STATE MIND

Voting

04-03-2024 - 12-03-2024

Table of contents

1. Project brief

Title	Description
Client	Lido
Project name	Voting
Timeline	04-03-2024 - 12-03-2024
Initial commit	997dd7e03f85b19a411b2ff721edc018563a1857
Final commit	50d9802f6e728388b80b275b7baea5d93b9b6b25

Short Overview

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.

Project Scope

The audit covered the following files:

Voting.sol

2. Finding severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Client regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Client is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	0 (0 fixed, 0 acknowledged)
Informational	6 (2 fixed, 4 acknowledged)
Total	6 (2 fixed, 4 acknowledged)

4. Conclusion

During the audit of the codebase, 6 issues were found in total:

- 6 informational severity issues (2 fixed, 4 acknowledged)

The final reviewed commit is 50d9802f6e728388b80b275b7baea5d93b9b6b25

Deployment

Contract	Address
Voting (Proxy)	0x2e59A20f205bB85a89C53f1936454680651E618e
Voting (Implementation)	0xf165148978Fa3cE74d76043f833463c340CFB704

5. Findings report

INFORMATIONAL-01	Excessive check in Voting._isVoteOpen()	Acknowledged

Description

At function **Voting._isVoteOpen()** there are two conditions:

1. if time is over
2. if the vote is executed

We think that the second check is excessive because the vote will be closed anyway if time is over. There is no opportunity to execute the vote until the time is up, also check if **vote.executed** is true already in **_canExecute** function.

Recommendation

We recommend removing the check for **executed** flag in **Voting._isVoteOpen()** function.

Client's comments

Upgrade doesn't affect this code since last audit so we prefer to leave it as is.

INFORMATIONAL-02	Code style recommendations	Fixed at: 08d43e3

Description

Some error names don't match their contents.

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

Missing some NatSpec parameters in several functions.

Voting._newVote(), **Voting._vote()**, **Voting._executeVote()**, **Voting._unsafeExecuteVote()**, **Voting._getDelegatedVotersAt()**, **Voting._canExecute()**, **Voting._isValidPhaseToVote()**, **Voting._getVotePhase()**, **Voting._isVoteOpen()**, **Voting._isValuePct()**.

Recommendation

We recommend applying our recommendations.

INFORMATIONAL-03	Inconsistent delegation check	Fixed at: 22fe318

Description

In the **Voting.setDelegate()** function, there's a requirement that checks a user's voting power is greater than 0 by checking the token balance at the previous block. This method overlooks users' ability to vote based on balances at past snapshot blocks, limiting delegation rights.

```
uint256 votingPower = token.balanceOfAt(msg.sender, getBlockNumber64() - 1);
require(votingPower > 0, ERROR_NO_VOTING_POWER); // <-- Disregards snapshot-based voting power
```

Users may have no current balance yet retain voting power from a past snapshot, a discrepancy that limits delegation.

Recommendation

It is recommended to remove this zero check as there are already in **Voting.vote()**, **Voting.attemptVoteForMultiple()** functions.

INFORMATIONAL-04	Lack of minimum and maximum Voting and Objection time	Acknowledged

Description

Lines:

- Voting.sol#L166
- Voting.sol#L180

There is a concern about the fact that there is no minimum and maximum amount for Voting and Objection time.

Impact:

If **UNSAFELY_MODIFY_VOTE_TIME_ROLE** gets compromised they could harm the voting process, from making the DoS attack to passing harmful scripts. DoS would happen if the voting time was extended to a very long time (e.g., 100 years). The passing of harmful scripts would happen if the compromiser puts 0 as an objection time, thus, the community will not have time to vote **Nay** for the proposal, if the attacker votes for a minimum quorum at the last second of voting time.

Recommendation

We recommend putting both minimum and maximum thresholds for both voting and objection time, such that even if the **UNSAFELY_MODIFY_VOTE_TIME_ROLE** gets compromised, it would have restrictions on its powers and could not execute the attack described in the impact section above.

Client's comments

Acknowledged

INFORMATIONAL-05	Duplicate voters addresses can be passed to **attemptVoteForMultiple**	Acknowledged

Description

Line: Voting.sol#L294.

It is possible to pass the same voter address multiple times and vote on their behalf. This doesn't lead to double vote accounting, but will emit multiple **CastVote** events for the same address which might complicate off-chain logic.

Recommendation

We recommend considering this edge case in off-chain logic.

Client's comments

Acknowledged. The off-chain logic will consider duplicated **CastVote** events.

INFORMATIONAL-06	Inconsistency in **Voting.canVote()** function	Acknowledged

Description

Line: Voting.sol#L400

In the comments section, there is a notice that the **Voting.canVote()** function tells if voters can participate in the main or objection phase. But the comment for the return result says that this function returns if the voter can participate only in the main phase.

```
/**
* @notice Tells whether `_voter`can participate in the main or objection phase of the vote #`_voteId`
...
* @return True if the given voter can participate in the main phase of a certain vote
*        both directly and indirectly by a delegate voting for them, false otherwise
*/
```

Also function **Voting.canVote()** doesn't consider all the conditions when the vote is in the objection phase. During the objection phase, users can only vote against it.

Recommendation

We recommend solving inconsistency in the comments and introducing an additional parameter - **bool _supports** - to cover all the cases of the vote during the main and objection phases.

```
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
}
```

Client's comments

Acknowledged. It was decided not to update the function interface for backward compatibility.

STATE
MIND