LIDO SANITY CHECKER SECURITY AUDIT REPORT

Jul 31, 2024

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 the Client. 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 clients' 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 reports 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 Client with an interim report.

5. Bug fixing & re-audit:

- The Client 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 Client 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

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit 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 of funds.
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 `OracleReportSanityChecker` contract is actively used by the Lido protocol to ensure that the input data from oracle reports is correct and cannot update the protocol to an inconsistent state. Most of the checks are conducted via `view` functions and only checks previously set limits. The `checkAccountingOracleReport` function is not a `view` function and it updates storage on every call to this function, so it is crucial that this function can be called only from approved addresses.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Lido
Project name	Sanity Checker
Timeline	June 21 2024 - July 23 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
21.06.2024	efeff81c18f85451ebf98e8fd8bb78b8eb0095f6	Commit for the audit
23.07.2024	f6deb4bcd4f1a05a7336111c0a139fcebcae6b68	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
contracts/0.8.9/lib/SafeCastExt.sol	SafeCastExt.sol
contracts/0.8.9/sanity_checks/OracleReportSanityChecker.sol	OracleReportSanityChecker.sol

Deployments

Deployment verification for this particular scope will not be conducted because it is not planned to be deployed in the current configuration. Verification will be conducted later for the updated scope after additional audits.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	8

ID	Name	Severity	Status
L-1	Add a check for the allowed range for `maxPositiveTokenRebase`	Low	Acknowledged
L-2	Change the access role name for the `secondOpinionOracle` function	Low	Acknowledged
L-3	Unused error	Low	Fixed
L-4	Add comments to functions and fix typo	Low	Fixed
L-5	The value from `storage` is used instead of `memory`	Low	Fixed
L-6	`churnLimit` can be zero	Low	Acknowledged
L-7	Incorrect revert	Low	Fixed
L-8	Unobvious durations of time in `_checkCLBalanceDecrease()`	Low	Acknowledged

1.6 Conclusion

During the Lido Sanity Checker security audit, the following attack vectors were checked:

1. **Theoretical case of unstake of all the tokens from the protocol doesn't lead to any errors.** It was checked that in a hypothetical situation of all tokens being unstaked there won't be any unexpected underflows or reverts inside the sanity checker. The `_withdrawalVaultBalance` parameter is accounted for and withdrawals are not considered as a negative consensus layer balance rebase.

2. **`reportData` array elements are correctly used during calculations.** There are `_sumNegativeRebasesNotOlderThan` and `_exitedValidatorsAtTimestamp` functions which use Consensus Layer negative balance change (if it ever happened) and data about exited validators at a particular timestamp previously pushed to `reportData`. Loops inside the mentioned functions are implemented correctly with the correct boundaries set.

3. **All the config functions have proper access roles configured.** There is a `LimitsList` struct that stores specific restrictions that are used during the sanity checks. These restrictions can be configured via a specific restricted function. There are special roles for each function which ensure that it is impossible to change any value by someone not granted a specific role.

4. **Every restriction parameter has a special setter function and is checked for correct boundaries.** There is an internal `_updateLimits` function which is called on every sanity check parameter update. That function correctly checks boundaries for each parameter being set.

5. **There are no possible unexpected reverts.** All the checks for Consensus Layer and Execution Layer balances, together with the data regarding withdrawal vault balance and validators count, are implemented correctly. It was checked that it is impossible for the sanity checker to get stuck in a specific state which will always revert. It was also checked that loops iterating over the `reportData` array won't lead to out-of-gas errors.

6. **Types casting is implemented correctly.** There are necessary checks that ensure that the data inside `uint256` variables can be packed into variables with a smaller size (in terms of bytes). All the conversions are done via the `LimitsListPacker` and `LimitsListUnpacker` libraries.

2. FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Add a check for the allowed range for `maxPositiveTokenRebase`
Severity	Low
Status	Acknowledged

Description

There is a `maxPositiveTokenRebase` parameter check at the OracleReportSanityChecker.sol#L880. This check allows `maxPositiveTokenRebase` to be assigned a value between `1` and `type(uint64).max`. But `maxPositiveTokenRebase` should be in range from `1` to `1e9` or equal to `type(uint64).max`.

Recommendation

We recommend adding a more strict check for the `maxPositiveTokenRebase` parameter.

Client's commentary

> Looks minor. Also it's unrelated to the negative rebase itself.

L-2	Change the access role name for the `secondOpinionOracle` function
Severity	Low
Status	Acknowledged

Description

There is a `setOracleReportLimits` function at the OracleReportSanityChecker.sol#L269. It has a restricted access only for the `ALL_LIMITS_MANAGER_ROLE` role. This function also allows setting the `secondOpinionOracle` address.

Recommendation

We recommend changing the role name for the `setOracleReportLimits` function to reflect all allowed actions.

Client's commentary

> Looks minor. I would say a second opinion address is just a part of all limits configuration.

L-3	Unused error
Severity	Low
Status	Fixed in f6deb4bc

Description

There is an unused error `IncorrectExitedValidators`: OracleReportSanityChecker.sol#L926.

Recommendation

We recommend removing the unused error.

L-4	Add comments to functions and fix typo
Severity	Low
Status	Fixed in f6deb4bc

Description

There is a typo `Invactivity` OracleReportSanityChecker.sol#L121.

There is a missing comment that the function also changes `secondOpinionOracle` OracleReportSanityChecker.sol#L266.

There is a missing comment that the function also changes `clBalanceOraclesErrorUpperBPLimit` value OracleReportSanityChecker.sol#L379.

Recommendation

We recommend fixing the mentioned typos and adding the necessary comments.

L-5	The value from `storage` is used instead of `memory`
Severity	Low
Status	Fixed in f6deb4bc

Description

There is a place in the code where the value from `storage` is used instead of a copy of the exact same value in the `memory`: OracleReportSanityChecker.sol#L705-L706.

Recommendation

We recommend using `_limitsList` instead of `_limits` in the `_checkCLBalanceDecrease` function.

L-6	`churnLimit` can be zero
Severity	Low
Status	Acknowledged

Description

If `churnValidatorsPerDayLimit` is small and the frequency of the report is high, then there is a chance that `churnLimit` will be rounded down to zero: OracleReportSanityChecker.sol#L780.

Recommendation

We recommend adding a minimum value limit for `churnLimit`.

Client's commentary

> Minor and impossible in practice. Also not related to negative rebase.

L-7	Incorrect revert
Severity	Low
Status	Fixed in f6deb4bc

Description

`IncorrectAppearedValidators` should show `churnLimit`, but instead of it, the revert shows `_appearedValidators`: OracleReportSanityChecker.sol#L782.

Recommendation

We recommend updating the specification of the `IncorrectAppearedValidators` revert.

L-8	Unobvious durations of time in `_checkCLBalanceDecrease()`
Severity	Low
Status	Acknowledged

Description

Two-time durations (`18 days` and `54 days`) are used directly in `_checkCLBalanceDecrease()` which is not obvious and explained only in the documentation. This might reduce code readability.

OracleReportSanityChecker.sol#L703-L706

Recommendation

We recommend using constants for `18 days` and `54 days` with brief comments.

Client's commentary

> No space left on the stack for explicit variable names.Explanation for 18 and 54 days present in the specification.

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