LIDO TWO-PHASE VOTING SECURITY AUDIT REPORT

June 20, 2022

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Customer. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the cients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reoprts into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Customer with an interim report.

5. Bug fixing & re-audit:

- The Customer either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Customer with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Customer with a re-audited report.

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 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 funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer 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 Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The auditted smart contracts have the logic of voting, based on Aragon standard voting contracts with an additonal objection phase to mitigate the "last-moment" voting problem. After accepting all the "yeas" and "nays" votes, the voting goes to the "objection" phase, when only "no" votes are accepted. The voting power is determined by the voter's balance of a special token (MiniMeToken), allowing to snapshot the voter's balance at the given block, mitigating the risk of the balance's reuse. The voting result is an ability to run the execution script that must be reviewed by voters, its security is not the object of the audit.

Filename	Description
`Voting.sol`	Contract, allowing a two-phase voting (the voting and objection phases) with MiniMeToken.
`MiniMeToken.sol`	Aragon ERC20 token, allowing to snapshot token balance at the given block number, used for voting.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Lido
Project Name	Two-Phase Voting
Timeline	2022-05-25 - 2022-06-01
Number of Auditors	5

Project Log

Date	Commit Hash	Note
2022-05-25	7e5cd1961697a1bc514bfebdeab08a296e51d700	Initial audit
2022-06-03	32d56a7490702d303be1e09b4e24b767e83751c3	Reaudit based on fixes provided
2022-06-07	4f7646fbc90ac31fada5ed9e95669fc4aecbbc1c	Final commit with all fixes

Project Scope

The audit covered the following files:

Filename	Link
Voting.sol	Voting.sol

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	1
Low	9

ID	Name	Severity	Status
M-1	It is possible to vote during the objection time if the proposal was rejected	Medium	Acknowledged
L-1	Using unsuitable visibility modifier	Low	Fixed
L-2	The vote is open for objection during the main voting time	Low	Fixed
L-3	Double check of the `_supports` variable	Low	Fixed
L-4	Unnecessary check	Low	Fixed
L-5	Gas optimization using `vote_.yea`	Low	Fixed
L-6	A malicious voter can potentially block a vote	Low	Acknowledged
L-7	Confusing naming can lead to vulnerabilities in the future code edits	Low	Fixed
L-8	Missed NatSpec parameters descriptions for public methods	Low	Fixed
L-9	The "last-minute" vote problem still persists for vote objection	Low	Acknowledged

1.6 Conclusion

Smart contracts have been audited and several suspicious places have been detected. During the audit, no critical problems were found, no high, one medium, and nine low issues were identified. After working on the reported findings, all of them were confirmed and fixed by the client.

Final commit identifier with all fixes: 4f7646fbc90ac31fada5ed9e95669fc4aecbbc1c

File name	Contract deployed on mainnet
Voting.sol	0x72fb5253AD16307B9E773d2A78CaC58E309d5Ba4

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	It is possible to vote during the objection time if the proposal was rejected
File	Voting.sol#L192
Severity	Medium
Status	Acknowledged

Description

At line Voting.sol#L192 during the objection time, the `vote()` function checks if the vote is `no`, if the vote is open for objection and if the voter has balance. But there is no check wether the vote was rejected or not. So it is possible to vote `no` for a proposal even if it was rejected during the voting time.

Recommendation

It is recommended to add a check if the vote was rejected.

Client's commentary

Acknowledged. The issue is harmless, while the fix will spend extra gas for every vote.

2.4 Low

L-1	Using unsuitable visibility modifier
File	Voting.sol#L267
Severity	Low
Status	Fixed in 52169ab6

Description

The visibility of the `canObject()` function at line Voting.sol#L267 which is not called internally from the same contract should be changed to `external` to save gas.

Recommendation

It is recommended to change `public` to `external`.

L-2	The vote is open for objection during the main voting time
File	Voting.sol#L475
Severity	Low
Status	Fixed in 1aa5fc6e, c96387aa, 32d56a74

Description

At line Voting.sol#L475
the `_isVoteOpenForObjection()` function returns `true` during the main voting time. It can be potentially misleading in functions `canObject()` and `getVote()` at the following lines:

- Voting.sol#L267
- Voting.sol#L316.

Recommendation

It is recommended to change the `_isVoteOpenForObjection()` function to return `true` only during the objection time or to rename the function to better indicate its purpose.

L-3	Double check of the `_supports` variable
File	Voting.sol#L374-L380
Severity	Low
Status	Fixed in 4587918f

Description

At line Voting.sol#L374-L380 there is a double check of variable `_supports`.

The assignment at line Voting.sol#L380 can be moved inside the if-else statement to save gas. Additionally, the if-else statement is more readable than a ternary operator.

Recommendation

It is recommended to change to:

```
if (_supports) {
    vote_.yea = vote_.yea.add(voterStake);
    vote_.voters[_voter] = VoterState.Yea;
} else {
    vote_.nay = vote_.nay.add(voterStake);
    vote_.voters[_voter] = VoterState.Nay;
}
```

L-4	Unnecessary check
File	Voting.sol#L423
Severity	Low
Status	Fixed in 1aa5fc6e, c96387aa, 32d56a74

Description

At line Voting.sol#L423 the check is unnecessary since the check at line Voting.sol#L428 returns `true` if the vote is open during the voting time. The vote cannot be executed until the end of the objection time.

Recommendation

Remove this check or refactor the `_isVoteOpen()` and `_isVoteOpenForObjection()` logic.

L-5	Gas optimization using `vote_.yea`
File	Voting.sol#L433-L440
Severity	Low
Status	Fixed in 16a5b868

Description

At line Voting.sol#L433-L440 it is possible to save gas by reading `vote_.yea` into a memory variable.

Recommendation

For example:

```
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;
}
```

L-6	A malicious voter can potentially block a vote
File	Voting.sol#L368
Severity	Low
Status	Acknowledged

Description

At line Voting.sol#L368 a voter can change their vote.
The malicious voter can initially vote `yes` misleading other users about their intention, then change their vote to `no` at the last moment potentially blocking the proposal.

Recommendation

It is recommended to check to prevent the last-minute `no` attacks against proposals.

Client's commentary

It's intentional, because we don't want to lose a possibility to convince voters to change their decision if some disastrous outcome is possible. In exchange we got this possibility for malicious voter to unexpectedly block the voting, but the possible harm is limited in this case.Also, we want DAO voters to be aware of these caveats too, so we are going to add a paragraph about to https://lido.fi/governance

L-7	Confusing naming can lead to vulnerabilities in the future code edits
File	Voting.sol#L384
Severity	Low
Status	Fixed in 1aa5fc6e, c96387aa, 32d56a74, 4f7646fb

Description

A voting passes two stages:

1. Vote is open for yeas and nays
2. Vote is open for nays only

To check for those stages these methods are used:

1. `_isVoteOpen`, `_canVote`, `canVote`
2. `_isVoteOpenForObjection`, `_canObject`, `canObject`

Namings to check that the voting is in the first stage are confusing, because name `_isVoteOpen` doesn't clearly say that it is checking the first stage only and for the second stage of voting it will return `false`. The same thing refers to `canVote()` and `_canVote()`. Their names suggest that they should return `true` for any unfinished voting, while they return `true` only if the voting is in the first stage.

This confusion may lead to bugs in the future if the contract code base becomes larger and it will become harder to keep in mind that the behavior of some methods differs from what their name seems to imply.

There are already cases of the incorrect use of methods in the code:
Voting.sol#L384

```
function _vote(uint256 _voteId, bool _supports, address _voter) internal {
  ...
  if (!_isVoteOpen(vote_)) { // objection phase
  ...
  emit CastObjection(_voteId, _voter, voterStake);
}
```

It is obvious that instead of `!_isVoteOpen(vote_)` you should use `_isVoteOpenForObjection(vote_)`. It is not a vulnerability now because in the current context these two lines are interchangeble.

However, if in the future the development team decides to add another voting phase, they will have to find and fix all the lines with the incorrect use of methods and if the code base becomes large and the namings would be still confusing it will be easy to miss some line and create a vulnerability.

Recommendation

Change method names to be more clear about what they really do. For example, instead of `_isVoteOpen` you could use `_isVoteOpenFirstStage`. And instead of `_canVote` you could use `_canVoteFirstStage`.

The `canVote` method is a public method so it may be better to keep its name for API compatibility. But perhabs it should be marked as deprecated in the code comments and another,`canVoteFirstStage` method should be added.

L-8	Missed NatSpec parameters descriptions for public methods
File	Voting.sol#L234
Severity	Low
Status	Fixed in 3fccffe4

Description

Voting.sol#L234
Voting.sol#L247
Voting.sol#L257
Voting.sol#L267
Voting.sol#L324

Recommendation

We recommend adding descriptions for the mentioned public methods parameters.

L-9	The "last-minute" vote problem still persists for vote objection
File	
Severity	Low
Status	Acknowledged

Description

The problem with the "last-minute" vote in this scheme is not fully mitigated, because a possibility of a vote to be denied at the last moment still persists. An attacker can object and deny the voting at the last moment like it could be done with the previous version of voting.

Recommendation

Should be simply acknowledged by the Lido team.

Client's commentary

It's a compromise that we're aware of. From our point of view, the possibility of passing the untelegraphed changes through the DAO is much more risky than last-minute blocking, whose possible impact is limited. Also, we want DAO voters to be aware of these caveats too, so we are going to add a paragraph about to https://lido.fi/governance

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes