owns enough tokens to cover all the rewards.
Severity: Medium
Recommendation: Check that in the `initialize` function each token address is unique, for example by storing the list of contract addresses in an array, and the `Reward` objects in a mapping from contract addresses to `Reward` objects, making sure the mapping entry is empty before writing to it.
Resolution: The `initialize` function now checks for duplicates.

D4. Unused variable secondsDuration [low] [resolved]

The `secondsDuration` variable is set in the `initialize` function but is never used.
Severity: Low - some gas can be saved
Recommendation: Remove the `secondsDuration` variable. If it is needed in the frontend, it can be calculated from the `startingTimestamp` and `endingTimestamp` variables.
Resolution: The variable was removed

D5. Superfluous contract calls in reward distribution functions [low] [resolved]

In the functions `recoverUnassignedRewards`, `claim`, and `claimAll`, the transfer functions and state updates are calculated also when the amount to transfer is 0.
The functions contain a check that the amount of the token to transfer is greater than 0, but that is used only to throw an error if none of the values in question is > 0.
Severity: Low
Recommendation: Do not call transfer funds (or do empty calculations) when the amount to transfer is equal to 0. Also consider removing the bookkeeping for keeping track that at least one token had a non-zero amount and the corresponding `require` statement, as this does not seem to be particularly useful functionality but does raise the gas costs of the functions significantly.
Resolution: The non-zero check before the transfer call was implemented

D6. Disable all functionality after contract is canceled [low] [resolved]

One line 171, in the `cancel()` function, the `canceled` flag is set to true. This disables most of the user functionality (such as staking or claiming), but not all the admin functions (for example, `addReward` can still be called).
Severity: Low
Recommendation: As the contract is (presumably) not meant to be used anymore, consider using the `selfdestruct(owner)` after all rewards have been reclaimed by the owner to send any ether held by the contract to the owner and remove the contract from the blockchain. This will also simplify the logic in the contract. If you do not want to call `selfdestruct` here, we recommend in any case to disable the addRewards function when the distribution is canceled.
Resolution: Our recommendation to use "`selfdestruct`" here was not followed, but the `addRewards` function now also requires that the `canceled` flag is set to false.

D7. Consider changing the access of recoverUnassignedRewards [low] [resolved]

The `recoverUnassignedRewards()` function sends any rewards that are available before any tokens are staked (and so could not be sent to stakers) to the owner of the contract. The function is protected by the `onlyStarted` modifier, which requires that the distribution is initialized, has started, and has not been canceled.
Severity: Low
Recommendation: We recommend removing this modifier, as no unassigned rewards exist before distribution has been started. We also recommend to instead add the `onlyOwner` modifier, as the owner may want to keep control over when these unassigned funds are sent.
Resolution: The `onlyOwner` modifier was added to the function.

D8. addRewards cannot be called before the start time has arrived [info] [not resolved]

Any call `consolidateReward` before the distribution has started will fail on line 310-311, where the `lastPeriodDuration` is calculated. This means that a call to `addRewards` functions will fail if called before `startTimeStamp`. This limitation seems unintended.
Severity: Info
Recommendation: Refactor the code to remove this restriction. Also, add tests to cover the `addRewards` function, which currently is not tested at all.
Resolution: None of our recommendations were implemented.

D9. Funds are not retrievable after calling addRewards when staking is canceled [low] [resolved]

`addRewards` is still callable even if staking has been canceled. In this case, it is not possible to retrieve the funds that are added.
Severity: Low
Recommendation: Disable the option to call `addRewards` if the staking has been canceled.
Resolution: The issue was resolved.

D10. Define functions as external where possible [low] [resolved]

The functions `claimableRewards`, `renounceOwnership`, `transferOwnership` should be declared as `external` instead of `public` for clarity, and to save some gas.
Severity: Low
Recommendation: Make these functions external.
Resolution: The issue was resolved as recommended.

D11. Loop in early exit of claimableRewards function is unnecessary [low] [resolved]

The `claimableRewards` function has a loop in the case that the contract wasn't initialized, or that the staking time didn't yet start, that sets the array of results to all zero values before returning it. This initialization logic is not necessary, as the array is already full with zero values
Severity: Low - some gas can be saved
Recommendation: Remove the loop in claimableRewards, lines 352 - 354.
Resolution: The issue was resolved.

D12. Superfluous modifier on exit() function [low] [resolved]

The `exit()` function is a helper function that allows users to claim their rewards and withdraw their stake in a single transaction. It is protected by the "`onlyStarted`" modifier, but so are both of the functions it calls. Removing the modifier will maintain the same behavior and save some gas.
Severity: Low
Recommendation: Remove the "`onlyStarted`" modifier from the `exit()` function.
Resolution: The issue was resolved.

D13. Significant code duplication across multiple functions [info] [not resolved]

Across the codebase, multiple functions share a very significant amount of code. This unnecessarily complicates the code, violates the DRY coding principle, and increases the deployment gas costs. Functions which share a significant amount of duplicated code include: the `consolidateRewards` and `claimableRewards` functions, the `claim`, and `claimAll` functions.
Severity: Info
Recommendation: Avoid code duplication by combining functions with similar functionality into a single function with special edge cases, or by extracting duplicated code into dedicated functions called where the relevant duplicated code used to be.
Resolution: It was decided to not make any significant code changes in such a late stage of the project, unless truly necessary, and so the code duplication was not removed. Some new functions to address other issues were, however, added, that contain more duplicate code, namely the functions `claimAllSpecific`, `recoverRewardsAfterCancel`, `recoverRewardAfterCancel`, and `recoverSpecificUnassignedRewards` functions.

D14. Unnecessary require that reward must be > 0 in initialize() [info] [not resolved]

On line 119, it is required that the amount of the reward is larger than 0:

```
require(_rewardAmount > 0, "SRD05");
```

As rewards can also be added later using addReward, this requirement seems unnecessarily strict - the use case in which the distribution is set up before any tokens are transferred is excluded.
Severity: Info
Recommendation: Consider removing this requirement.
Resolution: Our recommendation was not followed.

IERC20StakingRewardsDistribution.sol

ID1. Interface does not correspond with the implementation [low] [resolved]

The `IERC20StakingRewardsDistribution` interface defines an external function `consolidateReward()`, which is not implemented in `ERC20StakingRewardsDistribution`.
Severity: Low
Recommendation: Remove this function from the interface definition.
Resolution: The issue was resolved.

ERC20StakingRewardsDistributionFactory.sol

F1. Factory owner could steal the rewards of a distributions being created by changing the implementation [low] [not resolved]

The owner of the Factory contract could steal the rewards of a distribution being deployed by front-running a createDistribution call and changing the implementation that the deployer expected to a malicious one. This means it is not enough for users of the factory to verify the implementation, but they must also trust the factory owner.
Severity: Low
Recommendation: If in the final deployment, the owner is the DXdao DAO, it could be considered trusted and in any case not capable of front-running. However, a simple fix would be to add the implementation address as an argument to the `createDistribution` call and check if it is correct in the function body.
Resolution: The issue was not resolved - as DXdao is considered to be a trusted party. Also, it was decided that the already deployed Factory contract will not be upgraded unless truly necessary.

F2. Factory owner could steal the rewards of distributions by pausing staking [low] [not resolved]

The owner of the Factory contract could steal the rewards by staking even a single token right after a distribution reward was created, then calling the `pauseStaking` function to prevent others from staking and subsequently claiming all the rewards.
Severity: Low
Recommendation: Since the owner is the DXdao itself, it could be considered trusted and not taking further action would not be an issue. However, if you wish to address it, you could allow distribution contracts to opt-out of this pausing by the factory mechanism so as to reduce trust in the factory owner.

Resolution: The issue was not resolved - as DXdao is considered to be a trusted party. Also, it was decided that the already deployed Factory contract will not be upgraded unless truly necessary.

IERC20StakingRewardsDistributionFactory.sol

IF1. Interface does not correspond with the implementation [low] [resolved]

The `IERC20StakingRewardsDistributionFactory` interface is not implemented as such in the `ERC20StakingRewardsDistributionFactory`. Differences are:

- The interface defines the return type of `distributions()` as `address`, but it is of type `IERC20StakingRewardsDistribution`
- The `ERC20StakingRewardsDistributionFactory` contract does not implement the `upgradeTo` function defined in the interface

Severity: Low
Recommendation: Fix the discrepancies.
Resolution: The issue was resolved.