Origin Governance Audit

OPENZEPPELIN SECURITY | JULY 1, 2022

Security Audits

The Origin Protocol team asked us to review and audit their Origin governance system. We looked at the code and now publish our results.

Table of Contents

Summary

The OpenZeppelin team audited a governance system for OriginProtocol. The contracts allow users to stake their `OGV` tokens to receive voting power and be rewarded with additional `OGV` tokens.

Type
: Governance

Timeline
: From 2022-05-16
: To 2022-05-20

Languages
: Solidity

Total Issues
: 11 (8 resolved, 2 partially resolved)

Scope

We audited the OriginProtocol/ousd-governance repository at the `2b9761606d4ac4062b69367ebbad88220cea45ce` commit.

In scope were the following contracts:

```
contracts/
├── Governance.sol
├── OgvStaking.sol
└── RewardsSource.sol
```

Out of Scope

All other files that are not specifically mentioned in scope were not audited. This is especially relevant for the `RewardsSource` contract that relies on the `Governable` contract, which

System Overview

The system enables users to participate in the governance of the Origin Protocol through staking their `OGV` tokens, allowing users to earn more `OGV` as a staking reward.

The staking contract `OgvStaking` allows OGV governance tokens to be staked to obtain non-transferable `ERC20Votes`-based tokens called `OGVe` which can participate in a `GovernorBravo`-compatible governance vote. The minimum and maximum lockup periods are 7 and 1461 days, respectively. In addition, each staker earns `OGV` tokens at a pre-configured daily total rate set per time interval and emitted from the `RewardsSource` contract.

The `OGVe` tokens are awarded based on a constant inflation factor of 80% per year in relation to the end of the staking lockup period. This mechanism is designed to limit the voting power of stakes which are no longer in lockup.

Security Considerations

- Due to an initial voting delay of 1 block in the governance contract, we assume a delay between the deployment of `OgvStaking` and the transfer of assets or privileges to `Governance`, such that there is at least sufficient time for stakers to have received and delegated their voting shares before the governance contract becomes active. Further, we assume a cancellation of all proposals in the timelock queue directly prior to the transfer of assets or privileges to `Governance` to prevent the execution of proposals which have been passed before completion of the voting shares distribution.
- Based on the disabled transfer functionality of `OGVe` and the minimum staking duration of 7 days for `OGV`, flash-loan based governance attacks are mitigated.
- Due to a lack of external calls outside of the contract ecosystem, the functions within `OgvStaking` appear inherently safe against reentrancy. However, the `OGV` token is based on an upgradable proxy. A future upgrade introducing transfer-hooks with user-controllable data could render the `collectRewards` function vulnerable to reentrancy.
- The `PRBMathUD60x18` contract of the `paulrberg/prb-math` library is assumed to operate correctly if all operands and results can be expressed as a number with a 60 digit integer field and 18 digit fractional field.

High Severity

Extending the staking duration discards rewards

In the `OgvStaking` contract, updating a user's rewards is a two step process: First, the internal function `_collectRewards` must be called, which updates the accumulated per share rewards for all users and then computes and transfers an individual user's total outstanding rewards. The computation of a user's outstanding rewards uses the mapping `rewardDebt` for internal bookkeeping. Because `rewardDebt` contains a user's debt in absolute terms, it can only be updated as a second step outside of the `_collectRewards` function after a potential change of the user's stake has been accounted for. In effect, user rewards can only be computed correctly if a call to `_collectRewards` is jointly used with an update of `rewardDebt`.

The function `extend` only performs an update on `rewardDebt` without a prior call to `_collectRewards`. Hence, it always discards the rewards earned by a user instead of paying them out.

While calling `_collectRewards` within the `extend` function would mitigate the issue, consider instead solving the root cause by migrating to a mapping `rewardDebtPerShare`. This mapping can be updated within the `_collectRewards` function, which does not need to account for changes in the user's balance, thereby avoiding any future mismatches in reward accounting.

Update: *Fixed by the changes made in pull requests #88 and #98.*

Strongly coupled contracts can break core functionality

The `OgvStaking` contract is strongly coupled with the `RewardsSource` contract:

- In `OgvStaking` the external functions `stake`, `unstake` and `extend` must call the internal function `_collectRewards` to update and transfer a user's rewards.
- `_collectRewards` calls `RewardsSource.collectRewards` to update the `accRewardPerShare` variable and receive all rewards that accrued within `RewardsSource` since the last call to `collectRewards`.

This issue is further amplified by misleading documentation in the `RewardsSource.setRewardsTarget` function, which contains the comment “Okay to be zero, just disables collecting rewards”. However, setting the `rewardTarget` to the zero address would cause any calls from `OgvStaking` to `RewardsSource.collectRewards` to revert, which will disable `staking`, thereby not allowing any new `OGV` holders to participate in governance.

Consider wrapping the external call to `RewardsSource.collectReward` into a try/catch block to achieve decoupling of reward mechanics and staking-based governance. Additionally, consider removing the `noRewards` parameter of the `unstake` function which was originally intended for emergency withdrawals.

Update: *Fixed in pull request #97. In addition, consider catching the error reason and emitting it as an event parameter to allow detection of the otherwise silent error.*

Medium Severity

Staking function can lead to loss of funds

In the `OgvStaking` contract the function `stake` allows anybody to stake `OGV` tokens for a certain duration. Typically, a user interacting with a staking function expects the supplied funds to be attributed to `msg.sender`. However, to deposit a stake on another user’s behalf a parameter `address to` is supplied to indicate the receiver of the stake. Further, the receiver is set to `msg.sender` if the zero address is provided.

If a user wants to simply stake for themselves, this leaves them with the option to call either `stake(amount, duration, msg.sender)` or `stake(amount, duration, 0x0)`, both of which are surprising. Moreover, if the user wants to stake on behalf of another address, there are no checks to ensure the supplied account (which may be a contract) is able to call the `unstake` function or interact with `OGV` tokens. This behavior may lead to the loss of staked funds.

the `onlyGovernor` modifier to the latter function to limit staking on behalf of another user to official airdrops. Further, consider verifying, either within the contract or off-chain, that a smart contract receiver can operate on the received stake to prevent an unintended loss of funds.

Update: *Partially fixed in pull request #89. It is now simpler and more intuitive for a user to stake for themselves. However, staked funds may still be lost if a smart contract receiver is not designed to operate on the staked funds.*

Lack of event emission

The following functions do not emit relevant events after executing sensitive actions:

- The `setRewardsTarget` function changes the address `OGV` tokens are minted to as part of the staking reward system.
- The `setInflation` function deletes and optionally updates the rewards slopes.

Consider emitting events after sensitive changes take place to facilitate tracking and notify off-chain clients following the contracts' activity.

Update: *Fixed in pull request #96. In addition, consider indexing the event parameters.*

Incomplete test suite

The testing suite covering in-scope contracts is incomplete.
Although `OgvStaking.t.sol` and `Rewards.t.sol` test files were provided, the instructions in the `README` for the project do not sufficiently provide guidance on how to run comprehensive tests for the repo.

As the test suite was left outside the audit's scope, please consider thoroughly reviewing the test suite to make sure all tests run successfully after following the instructions in the `README` file. Extensive unit tests aiming at 95% coverage are recommended in order for the security of the project to be assessed in a future audit. Integrating test coverage reports in every single pull request of the project is also highly advisable.

Update: *Fixed in pull request #100.*

Throughout the codebase, we found several instances where documentation was lacking. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned, and the events emitted. For instance:

- The functions and variables in the `OgvStaking` contract lack documentation.
- The functions and some variables in the `RewardsSource` contract lack documentation.
- The `setInflation` function should explicitly state:
  - The start time of the first slope may lie in the future, which allows an implicit configuration of a zero slope before the first start time.
  - The end time of the end slope will always be set to infinity (`type(uint64).max`), which implies that unless `Slope.ratePerDay` is set to zero in the last slope a potentially unbounded number of OGV tokens could be minted.
- In the `OgvStaking` contract, when a user stakes `OGV` tokens before the `epoch` time is reached, the tokens are locked for the specified duration after epoch. Hence, the tokens are locked for longer than expected. Users might be unaware of this behavior, so consider documenting it explicitly.

Consider thoroughly documenting the aforementioned code using the NatSpec format to increase the understandability of the codebase.

Update: *Fixed in pull request #102.*

Voting token name and symbol are mixed up

In the `OgvStaking` contract the return values of the `name` and `symbol` functions are mixed up. Also, following this blog post from March 29, 2022, it is stated that the token symbol will be `veOGV` instead of `OGVe` .

Consider swapping the return values to correctly reflect the token parameters as well as renaming the symbol to match the announcement.

Notes & Additional Information

Gas savings

Throughout the codebase we found some instances that could be improved to save gas:

- In the `setInflation` function of the `RewardsSource` contract, gas can be saved with the following changes:
  - Consider writing `slopes.length` to the stack with `uint256 slopesLength = slopes.length;` and using that variable instead of reading from memory each time.
  - Additionally, the last overwrite of `minSlopeStart` for `index i = slopes.length - 1` is not needed. Consider moving it within the condition two lines above for a minor gas optimization.
- To calculate staking rewards the internal function `_calcRewards` iterates through an array of up to 48 slopes to compute the results that have been accumulated since it was last called. While the computation appears to be correct, many conditions lead to an unnecessary iteration of the entire array:
  - The gas optimization to return zero for the `nextSlopeIndex` has not been applied consistently, it is missing for the condition `last >= block.timestamp`.
  - The condition `rangeEnd < slopeStart` leads the loop to `continue` with the next iteration, while the correct behavior would be to `break`, because no future slope can match a `rangeEnd` in the past.
  - The condition `slopeEnd < rangeEnd` will never be true and is not the condition that corresponds to the comment “No future slope could match”. Consider replacing it with the condition `rangeEnd < slopeEnd` which holds when no future slope can match.
  - To skip slope iterations when range limits match the slope limits, the conditions need to include the equal case. Hence, change to `rangeStart >= slopeEnd`, `rangeEnd <= slopeStart`, and `slopeEnd >= rangeEnd`.

Consider applying the above changes to make the code more gas efficient while maintaining its readability.

Undocumented magic numbers

In the `Governance` contract the constructor uses several magic numbers to define key characteristics of the deployed system. This includes proposal delay, voting period, and voting threshold as well as the time extension in the case of a late quorum. Moreover, the magic numbers lack documentation and implicitly assume a fixed block time of 15 seconds.

Consider documenting these numbers more explicitly by describing their purpose and provide contextual information regarding time spans and average block times, such that it is easier for anyone to understand the characteristics.

Unused imports

In `RewardsSource` the following imports are not used:

- `ERC20Votes`
- `ERC20Permit`
- `PRBMathUD60x18`

Consider removing the imports.

Additionally, the import `ERC20` is used as the type of `ogv`. However, no ERC20-functions are called on it. Consider declaring `ogv` as type `Mintable` as only the `mint` function is called on it and remove the `ERC20` import.

***Update**: Fixed in pull request #94.*

Missing license identifier

In the `RewardsSource.sol` and `OgvStaking.sol` files, a SPDX-license-identifier comment is missing.

Consider adding the MIT license identifier in accordance with the rest of the codebase.

***Update**: Fixed in pull request #95.*

better documentation and testing is added to the codebase, along with addressing the reported findings, the overall quality of the project will be in a good spot. It was great to work together with Origin on this, as they were very open and responsive to discussions. We reported two high severity findings along with additional lower severity recommendations that address best practices and reduce the potential attack surface.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin