Lido On Polygon Smart Contracts Security Audit Report for PR#69

April 25, 2022

OXORIO

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 changes made in PR#69 to the following contracts:

- PoLidoNFT.sol
- StMATIC.sol

The audited commit identifier is 1d8e4696d9a225f9079bcaff1cb8a60c8eff8131

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

No issues found.

4.3 WARNING

No issues found.

4.4 INFO

4.4.1 Not descriptive variable name `token2Index`

Description

The `token2Index` mapping at PoLidoNFT.sol#L25 actually points to an NFT id index within an array of an owner's tokens inside `owner2Tokens`.

Recommendation

We recommend renaming `token2Index` variable to `tokenIdToIndexInOwnerTokens` or similar.

4.4.2 Reducing readability by using mutable variable

Description

Incrementing `currentIndex` variable on the next line at PoLidoNFT.sol#L59 makes the code less readable and cost a little more gas.

```
uint256 currentIndex = tokenIdIndex;
currentIndex++;
```

Recommendation

We recommend making `currentIndex` immutable:

```
uint256 currentIndex = tokenIdIndex + 1;
```

4.4.3 Not descriptive variable name `tokenIndex`

Description

There are several indices in the scope, and the `tokenIndex` variable at PoLidoNFT.sol#L124 is actually an index of burned NFT id within the tokens array inside `owner2Tokens`.

Recommendation

We recommend renaming `tokenIndex` to `burnedTokenIndexInOwnerTokens`.

4.4.4 Not descriptive variable name `length`

Description

The `length` variable at PoLidoNFT.sol#L125 actually is length of `ownerTokens` array.

Recommendation

We recommend renaming `length` to `ownerTokensLength`.

4.4.5 Not saving index calculation result to a variable with a meaningful name may decrease readability

Description

`length - 1` expression at PoLidoNFT.sol#L127 makes the code less readable because a lot of things happen in that part of code. Introducing a variable may help to decrease a cognitive load.

Recommendation

We recommend using `lastOwnerTokensIndex`:

```
uint256 lastOwnerTokensIndex = length - 1;
if (tokenIndex != lastOwnerTokensIndex && length != 1) {
```

4.4.6 One-letter variable `t`

Description

One-letter `t` variable at PoLidoNFT.sol#L128 makes the code less readable. It is actually the last id within `ownerTokens` array.

Recommendation

We recommend renaming `t` to `lastOwnerTokenId`.

4.4.7 Duplicate storage reading

Description

The second storage reading of `ownerTokens.length` at PoLidoNFT.sol#L128 and `ownerTokens[ownerTokens.length - 1]` at PoLidoNFT.sol#L130.

Recommendation

We recommend reusing memory variable to save gas:

```
 uint256 t = ownerTokens[length - 1];
token2Index[t] = tokenIndex;
ownerTokens[tokenIndex] = t;
```

4.4.8 Complicated id burning section of code

Description

The implementation of not the last token id burning logic at PoLidoNFT.sol#L127 is quite difficult to understand. This issue summarize all the issues above.

Recommendation

We suggest renaming variables to more descriptive names and adding a comment with an example to explain how burning occurs.

A comment with an example before renaming the variables may look like (draft):

- The `ownerTokens` array is [111, 222, 333] and `token2Index` is { 111: 0, 222: 1, 333: 2 }.
- The burned NFT id `tokenId` is 222. Then index of burned id within `ownerTokens` array `tokenIndex` is 1.
- The last id within `ownerTokens` array `t` is 333; the length of `ownerTokens` array `length` is 3; last id index within `ownerTokens` array is 2.
- `token2Index[333]` is 2, `token2Index[222]` is 1.

- `ownerTokens[1]` is 222, `ownerToken[2]` is 333.
- Change the index of the last id `t` from 2 to burned id index, 1, so `token2Index[333]` is 1 now and `token2Index[222]` is 1 as well. The `token2Index` now is {111: 0, 222: 1, 333: 1}.
- Next, update `ownerTokens` array to get `t` instead of burned id: `ownerTokens[tokenId] = 333`. Therefore, `ownerTokens[1]` is 333 and `ownerTokens[2]` is 333 as well. The `ownerTokens` array now is [111, 333, 333].
- Finally, remove burned id index from `token2Index` and the last id from `ownerTokens`. The `ownerTokens` array is [111, 333] and `token2Index` is {111: 0, 222: 0, 333: 1}.

A rewritten code with an example may look like (draft):

```
 // token2Index renamed to tokenIdToIndexInOwnerTokens;
uint256 burnedTokenIndexInOwnerTokens =
tokenIdToIndexInOwnerTokens[tokenId];
uint256 ownerTokensLength = ownerTokens.length;
// ...
uint256 lastOwnerTokensIndex = ownerTokensLength - 1;
uint256 lastOwnerTokenId = ownerTokens[lastOwnerTokensIndex];
// Example:
// tokenId (the burned one) is 22, burnedTokenIndexInOwnerTokens 2
// lastOwnerTokenId is 333, ownerTokensLength = 4 =>
lastOwnerTokensIndex = 3 => index in ownerTokens is 3
// => :
// tokenIdToIndexInOwnerTokens[333] => 3,
tokenIdToIndexInOwnerTokens[22] => 2
// ownerTokens[2] => 22, ownerToken[3] => 333

// Make lastOwnerTokenId, 333, change it's index in ownerTokens to
burnedTokenIndexInOwnerTokens, 2
// So tokenIdToIndexInOwnerTokens[333] => 2,
tokenIdToIndexInOwnerTokens[22] => 2
// So when someone wants to read lastToken it will get burned token
index
tokenIdToIndexInOwnerTokens[lastOwnerTokenId] =
burnedTokenIndexInOwnerTokens;
// Then update ownerTokens, make burned token the last one
// ownerTokens[burnedTokenIndexInOwnerTokens] = 333;
// Results:
// tokenIdToIndexInOwnerTokens[333] => 2,
tokenIdToIndexInOwnerTokens[22] => 2
// ownerTokens[2] => 333, ownerToken[3] => 333
// We need to remove tokenIdToIndexInOwnerTokens[tokenId] and the last
element from ownerTokens
ownerTokens[burnedTokenIndexInOwnerTokens] = lastOwnerTokenId;
```

4.4.9 `0 is used in several meanings in token2Index`

Description

When `token2Index` is removed it is set to 0 PoLidoNFT.sol#L134.

```
token2Index[tokenId] = 0;
```

But 0 is also a valid index. It may lead to confusion while reading the code or changing it. E.g. checking if index is set by comparison with 0. Or some other subtle bugs. It may also lead to confusion in 3rd parties who use `token2Index` from your contract.

Recommendation

Consider rewriting the logic so 0 is not ambiguous (not set or index 0).

4.5 Results

Level	Amount
CRITICAL	0
MAJOR	0
WARNING	0
INFO	9
Total	9

5 Conclusion

Changes in 2 smart contracts have been audited and no critical, major or warning issues were found. Some minor recommendations for code readability and best practices were marked as informational.