OpenZeppelin | security

OGV and OGN Merge Audit

ORIGIN

May 30, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-04-29 To 2024-05-03
Languages	Solidity

Total Issues	11 (5 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (1 resolved)
Notes & Additional Information	9 (4 resolved)

Scope

We audited the OriginProtocol/ousd-governance repository at commit d0fd40f.

In scope were the following files:

```
contracts
├── ExponentialStaking.sol
├── FixedRateRewardsSource.sol
├── Migrator.sol
└── OgvStaking.sol
```

System Overview

On 9 April 2024, OGN holders approved merging with OGV, Origin's DeFi governance token. One week later on 16 April 2024, OGV holders also approved this merger. This audit investigates the plans to facilitate this merger which includes OGN adopting the staking mechanics and features of OGV. Namely, users are able to stake their OGN for a period of time and receive a new token called xOGN which allows them to vote in DAO proposals. The concepts of how much xOGN tokens stakers will receive and their voting power over time are documented in pull request #77.

The migration consists of several components.

- The `Migrator` contract will exchange users' OGV for OGN and can also migrate users' staked positions across as well. The exchange of tokens will take place at a fixed exchange rate which is specified in the aforementioned proposals.
- The `FixedRateRewardsSource` contract will accrue OGN tokens for OGN stakers.
- The `OgvStaking` contract, where users would stake their OGV tokens in the past, will no longer have the ability to stake new positions. However, users will still be able to end stakes and withdraw.
- To mitigate the outsized influence of the first token migrators, a multi-sig wallet will have sole power over the DAO until they deem the migration sufficient enough to be representative of the community.

Security Model and Trust Assumptions

The `Migrator` contract exchanges users' OGV for OGN at a fixed exchange rate. This fixed rate may be different from the current market's implied exchange rate and would encourage arbitrageurs to capture the difference. However, the small market size could lead to inefficiencies and this arbitrage opportunity could remain unexploited.

During the migration procedure, the timelock is going to be controlled by a multi-sig wallet until enough xOGN tokens are staked. Once this happens, control is transferred to the new xOGN

governance system. This mitigates the risk of the attacker holding the majority of either veOGV or xOGN tokens and executing a malicious proposal.

The `ExponentialStaking` contract relies on the `prb-math` library for floating point calculations. We assume this library works accurately as described.

Privileged Roles

The `FixedRateRewardsSource` contract has the strategist and governor roles. The strategist can change the rate at which the rewards are released. The governor can change the rate, the strategist, and the target address which collects the rewards.

The `Migrator` contract has the governance role which can start the migration and, once the migration has finished, transfer the remaining OGN tokens to an arbitrary address.

Low Severity

L-01 Last Rewards Cannot Be Collected

The `collectRewards` function of the `FixedRateRewardsSource` contract will attempt to release funds at a fixed rate per second regardless of how many tokens it owns. For the caller who attempts to collect the last remaining funds in the contract, the released amount may surpass the owned amount. Meaning, any call to collect rewards will revert and the owned tokens will never be released. It is possible to release owned tokens if the exact difference between released and owned tokens is transferred prior to collecting the rewards.

Consider adding logic to calculate the released tokens as the minimum between the calculated amount and the owned amount.

Update: *Resolved in pull request #413. The Origin Protocol team stated:*

> *We also detected this, and have already done a PR with this change during the audit.*

L-02 Missing Docstrings

Throughout the codebase, there are multiple code instances that do not have docstrings.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. This includes all public variables as well. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *We will leave these as they are.*

Notes & Additional Information

N-01 Code Clarity

Consider changing the double parentheses to single parentheses in line 269 of the `ExponentialStaking` contract.

Update: *Resolved in pull request #421. The Origin Protocol team stated:*

> *We have removed the double parentheses and simplified the calculation on this line.*

N-02 Functions Rendered Ineffective by `transfer`

The `ExponentialStaking` contract has its ERC-20 transfer functions disabled. However, there are other functions like `approve`, `increaseAllowance`, `decreaseAllowance`, and `permit` that are not disabled but do not do anything useful given the absence of the ERC-20 `transfer` functions.

Consider disabling the aforementioned functions to save gas during deployment and reduce the attack surface.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *We will keep these functions as they are. It is a trade-off here between bytecode simplicity and Solidity code simplicity, and for this contract, we value the code simplicity more.*

N-03 Lack of Input Validation

Within the `stake` function of the `ExponentialStaking` contract, a `lockupId` of less than `-1`, or a value greater than or equal to the array length of `lockups[to]`, will cause the function call to revert on line 116 with an "index out-of-bounds" error.

Consider validating the `lockupId` input value to ensure that it is either equal to `NEW_STAKE` (`-1`) or is within the length of the `lockups` array for the input `to` address.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *We will leave this as is given that the only impact is the error message shown from the contract and we would be duplicating checks that are already happening at the Solidity level.*

N-04 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

For conciseness and gas savings, consider replacing `require` and `revert` statements in `ExponentialStaking.sol` and `OgvStaking.sol` with custom errors.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *We will leave these as they are.*

N-05 Constant Not Using `UPPER_CASE` Format

In `ExponentialStaking.sol`, the `maxStakeDuration` constant is not declared using the `UPPER_CASE` format.

According to the Solidity Style Guide, constants should be named using all capital letters with underscores separating the words. For better readability, consider following this convention.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Keeping this for backwards compatibility reasons.*

N-06 Non-Explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity file or when inheritance chains are long.

Throughout the codebase, global imports are being used:

- The import "OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/IERC20.sol"; import in `FixedRateRewardsSource.sol`
- The import "OpenZeppelin/openzeppelin-contracts@4.6.0/contracts/token/ERC20/extensions/ERC20Burnable.sol"; import in `Migrator.sol`
- The import "./Governable.sol"; import in `Migrator.sol`

Following the principle that clearer code is better code, consider using the named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #424.*

N-07 State Variable Visibility Not Explicitly Declared

Throughout the codebase, there are state variables that lack an explicitly declared visibility:

- The `YEAR_BASE` state variable in `ExponentialStaking.sol`
- The `NEW_STAKE` state variable in `ExponentialStaking.sol`
- The `YEAR_BASE` state variable in `OgvStaking.sol`

For improved clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #423. The Origin Protocol team stated:*

> *Updated to be explicit.*

N-08 Unused Errors

Throughout the codebase, there are unused errors:

- The `InvalidRewardRate` error in `FixedRateRewardsSource.sol`
- The `MigrationIsInactive` error in `Migrator.sol`
- The `MigrationNotComplete` error in `Migrator.sol`

To improve the overall clarity, intentionality, and readability of the codebase, consider either using or removing any currently unused errors.

Update: *Resolved in pull request #416 and pull request #422. The Origin Protocol team stated:*

> *Removed these unused methods.*

N-09 Gas Griefing Vector

The `stake` function of the `ExponentialStaking` contract is intended to allow gifts but is not intended to control others' stakes or rewards. However, there is one case where this does not hold. If a user wants to stake their rewards, their transaction can be front-run by a 1 wei gift so that the rewards are sent to the user's address and the original stake transaction fails.

While the user can still stake their rewards, doing so will require them to make another transaction. We decided to raise this as an informational finding because there is no clear incentive to exploit this behavior, and even if exploited, there is a recovery plan. Furthermore, mitigating this issue will require changing how the rewards work which may not be feasible in this case.

Update: *Acknowledged, not resolved. The Origin Protocol team stated:*

> *Acknowledged.*

Conclusion

This audit covered the OGV and OGN merger which introduces a migration procedure to migrate from OGV to OGN and from veOGV to xOGN, incorporates new OGN staking contracts, and disables the OGV staking functionality.

A few low-severity issues related to edge cases were identified and several recommendations centered around best practices were made. Overall, the codebase was found to be well-written. Throughout the audit, the Origin team remained very responsive, providing valuable insights and detailed explanations.