Lido On Polygon Smart Contracts Security Audit Report for PR#67

April 25, 2022

0XORIO

List of contents

1 Introduction

This report consists of the audit results performed by **Oxorio team** on the Lido On Polygon project, at the request of the Lido team. The audited code can be found in the public Lido for Polygon Github Repository.

The main goals of this audit are:

- to review the changes introduced in this PR for Lido On Polygon's solidity implementation for its decentralized staking model,
- to study potential security vulnerabilities, its general design and architecture, that may be changed by this PR
- to uncover errors and bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations which could improve its quality as a whole.

1.1 Disclaimer

Note that as of the date of publishing, the contents of this document reflect the current understanding of investigated security patterns and the state of art regarding smart contract security. Given the size of the project, the findings detailed here are not to be considered exhaustive. Further testing and auditing are recommended after the covered issues would be fixed.

1.2 Methodology

On the methodology part, we do the following audit steps:

1. Manual code study
Manually code study to find out the errors and bugs.
2. Check the code against the list of known vulnerabilities
Verification process of the code against the constantly updated list of already known vulnerabilities maintained by the company.
3. Architecture and structure check of the security model
Study project documentation and its comparison against the code including the study of the comments and other technical papers.
4. Result's cross-check by different auditors
Normally the research of the project is made by more than two auditors. After that, there is a step of the mutual cross-check process of audit results between different task performers.
5. Report consolidation
Consolidation of the audited report from multiple auditors.
6. Reaudit of new editions

After the client's review and fixes, the founded issues are being double-checked. The results are provided in the new audit version.

7. Audit report publication on the official website

The final audit version is provided to the client and also published on the official website of the company.

1.3 Structure of the Document

This report contains the list of issues and comments divided by their severity and status levels. Each issue is aligned with the code file that it is represented in for the readability of the report. For an easy way of navigation, a table of contents is provided at the beginning of the report.

1.4 Documentation

For this audit, the following sources of truth about how the Lido On Polygon smart contracts should work were used:

- main GitHub repository of the project
- Almanac documentation provided by the client.

These were considered the specification, and when discrepancies arose with the actual code behaviour, there were consultations directly with the Lido team.

1.5 About Oxorio

Oxorio is a young but rapidly growing audit and consulting company in the field of the blockchain industry, providing consulting and security audits for organizations from all over the world. Oxorio has participated in multiple blockchain projects where smart contract systems were designed and deployed by the company.

Oxorio is the creator, maintainer, and major contributor of several blockchain projects and employs more than 5 blockchain specialists to analyze and develop smart contracts.

Clients include Lido, among others. More info at: oxor.io

1.6 Project overview

Lido on Polygon is a liquid staking solution for MATIC.

2 Scope of the Audit

The scope of the audit includes checking the following fixes in PR#67:

- Added the StMatic NFT amount to the total Pooled Matic.
- Added function calculatePendingBufferedTokens to calculate the total amount hold
- Fixed getMaticFromTokenId to return the request from stMatic contract

The audited commit identifier is b992dc60ccb638227022e912fab3face9097fc41

3 Findings Severity breakdown

3.1 Classification of Issues

The following severity levels were assigned to the issues described in the report :

- **CRITICAL**: A bug leading to assets theft, fund access locking, or any other loss of funds due to transfer to unauthorized parties.
- **MAJOR**: A bug that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
- **WARNING**: A bug that can break the intended contract logic or expose it to DDoS attacks.
- **INFO**: Minor issue or recommendation reported to / acknowledged by the client's team.

3.2 Findings' breakdown status

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

- **FIXED**: Recommended fixes have been made to the project code and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
- **NO ISSUE**: Finding does not affect the overall security of the project and does not violate the logic of its work
- **DISMISSED**: The issue or recommendation was dismissed by the client.
- **NEW**: Waiting for the project team's feedback.

4 Report

4.1 CRITICAL

No issues found.

4.2 MAJOR

4.2.1 Attackers can make `submit` and `requestWithdraw` fail with out-of-gas

Description

Steps:

1. Attackers deposit some MATIC
2. They call requestWithdraw(1) several times and receive NFTs. It cost them ~600k gas for each call.
3. They send those NFTs to StMatic.
4. Each call to getMaticFromTokenId costs ~12k gas.
5. claimTokens2StMatic will not help until stakeManager.epoch() >= lidoRequests.requestEpoch, then the attackers can repeat the attack.
    - Epoch is ~3 hours, withdrawalDelay is 80 epochs (see block explorer) therefore withdrawalDelay is ~240 hours or ~10 days.
    - This is in contradiction with the Polygon doc which says that the unbonding period is ~3 hours * 80 = 3-4 days.
    - But even so, that is quite a lot of time.

Results:
Function calls `calculatePendingBufferedTokens` => `getTotalPooledMatic` => `convertStMaticToMatic` / `convertMaticToStMatic` => `requestWithdraw` and `submit` will be locked.
Users will not be able to neither deposit nor submit.

The attack costs 1800kk gas for 3000 NFTs, on a good day and time it could be ~36eth. It will increase the gas price of the `requestWithdraw` and `submit` calls up to 36kk, more than the average block gas limit. Even a smaller amount would make the price of participation too high so that almost no one would have any incentive to do it.

In addition to this, if there are many withdrawals using `withdrawTotalDelegated` at some point this call will start to fail with out-of-gas. Before that, every call to `requestWithdraw` and `submit` will cast more and more. If there are 100 000 elements in `getOwnedTokens()` the cost will be ~ 300 000 000 gas.

Recommendation

We recommend removing the token from the owner2Tokens[anAddress] array on burn.

Update

Fixed at d51fef738d09ece817050a25ae72b07b25a5919c and c0359db30314dd19de5fc18e0e2000f853f3f56f.

4.3 WARNING

No issues found.

4.4 INFO

4.4.1 Inconsistent naming for `ValidatorProxy.operator` variable

Description

`ValidatorProxy.operator` at ValidatorProxy.sol#L19 is actually a `nodeOperatorRegistry`. The same goes for a function `setOperator` at ValidatorProxy.sol#L54

Recommendation

We recommend renaming `operator` variable and function `setOperator` to `operatorRegistry` and `setOperatorRegistry` respectively.

4.4.2 Misleading revert message in case of pending funds

Description

If there are pending funds, the user may not be able to withdraw and will get a general error "Too much to withdraw" (StMATIC.sol#L184). Please consider adding a relevant error message to UI and to explain users that they should wait X days as the funds have not been received yet.

Recommendation

We suggest handling this case in UI.

4.4.3 Magic numbers are used

Description

Magic numbers at StMATIC.sol#L897 decrease code readability. A reader won't understand what they mean without context. Moreover, it complicates further code maintenance.

```
uint256 exchangeRatePrecision = validatorId < 8 ? 100 : 10**29;
```

Recommendation

We recommend using constants with descriptive names or adding a comment explaining what is happening.

Update

Fixed at 1d8e4696d9a225f9079bcaff1cb8a60c8eff8131.

4.5 Results

Level	Amount
CRITICAL	0
MAJOR	1
WARNING	0
INFO	3
Total	3

5 Conclusion

We have reviewed PR#67 and concluded that the introduced fixes work as they are supposed to. However, there was one major issue not related to this PR content. That issue have been fixed. In addition to this, we have several minor suggestions that may help to increase the code readability.