Lido

Simple Delegation

by Ackee Blockchain

05.07.2024

Contents

1. Document Revisions

1.0	Final report	28.03.2024
1.1	Re-audit final report	10.04.2024
2.0	Final report	01.07.2024
2.1	Fix review	05.07.2024

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Wake is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don’t jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven’t found a way to exploit the issue given the time constraints, it might be marked as a ”Warning” or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member’s Name	Position
Andrey Babushkin	Lead Auditor
Michal Převrátil	Auditor
Štěpán Šonský	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Revision 1.0

Lido engaged Ackee Blockchain to perform a security review of the Simple Delegation with a total time donation of 10 engineering days in a period between Mar 18 and Mar 28, 2024, with Andrey Babushkin as the lead auditor. The scope includes the implementation of voting for Lido DAO with the simple delegation of votes.

The audit was performed on the commit `08d43e3` [1] and the scope was the following:

- Voting.sol

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved the Wake testing framework.

A complex fully differential fuzz test was prepared to ensure the correctness of the system. The fuzzing was performed with the bytecode generated by the Solidity compiler in version 0.4.24 with the optimizer enabled. The full source code of the fuzz test is available at https://github.com/Ackee-Blockchain/tests-lido-simple-delegation.

During the review, we paid special attention to:

- ensuring the code is not subject to bugs of the outdated compiler version,
- the voting logic is correct and is not affected by the additional delegation logic,
- ensuring the arithmetic of the system is correct and there are no overflows or underflows in the arithmetic operations,

- the upgradeability is implemented and used correctly,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validation.

Our review resulted in 10 warnings. In addition, we have concerns about the very concept of the vote delegation. The delegation of votes is a powerful tool that can be used to increase voter turnout, but it can also be used to centralize power in the hands of a few. Moreover, this mechanism can be potentially misused, see W2. Since Lido has a strong influence on the Ethereum ecosystem, we encourage the team to consider a different approach to address voter apathy that would be more conducive to decentralization.

Ackee Blockchain recommends Lido:

- consider using the latest Solidity version to take advantage of the latest optimizations and bug fixes,
- consider a different approach to address voter apathy that would be more conducive to decentralization,
- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

Ackee Blockchain performed a fix review of findings in Revision 1.0 on commit `e3cef8c`. 3/10 findings were fixed, the remaining 7 findings were acknowledged by the team with explanations (see individual findings for details). The concerns about the vote delegation were taken into account, and the team decided to keep the delegation mechanism as it is with possible improvements in the future. Additionally, for this revision, the Lido team added several new tests and improved the documentation of private functions.

Revision 2.0

Lido engaged Ackee Blockchain to perform an incremental review of the Simple Delegation project with a total time donation of 2 engineering days in a period between Jun 27 and Jul 1, 2024, with Michal Převrátil as the lead auditor.

The audit was performed on the commit `079dd88` [2] with the file `Voting.sol` as the scope.

The review began with migration to Solidity 0.6.2 needed for running Wake static analysis detectors. Static analysis yielded the I1 finding. We then continued with updating the fuzz test prepared in the previous revision and performed an incremental manual review in parallel with the fuzzing.

During the review, we especially checked:

- the refactoring did not introduce new ways to exploit the system,
- the code style and readability remained at a high level.

Our review resulted in 4 informational findings. The vote delegation concept remained unchanged.

Ackee Blockchain recommends Lido:

- address the discovered findings.

See Revision 2.0 for the updated system overview.

Revision 2.1

Lido engaged Ackee Blockchain to perform a fix review of the findings discovered in the previous revision on the commit `50d9802` [3].

3 out of 4 findings were fixed, the I2 finding was acknowledged. Except for the fixed findings, two minor changes were made to the codebase to improve the readability and gas usage. The new changes were reviewed as well. See Revision 2.1 for the details of the updated codebase.

[1] full commit hash: 08d43e3287051db51049b5703d97ddd9b02afaff

[2] full commit hash: 079dd88e1e6e55f223792d1becb3fe68427473e5

[3] full commit hash: 50d9802f6e728388b80b275b7baea5d93b9b6b25

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
W1: Usage of `solc` optimizer	Warning	1.0	Acknowledged
W2: Delegation does not expire	Warning	1.0	Acknowledged
W3: The initializer can be front-run	Warning	1.0	Acknowledged
W4: The initializer does not have validations for the correctness of `_token`	Warning	1.0	Acknowledged
W5: Declaration shadowing	Warning	1.0	Fixed
W6: Unused function parameters	Warning	1.0	Fixed

	Severity	Reported	Status
W7: Outdated Solidity version	Warning	1.0	Acknowledged
W8: Division rounding error	Warning	1.0	Acknowledged
W9: Set delegate with zero voting power	Warning	1.0	Fixed
W10: Inconsistent `attemptVoteForMultiple` checks	Warning	1.0	Acknowledged
I1: Unused function	Info	2.0	Fixed
I2: Reserved keyword	Info	2.0	Acknowledged
I3: Cache array length	Info	2.0	Fixed
I4: Incorrect NatSpec format	Info	2.0	Fixed

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

The audited scope consists of one contract, namely Voting. Voting is one of several apps within the Lido DAO system built atop the Aragon framework. The current scope mainly includes the implementation of the so-called ”vote delegation” feature. The delegation mechanism allows token holders to delegate their voting power to another address without transferring their tokens. The delegated address can then vote on behalf of the delegator. This feature is a straightforward solution to the voter apathy problem, where token holders do not express interest in voting, and it becomes difficult to reach a quorum.

Contracts

Contracts we find important for better understanding are described in the following section.

Voting

The contract implements a voting system where votes are proposals created within Lido DAO. The contract allows the creation of new votes with an executable script payload, and the creation of new votes is restricted to a specific role, `CREATE_VOTES_ROLE`. Every vote has a start and end time and is split into two phases: the main phase and the objection phase. During the main phase, token holders can vote with YEAs or NAYs. The objection phase is a time window where token holders can only vote with NAYs. The start time is set to the current block timestamp, the total duration of the vote is set by the `_voteTime` variable, and the objection phase is set by the `_objectionPhaseTime` variable. These variables are stored in the contract storage and are shared between all votes.

The vote is considered successful if (1) the percentage of YEAs from the total casted voting power is higher than `supportRequiredPct`, and (2) the total voting power is higher than `minAcceptQuorumPct`. The voting power of a voter is the number of governance tokens held by a voter on the block prior to the block when the vote is created. As in the case of the vote time variables, the parameters for the minimum quorum and the percentage of YEAs are shared between all votes and are stored in the contract storage.

Each vote can have a payload that can be executed by calling `executeVote` after the vote is finished and is successful. The payload is an EVM script. Everything above is defined in the Aragon framework.

An additional feature of the contract is the vote delegation mechanism. Token holders can delegate their voting power to another address without transferring their tokens using the `setDelegate` function. The delegate can then vote on behalf of the delegator with the voting power of the delegator. A delegator can delegate the voting power to one address and revoke the delegation at any time by using the `resetDelegate` function. However, if the delegate has already voted on behalf of the delegator, the vote remains unchanged (only if the delegator does not change their vote themselves). The delegate can vote and change the vote at any time during the correct voting phase, nevertheless, they cannot change the vote if it is cast by the delegator themselves, that is, the original vote by the delegator has a higher priority. The delegate can vote on behalf of one delegator by calling `attemptVoteFor` or on behalf of multiple delegators by calling `attemptVoteForMultiple`, and the list of delegators is passed as an argument to the function.

The Voting contract is upgradeable, it extends the AragonApp contract and implements the IForwarder interface. The IForwarder interface is used to forward the calls to the Voting contract an entity with correct permissions. Forwarding here means the creation of a new vote with a script. For example, if one wants to execute some operation and cannot do it directly, they can create a vote with the operation as a payload and execute it in case the vote is successful.

Actors

This part describes the actors of the system, their roles, and permissions.

Initializer

The user who deploys the proxy contract and calls the initialize function of Voting through the proxy. It can be any user without specific permissions. The initializer sets all global parameters, such as the address of the governance token, the support required percentage, the minimum quorum percentage, the voting time, and the objection time.

Support Modifier

It is a user enabled to change the support required percentage. An address is granted with this permission by setting the MODIFY_SUPPORT_ROLE role through the ACL application.

Quorum Modifier

It is a user enabled to change the minimum quorum percentage. An address is granted with this permission by setting the MODIFY_QUORUM_ROLE role through the ACL application.

Vote Time Modifier

It is a user enabled to change the voting time and the objection time. An address is granted with this permission by setting the `UNSAFELY_MODIFY_VOTE_TIME_ROLE` role through the ACL application.

Vote Creator

It is a user enabled to create new votes and forward the calls to the `Voting` contract with a script payload. An address is granted with this permission by setting the `CREATE_VOTES_ROLE` role through the ACL application.

User without Governance Tokens

A user without governance tokens cannot vote or delegate their voting power. However, they can read the state of the contract and execute the payload of successful votes.

User with Governance Tokens (Voter)

If a user has governance tokens in a block prior to the block when the vote is created, they can vote on the vote. The voting power of a voter is the number of governance tokens held by the voter. The user can vote with YEAs or NAYs during the main phase and only with NAYs during the objection phase. The user can delegate their voting power to another address without transferring their tokens. The user can revoke the delegation at any time. Finally, as the user without governance tokens, the user can read the state of the contract and execute the payload of successful votes.

Delegate

A delegate is an address to which the voting power is delegated. The delegate can vote on behalf of the delegator with the voting power of the delegator. The delegate can vote and change the vote at any time during the correct voting phase. The delegate can vote on behalf of one delegator or multiple delegators. The delegate cannot change the vote if it is cast by the delegator themselves. The delegate can read the state of the contract and execute the payload of successful votes.

5.2. Trust Model

Users must trust an entity with the `CREATE_PERMISSION_ROLE` role to grant permissions to other entities, which can later affect the correct operation of the voting protocol. Delegators must trust the delegate to behave honestly and not to vote for proposals that can harm the interests of the delegator or the system, especially because the delegation does not have an expiry date.

The latter is especially important because a small portion of delegators can aggregate a significant amount of voting power and that poses a huge risk of centralization. If a user decides to delegate their voting power, they must take into consideration these risks and choose a delegate who has not accumulated a significant amount of voting power yet. These huge delegates do not have to be malicious, but they can be a single point of failure in the system. For example, the trustworthy Lido team may persuade the users to delegate their voting power to them, and the team may vote for proposals that are beneficial for the team but not for the users. This is a risk that the users must take into consideration when delegating their voting power.

W1: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

Exploit scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> Considering that the Solidity team states that all semantic tests are running with the optimizer enabled and disabled, we concluded that turning off the optimizer would not benefit enough.

Go back to Findings Summary

W2: Delegation does not expire

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Trust model

Description

The simple delegation mechanism does not implement the expiration of the right to vote on behalf of a delegator. This may open a risk of an adversary maliciously becoming a delegator and collecting the voting power of token holders (for example, by use of fishing techniques) and then voting on any malicious proposal that favors this bad actor. Because the delegation does not expire, any future proposal will be affected until the user revokes the permission, however, since delegating voting rights does not cause any financial loss, the probability that the user notices the malicious delegation is extremely low.

Recommendation

Implement the mechanism of delegation duration. When the delegation right expires, the delegate cannot vote on behalf of the delegator anymore.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> Although the question of delegation expiration is an important subject to discuss, we made a decision not to introduce such mechanism in the Simple Delegation update. However, we will take this point into account during the development of a full-scale delegation solution.

Go back to Findings Summary

W3: The initializer can be front-run

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Access control

Description

There is a possibility that an attacker can front-run the call of the initializer after the deployment of the proxy contract. This will initialize the contract with incorrect parameters, and the most dangerous is setting the malicious token address for the governance token. If the deployment of the voting contract is held automatically, make sure the deployment script handles the reverting initialization correctly.

Recommendation

Either implement access controls to the initializer function or make sure the deployment and upgrading scripts handle reverting initialization correctly and fail.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> The contract has already been initialized; thus, this case doesn't look possible. If there is a new deployment, the contract initialization will be protected from front-running at the deploy script level, which guarantees that the proxy deployment and initialization happen within a single transaction.

Go back to Findings Summary

W4: The initializer does not have validations for the correctness of `_token`

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Input validation

Description

The `_token` provided to the initializer is not checked for correctness. Type-casting to an interface or a class from `address` does not imply any additional implicit Solidity validations; one can easily pass the zero address or the address of a contract that does not follow the `MiniMeToken` interface.

Recommendation

Consider adding basic input validations for the `_token` argument like the zero-address check.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> The contract has already been initialized; thus, this case doesn't look possible. If there is a new deployment, we will do additional address verifications and re-deploy the contract if an invalid address is passed.

Go back to Findings Summary

W5: Declaration shadowing

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Code quality

Description

The input argument `_vote` in functions `_isValidPhaseToVote`, `_canVoteFor`, `_hasVotedDirectly`, `_hasVotingPower`, `_getVotePhase` and `_isVoteOpen` shadows the internal function `_vote`.

Recommendation

Consider renaming the input parameter to get rid of compiler warnings.

Solution (Revision 1.1)

The issue is fixed by renaming all the occurrences of the *vote* `input argument in all functions to vote`.

Go back to Findings Summary

W6: Unused function parameters

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Code quality

Description

Functions `newVote` and `vote` are declared in two variants, one of which includes arguments with the `_deprecated` suffix. These arguments generate compiler warnings, which may distract developers from other potentially more severe warnings:

Listing 1. Excerpt from Voting

```
    function newVote(bytes _executionScript, string _metadata, bool
_castVote_deprecated, bool _executesIfDecided_deprecated)
```

Listing 2. Excerpt from Voting

```
    function vote(uint256 _voteId, bool _supports, bool
_executesIfDecided_deprecated) external voteExists(_voteId) {
```

Leaving these function declarations as-is is not a part of best practices.

Recommendation

Consider commenting out the unused arguments and specifying the deprecation status in the NatSpec documentation:

```
function newVote(bytes _executionScript, string _metadata, bool
/*_castVote_deprecated*/ , bool /*_executesIfDecided_deprecated*/) ...;
function vote(uint256 _voteId, bool _supports, bool
/*_executesIfDecided_deprecated*/) ...;
```

Solution (Revision 1.1)

The unused function parameters were commented out and the documentation was updated to reflect the deprecation status.

Go back to Findings Summary

W7: Outdated Solidity version

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Outdated Solidity

Description

The project is written in Solidity 0.4.24. While this makes good sense in that the code was proven safe over time, it cannot use security and gas optimization features implemented in the later versions of Solidity. These include:

- `staticcall` when calling `view` and `pure` functions,
- immutable variables,
- `push0` opcode,
- user-defined errors,
- `assert` call not using all available gas,
- array out-of-bounds access checks, and more.

Recommendation

Consider using a newer version of Solidity.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> Updating the Solidity version will require upgrading at least `AragonApp`, which is also compiled by v0.4.24. We consider such an update to be outside the scope of the Simple Delegation, so

> we have decided not to introduce it.

Go back to Findings Summary

W8: Division rounding error

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Arithmetic

Description

The function `_isValuePct` is used to compare the currently achieved percentage of positive votes in the total number of cast or possible votes to a given threshold.

Listing 3. Excerpt from Voting

```
    function _isValuePct(uint256 _value, uint256 _total, uint256 _pct) internal pure returns (bool) {
        if (_total == 0) {
            return false;
        }

        uint256 computedPct = _value.mul(PCT_BASE) / _total;
        return computedPct > _pct;
    }
```

Division is used to compute the percentage of positive votes. However, the division may be subject to rounding errors. The expected rounding error is around 10 ** -16 percent.

Recommendation

Replace the current implementation of the `_isValuePct` function with the following:

```
function _isValuePct(uint256 _value, uint256 _total, uint256 _pct) internal pure
returns (bool) {
    if (_total == 0) {
        return false;
    }
    return _value.mul(PCT_BASE) > _total.mul(_pct);
}
```

Both sides of the comparison are expected to be up to 10 ** 27 (LDO total supply) * 10 ** 18 (PCT_BASE) = 10 ** 45, which is within the range of `uint256`.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> The proposed change is outside the scope of Simple Delegation. It is a part of the core of the contract, and we consider such a change not worth the increase in gaining insignificant accuracy.

[Go back to Findings Summary]

W9: Set delegate with zero voting power

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Logic error

Description

The `setDelegate` function uses the following restrictions on the voter:

Listing 4. Excerpt from [Voting]()

```
        uint256 votingPower = token.balanceOfAt(msg.sender,
getBlockNumber64() - 1);
        require(votingPower > 0, ERROR_NO_VOTING_POWER);
```

However, every vote uses the voting power of the voter on the block before the block where the vote was created. This means that the validation mentioned above does not help filter out voters without voting power — the voter can get the governance token, set the delegate in the next block, and get rid of the token again. At the same time, it disallows voters who previously owned the governance token to set their delegate for the voting they could still influence.

Recommendation

Remove the restriction on the voting power of the voter in the `setDelegate` function.

Solution (Revision 1.1)

The aforementioned lines of code were removed from the `setDelegate` function.

Go back to Findings Summary

W10: Inconsistent attemptVoteForMultiple checks

Impact:	Warning	Likelihood:	N/A
Target:	Voting	Type:	Logic error

Description

In attemptVoteForMultiple, the loop iterates over all input voters. The whole function call will fail if one voter does not have the voting power for the required snapshot block. At the same time, a voter is silently ignored if the delegate cannot vote for the voter.

Listing 5. Excerpt from Voting

```
    function attemptVoteForMultiple(uint256 _voteId, bool _supports, address[] _voters) public voteExists(_voteId) {
        Vote storage vote_ = votes[_voteId];
        require(_isValidPhaseToVote(vote_, _supports), ERROR_CAN_NOT_VOTE);
        bool hasManagedToVote = false;

        address voter;
        for (uint256 i = 0; i < _voters.length; ++i) {
            voter = _voters[i];
            require(_hasVotingPower(vote_, voter), ERROR_NO_VOTING_POWER);
            if (_canVoteFor(vote_, msg.sender, voter)) {
                _vote(_voteId, _supports, voter, true);
                hasManagedToVote = true;
            }
        }

        require(hasManagedToVote, ERROR_CAN_NOT_VOTE_FOR);
    }
```

The inconsistency in how the conditions are checked is confusing and may lead to user experience issues.

Recommendation

Either skip the voters with zero voting power or revert on the first encounter of the condition when the delegate cannot vote for a voter.

Solution (Revision 1.1)

The issue was acknowledged with a comment:

> Such a design decision addressed two issues: code consistency and delegates' user experience. First of all, we wanted `attemptVoteForMultiple` semantics to be as close to the `vote` as possible, and that is why there is a strict requirement for all the voters from the list to have a non-zero token balance. Secondly, we wanted to eliminate the possibility of malicious behavior, a potential front-run attack from a delegate's voter that could harm the delegate's user experience, so we didn't use reverts in situations where a token holder votes directly before a delegate or resets a delegate right before it votes.

Go back to Findings Summary

6. Report revision 2.0

6.1. System Overview

The new revision introduces the following renames and other breaking changes:

- event `SetDelegate` renamed to `AssignedDelegate`,
- event `ResetDelegate` renamed to `UnassignedDelegate`,
- event `CastVoteAsDelegate` renamed to `AttemptCastVoteAsDelegate` with its signature changed,
- function `setDelegate` renamed to `assignDelegate`,
- function `resetDelegate` renamed to `unassignDelegate`,
- function `getVotersStateAtVote` renamed to `getVoterStateMultipleAtVote`
- function `getDelegatedVoters` no longer returns the voting power of the voters,
- function `getDelegatedVotersAtVote` was removed,
- new functions `getVotingPowerMultiple` and `getVotingPowerMultipleAtVote` were introduced.

Other changes are internal to the implementation and represent mainly documentation improvements and code refactoring focusing on better gas optimization.

I1: Unused function

Impact:	Info	Likelihood:	N/A
Target:	Voting	Type:	Code quality

Description

The function `_isVoteOpen` is no longer used in the codebase.

Listing 6. Excerpt from Voting

```
    function _isVoteOpen(Vote storage vote_) internal view returns (bool) {
        // The `!vote_.executed` check must stay in place.
        // In a particular case `unsafelyChangeVoteTime` call might
        // break the vote state and make it possible to vote for the executed vote.
        return getTimestamp64() < vote_.startDate.add(voteTime) && !vote_.executed;
    }
```

Recommendation

Remove the function to improve readability and maintainability.

Fix 2.1

The function was removed from the codebase.

Go back to Findings Summary

I2: Reserved keyword

Impact:	Info	Likelihood:	N/A
Target:	Voting	Type:	Code quality

Description

`supports` is a reserved keyword introduced in later versions of the Solidity compiler, i.e., later versions of solc forbid using `supports` as the name of a variable.

Listing 7. Excerpt from Voting

```
    event CastVote(uint256 indexed voteId, address indexed voter, bool supports, uint256 stake);
```

Recommendation

Consider changing the name of the `supports` variable from the code snippet.

Solution 2.1

The client acknowledged the finding — the variable name will be changed when the contract upgrades to a newer version of Solidity.

Go back to Findings Summary

I3: Cache array length

Impact:	Info	Likelihood:	N/A
Target:	Voting	Type:	Gas optimizations

Description

`_voters.length` in the following code snippet can be cached before the the for loop to improve the gas usage:

Listing 8. Excerpt from [Voting.attemptVoteForMultiple]()

```
        bool votedForAtLeastOne = false;
        uint64 voteSnapshotBlock = vote_.snapshotBlock;
        for (uint256 i = 0; i < _voters.length; ++i) {
            address voter = _voters[i];
```

Recommendation

Read the array length before the for loop and use the cached value inside the for condition.

Fix 2.1

The array length is now cached before the for loop.

[Go back to Findings Summary]()

I4: Incorrect NatSpec format

Impact:	Info	Likelihood:	N/A
Target:	Voting	Type:	Code quality

Description

The correct NatSpec format for named return parameters is:

```
@return parameterName Description of the parameter
```

The functions `getVote`, `getDelegatedVoters`, `getVoterStateMultipleAtVote`, `getVotingPowerMultipleAtVote` and `getVotingPowerMultiple` do not follow this format, omitting the `parameterName` part.

Recommendation

Fix the NatSpec comments in the functions mentioned above.

Fix 2.1

All of the incorrect NatSpec documentation occurrences were updated, adding the names of the return parameters.

Go back to Findings Summary

7. Report revision 2.1

7.1. System Overview

Except for the fixes of the findings from the previous revision, two minor changes were introduced.

An unreachable condition was removed from the `getDelegatedVoters` function, replacing the line

```
if (_offset >= votersCount || votersCount == 0) {
```

with

```
if (_offset >= votersCount) {
```

The second change represents a different handling of the last possible voting phase in the function `_getVotePhase`, replacing

```
if (timestamp < voteTimeEnd) {
    return VotePhase.Objection;
}
assert(false); // Should never reach this point
```

with

```
assert(timestamp < voteTimeEnd);
return VotePhase.Objection;
```

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Lido: Simple Delegation, 05.07.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Thank You

Ackee Blockchain a.s.

📍 Prague, Czech Republic

✉ hello@ackeeblockchain.com

🐦 https://twitter.com/AckeeBlockchain