# ScopeLift Flexible Voting Audit

OPENZEPPELIN SECURITY | JUNE 8, 2023

Security Audits

June 8, 2023

This security assessment was prepared by **OpenZeppelin**.

## Table of Contents

## Summary

Type
Governance
Timeline
From 2023-03-07
To 2023-03-09
Languages
Solidity

Total Issues
6 (6 resolved)
Critical Severity Issues
0 (0 resolved)
High Severity Issues
0 (0 resolved)
Medium Severity Issues
0 (0 resolved)
Low Severity Issues
3 (3 resolved)
Notes & Additional Information
3 (3 resolved)

## Scope

We audited the `ScopeLift/flexible-voting` repository at the ef95039 commit.

In scope was the following contract:

```
flexible-voting/src/
└── GovernorCountingFractional.sol
```

## System Overview

The Flexible Voting system developed by ScopeLift allows delegates to split their voting weight into fractions between `Against`, `For` and `Abstain` votes. Without this system, a delegate can only cast their entire voting weight into one of the three options. This splitting of voting weight is particularly useful for delegates (contracts) that store many different users' funds, such as lending

Therefore, this system creates a mechanism such that one delegate can split their voting weight proportionally according to the votes of their users. The `GovernorCountingFractional` contract is an extension to the OpenZeppelin contract-library's `Governor` contract and replaces the `GovernorCountingSimple` contract with a few notable differences.

The primary difference, as the use case suggests, is in the counting of votes. The Flexible Voting system overrides the traditional `_countVote` function by utilizing the `bytes memory` parameter in the function signature. A non-empty value for this parameter represents the weights that the delegate assigns to the `Against`, `For`, and `Abstain` votes. If the value of this parameter is empty, the total voting weight of the delegate is counted towards only one of the three voting options.

The second difference is in how the contract accounts for the votes cast by an address for any given proposal. The `GovernorCountingFractional` contract removes the `hasVoted` mapping from the `ProposalVote` struct that OpenZeppelin's `GovernorCountingSimple` contract uses to check if an address has voted on a proposal. Instead, this contract stores the number of votes per account per proposal in the `_proposalVotersWeightCast` mapping. This allows for the additional functionality of rolling voting, in which one account can vote for a proposal with a portion of its total weight, and then again subsequently with some remainder of its weight. However, once a portion of the weight is cast, that portion cannot be changed in future voting.

In addition to the code review, the integration of the `GovernorCountingFractional` contract with OpenZeppelin's suite of governance contracts was examined and found to be compatible.

## Security Model & Trust Assumptions

The scope of this audit was limited to the `GovernorCountingFractional` contract and was done under the trust assumption that the governance tokens or ERC-20 tokens used for voting are vetted by the projects integrating with this contract.

# Client-Reported Findings

## Signature Can Be Replayed for Fractional Voting

After the audit, the client reported an issue in the `GovernorCountingFractional` contract where signature-based votes can be replayed if the delegate's voting weight was not exhausted, the cast vote utilized less than or equal to half of the voting weight, and the vote was cast using the `castVoteWithReasonAndParamsBySig` function. This replaying of the signature would result in consuming more of the user's voting weight, while keeping the proportion of the `Against`, `For`, and `Abstain` votes the same.

Consider a scenario where a user with 100 voting weight signs a message to cast four votes, `1 Against, 2 For, 1 Abstain`, and calls the `castVoteWithReasonAndParamsBySig` function. An attacker sees this transaction on-chain and keeps replaying the signature to consume the remainder of the user's voting weight. Therefore, the final vote count would be `25 Against, 50 For, 25 Abstain`. Since the `castVoteWithReasonAndParamsBySig` function does not track the signature's nonce, the votes may be cast against the user's intentions and their voting weight is exhausted.

***Update:*** *The ScopeLift team resolved the issue in pull request #51 at commit e5de2ef.*

# Low Severity

## Inconsistent Ordering of Votes

In `GovernorCountingFractional.sol`, the `ProposalVote` struct is defined in the following order: `againstVotes`, `forVotes`, `abstainVotes`. This order aligns with the standard usage in OpenZeppelin's contracts library as well as in Compound's Bravo.

However, the `voteData` bytes parameter received by the `_countVote` function requires a different order: `forVotes`, `againstVotes`, `abstainVotes`. While the code currently handles the decoding of this bytes parameter correctly, this inconsistency in the ordering can be

In the interest of improving clarity and being less error-prone, consider changing the order of the `voteData` to require votes in the originally-defined order ( `againstVotes` , `forVotes` , `abstainVotes` ).

***Update**: Resolved in pull request #40 at commit 8bea587.*

## Change Quorum to Account for `Abstain` Votes

The `COUNTING_MODE` function in the `GovernorCountingFractional` contract sets the quorum to `bravo` , meaning that only the `For` votes are counted to reach the quorum.

It is noted that the projects integrating with this contract can `override` the `COUNTING_MODE` and `_quorumReached` functions to choose their preferred quorum setting. However, since the `GovernorCountingFractional` contract is to be used as a library, it is recommended to allow both `For` and `Abstain` votes to be included in reaching a quorum. This would also help in improving the compatibility and consistency with OpenZeppelin's governance modules.

Consider changing the quorum to `for,abstain` to be consistent with OpenZeppelin's `GovernorCountingSimple` contract.

***Update**: Resolved in pull request #41 at commit 450b444.*

## Incomplete Docstrings

Although docstrings are present above the `_countVoteNominal` and `_countVoteFractional` functions, they do not provide complete information about the purpose of these functions and the parameters passed to them.

Since the `GovernorCountingFractional` contract supports rolling votes, once an address has cast a portion of its votes using the `_countVoteFractional` function, the subsequent votes should be cast by calling the `_countVoteFractional` function again, even if the votes are in favor of only one option.

Consider adding proper docstrings to the functions, following the NatSpec format, stating the scenarios in which these functions should be called, and mentioning their limitations.

## Notes & Additional Information

### OpenZeppelin Imports in Incorrect Format

In `GovernorCountingFractional.sol`, the current format for importing OpenZeppelin files is `import {SomeContract} from "openzeppelin-contracts/..."`. With the current import statement, developers who are using this contract (particularly those who are not using Foundry as their smart contract development toolchain) may run into compilation errors when trying to integrate. For ease of consumption, consider changing the imports' format to `import {SomeContract} from "@openzeppelin/contracts/..."` as described in the OpenZeppelin documentation.

***Update**: Resolved in pull request #45 at commit 540cf75.*

### Naming Suggestions

To enhance explicitness and readability, there are several areas in the contract that could benefit from more precise and descriptive naming:

- The `proposalvote` variables on line 77 and line 86 are inconsistent with `proposalVote`, which has camel case and is prevalent both in OpenZeppelin's contracts as well as elsewhere in this contract. Consider renaming it to `proposalVote` for consistency.
- The `weight` variables on line 107, line 130, and line 159 are not explicit about which `weight` they are referring to. Consider renaming them to `totalWeight` for clarity.
- The `safeWeight` variable is not explicit about which `safeWeight` it is referring to. For more clarity, consider renaming it to `safeTotalWeight`.

***Update**: Resolved in pull request #43 at commit 17b820f.*

### Remove Unnecessary Comments

To improve readability, consider removing the following comments related to the Foundry forge format:

- Lines 6 and 7 of the `GovernorCountingFractional` contract

## Conclusions

This audit was conducted over the course of 3 days. Through an in-depth review of the `GovernorGovernorCountingFractional.sol` file, we uncovered three low-severity issues. The ScopeLift team provided clear documentation, which helped the auditors understand this system.

The pull requests containing the fixes were merged to the `master` branch of the `flexible-voting` repository at commit `4399694`.

## Appendix

### Monitoring Recommendations

While the ScopeLift team's Flexing Voting system is meant to be used by other projects, it is important for projects that would like to integrate with this system to monitor certain activity. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

Hence, with the goal of providing a complete security assessment, we encourage projects to monitor the events for when proposals are created, voted on, executed, and cancelled. In this case, they should monitor the `ProposalCreated`, `VoteCast`, `VoteCastWithParams`, `ProposalExecuted`, and `ProposalCanceled` events.


## Related Posts

Beefy

BRUSHFAM

OpenBrush Contracts

Linea

## 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