Lido Voting Security Review

Prepared by: Vaibhav Sutar

Table of Contents

Disclaimer

The **Vaibhav Sutar** audit team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Severity	Description
High	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Low	Low impact and low/medium likelihood events where assets are not at risk.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Protocol Summary

The Lido DAO is operating via on-chain votings. Each voting has several options; the option with the most tokens voted for determines the outcome. Moreover, for voting to be considered valid, it is necessary that a certain quorum be reached. The presence of a quorum is an important marker indicating the current health of a DAO.

However, in the current conditions, reaching a quorum on Lido DAO voting has become highly challenging, and several consecutive votes are still needed to reach a quorum.

The Simple Delegation is a straightforward solution to integrating on-chain delegation for Lido DAO voting. The key objective of the project is to enable simple and secure delegation to overcome the current challenges in achieving quorum for on-chain voting.

Audit Details

The findings described in this document correspond the following commit hash:

```
997dd7e03f85b19a411b2ff721edc018563a1857
```

Scope

```
├── apps
│   ├── voting
│       ├── Contracts
│           └── Voting.sol
```

Executive Summary

Issues found

Severity	Findings
Critical	0
High	0
Medium	0
Informational	6
Total	**6**

Tools Used

- Slither
- Aderyn
- cloc
- Solidity Metrics (audit estimation)
- Solidity Visual Developer

Findings

Informational

[I-01]: Code style recommendations.

Description:

1. Some Errors and their content do not match

- Voting.sol#L38
- Voting.sol#L39

```
string private constant ERROR_CHANGE_VOTE_TIME = "VOTING_VOTE_TIME_TOO_SMALL";
string private constant ERROR_CHANGE_OBJECTION_TIME = "VOTING_OBJ_TIME_TOO_BIG";
```

2. Several functions are missing required NatSpec parameters.

- Voting._newVote()
- Voting._vote()
- Voting._unsafeExecuteVote()
- Voting._getDelegatedVotersAt()
- Voting._canExecute()
- Voting._isValidPhaseToVote()
- Voting._getVotePhase()
- Voting._isVoteOpen()
- Voting._isValuePct()

Recommendation

- We recommend implementing the suggested changes.

[I-02]: Extra checks in Voting._isVoteOpen()

Description

The function Voting._isVoteOpen() checks two things:

1. If the voting time is over.

2. If the vote has already been executed.

```
function _isVoteOpen(Vote storage vote_) internal view returns (bool) {
@>        return getTimestamp64() < vote_.startDate.add(voteTime) &&
!vote_.executed;
    }
```

We believe the second check is not needed. This is because if the time is over then vote will not be considered. and function _canExecute() checks the vote has been executed or not

```
function _canExecute(uint256 _voteId) internal view returns (bool) {
        Vote storage vote_ = votes[_voteId];

@>        if (vote_.executed) {
            return false;
        }

        // Vote ended?
        if (_isVoteOpen(vote_)) {
            return false;
        }

        // Has enough support?
        uint256 voteYea = vote_.yea;
        uint256 totalVotes = voteYea.add(vote_.nay);
        if (!_isValuePct(voteYea, totalVotes, vote_.supportRequiredPct)) {
            return false;
        }
        // Has min quorum?
        if (!_isValuePct(voteYea, vote_.votingPower, vote_.minAcceptQuorumPct)) {
            return false;
        }

        return true;
    }
```

Recommendation

- We recommend to remove the vote_.executed flag from Voting._isVoteOpen()

[I-03]: Incorrect Voting Power Check Blocks Legitimate Delegation

- Line: Voting.sol#L255

Description:

- In the `Voting.setDelegate()` function, the system incorrectly checks if a user has voting power by looking at their token balance right now. This is a mistake because voting power for a specific proposal is based on a user's balance at a past snapshot block.
- This means a user could have the right to vote on an active proposal based on the snapshot but be blocked from appointing a delegate because their current balance is zero.

```
    function setDelegate(address _delegate) external {
           require(_delegate != address(0), ERROR_ZERO_ADDRESS_PASSED);
           require(_delegate != msg.sender, ERROR_SELF_DELEGATE); // <---
```

Recommendation

- Remove the zero check

```
(require(votingPower > 0, ERROR_NO_VOTING_POWER);)
```

The system already correctly validates snapshot-based voting power in Voting.vote() and Voting.attemptVoteForMultiple() via the _hasVotingPower() function, making this check redundant and harmful.

[I-04] : Duplicate Voters in Batch Voting Create Redundant Events

Description: The attemptVoteForMultiple function allows the same voter address to be listed more than once. While the contract correctly prevents a single voter from having their vote counted twice, it will still emit multiple CastVote events for that voter. This can clutter the event log and complicate off-chain systems that track voting activity.

Recommendation: Handle this edge case in off-chain applications by filtering out duplicate CastVote events for the same voter and vote ID.

[I-05] : Lack of minimum and maximum Voting and Objection time

Description:

- Lines: Voting.sol#L166 Voting.sol#L180

The contract lacks minimum and maximum limits for voting and objection durations. If the UNSAFELY_MODIFY_VOTE_TIME_ROLE is compromised, an attacker could:

- Set voting time excessively long (e.g., 100 years) causing denial-of-service
- Set objection time to zero, preventing community opposition to malicious proposals

Recommendation:

- Implement minimum and maximum thresholds for both voting and objection times to limit potential damage from role compromise.

[I-06]:Inconsistent Documentation and Logic in canVote() Function

Description: Line: Voting.sol#L400 The canVote() function has conflicting documentation and incomplete phase handling. The function comment states it covers both main and objection phases, but the return description mentions only the main phase. Additionally, the function doesn't properly handle objection phase constraints where users can only vote against proposals.

Impact:

- Misleading documentation creates confusion for developers and users
- Function may return incorrect results during objection phases
- Potential integration errors with external systems

Recommendation:

1. Fix documentation inconsistencies to accurately reflect function behavior
2. Add a `_supports` parameter to properly handle objection phase logic:

```
function canVote(uint256 _voteId, address _voter, bool _supports) external view
voteExists(_voteId) returns (bool) {
    Vote storage vote_ = votes[_voteId];
    VotePhase votePhase = _getVotePhase(vote_);
    uint256 votingPower = token.balanceOfAt(_voter, vote_.snapshotBlock);
    return _isValidPhaseToVote(votePhase, _supports) && votingPower > 0;
}
```