Lido L2 Smart Contracts Security Audit Report

July 21, 2022

OXORIO

List 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.

1.2 Security Assessment Methodology

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

1. Project architecture review:

Manually code study of the architecture of the code based on the source code only 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. Final audit report publication

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

1.2.1 Severity Level Reference

Findings discovered during the audit are classified as follows: Every issue in this report was assigned a severity level from the following:

- **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.

1.2.2 Status Level Reference

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:

- **NEW**: Waiting for the project team's feedback.
- **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.

1.3 Project overview

This project contains the implementation of the L2 ERC20 token bridges for Arbitrum and Optimism chains. The current solution allows transferring ERC20 tokens between L1 and L2 chains.

1.4 Audit Scope

The scope of the audit includes the following smart contracts at:

- BridgeableTokens.sol
- BridgingManager.sol
- token/ERC20Core.sol
- token/ERC20Metadata.sol
- token/ERC20Bridged.sol
- proxy/OssifiableProxy.sol
- optimism/CrossDomainEnabled.sol
- optimism/L1ERC20TokenBridge.sol
- optimism/L2ERC20TokenBridge.sol

- arbitrum/InterchainERC20TokenGateway.sol
- arbitrum/L1CrossDomainEnabled.sol
- arbitrum/L1ERC20TokenGateway.sol
- arbitrum/L2CrossDomainEnabled.sol
- arbitrum/L2ERC20TokenGateway.sol

The audited commit identifier is 83ad65669c25dfa76e793f46b2e54f12c8fee0af

1.4.1 Assumptions

- Reverted transaction between Optimism and L1 may always be replayed

2 Report

2.1 CRITICAL

2.1.1 User looses their funds if `maxSubmissionCost` is low

Severity	CRITICAL
Status	ACKNOWLEDGED

Description

L1ERC20TokenGateway.sol#L61

There is no checks for `maxSubmissionCost` value. The Arbitrum docs says:

> If an L1 transaction underpays for a retryable ticket's base submission free, the retryable ticket creation on L2 simply fails. Given that this potentially breaks the atomicity of the L1 / L2 transactions, applications should avoid this scenario. ... it is highly recommended that applications judiciously overpay relative to the current price.

After that funds are impossible to restore, they are locked on the contract.

Recommendation

Check that `maxSubmissionCost` is at least `ArbRetryableTx.getSubmissionPrice`. The current base submission fee returned by `ArbRetryableTx.getSubmissionPrice` increases once every 24 hour period by at most 50% of its current value. Giving the risk of loosing bridged funds it's best to set it higher than the current.

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/4

As shown in the description of the finding, the problem concerns not only the `L1ERC20TokenGateway` contract but the current Arbitrum's L1 -> L2 messages passing design as a whole. Such kind of failure, for example, also might take place in the default `L1GatewayRouter` contract. At this point, there is no convenient way to validate the passed `maxSubmissionCost` value from the L1 chain. The recommendation provided by the Oxorio team is not feasible because the `ArbRetryableTx` contract does not exist on the Ethereum chain, only on Arbitrum. However, additional validation for the zero value of

`maxSubmissionCost` was added into the `L1ERC20TokenGateway`, to provide same garuantees as `L1GatewayRouter`.

In the next update of the Arbitrum protocol titled Nitro, the validation logic for the submission fee will be moved to the L1 chain. According to the documentation of the Arbitrum protocol:

> In a future release, the base submission fee will be calculated using the 1559 `BASE_FEE` and collected directly at L1; underpayment will simply result in the L1 transaction reverting...

The devnet with Nitro update has been successfully launched by the Arbitrum team. The next stage is the official testnet launch and mainnet release. This update will eliminate the issue. The updated version of the Inbox contract that validates the passed `maxSubmissionCost` value might be found here: https://github.com/OffchainLabs/nitro/blob/21ce4812a7d202e41c165b8ec1c9154801325e1c/contracts/src/bridge/Inbox.sol#L430-L432.

But even the usage of the current version of the Arbitrum with an incorrect `maxSubmissionCost` value will not lead to a complete loss of the user funds. As both `L1ERC20TokenGateway` and `L2ERC20TokenGateway` are upgradable, user funds might be restored via the upgrade of the gateway on the same code version but with additional initialization logic. The new initializer might mint the required amount of tokens on the L2 or transfer tokens back to the user on the L1. Of course, such a procedure is an extreme measure but might be used in case of a massive malfunction in the Arbitrum's bridge UI in case bridging transactions would have been sent with a low base submission fee.

As it was mentioned, the `L1ERC20TokenGateway` and `L2ERC20TokenGateway` are supposed to be used with the default Arbitrum's router gateway via official Arbitrum's bridging UI. Such usage guarantees that `maxSubmissionCost` for outbound transfers will have the correct value. In case of a standalone usage of the contract user, **MUST** follow the recommendations of the Offchain Labs and **MAKE SURE** that passed `maxSubmissionCost` has the correct value.

2.2 MAJOR

2.2.1 User may loose their funds if `msg.value`, `maxGas_` or `gasPriceBid_` is low

Severity	MAJOR
Status	FIXED

Description

L1ERC20TokenGateway.sol#L80-L82

The `maxGas_`, `gasPriceBid_` and `msg.value` values are not checked. If `maxGas_`* `gasPriceBid_` is not enough the transaction won't be executed immediately. The same goes for `msg.value`. And users will have to execute it themself or ask someone to do it. It may take a lot of time for the non-tech-savvy user.

According to Arbitrum docs the created ticket expires in 7 days and if no one executes it the funds would be lost:

> If no gas is provided or the execution reverts, it will be placed in the L2 retry buffer, where any user can re-execute for some fixed period (roughly one week).

Recommendation

Add a check as in L1GatewayRouter:

```
uint256 expectedEth = _maxSubmissionCost + (_maxGas * _gasPriceBid);
require(msg.value == expectedEth, "WRONG_ETH_VALUE");
```

You may also want to consider implementing a way to recover funds in case they are locked because the fix above is not bulletproof.

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/5

The issue was fixed according to the provided recommendations. It should be noted that even if the retryable ticket expires, the money still might be returned to the user via the contracts upgrade. See the previous section for details.

2.3 WARNING

2.3.1 Not all the ERC20 tokens are supported

Severity	WARNING
Status	FIXED

Description

According to your documentation (Arbitrum, Optimism) this implementation supports all ERC20 compatible tokens:

> ...implementation of the bridging of the ERC20 compatible tokens...

But rebasing tokens, tokens with fee on transfer are not supported. Also some tokens with callback on transfer (`ERC777`) or other peculiar tokens may not be supported because they has additional logic not handled by `ERC20Bridged`.

Also your implementation require Arbitrum team to call `setGateways` if you want to use the Router which may not be suitable for all the tokens.

Recommendation

Make sure you don't use this bridge with tokens mentioned above. Update the docs.

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/14

The bridges documentation was updated according to the provided recommendations.

2.4 INFO

2.4.1 Pragma is not locked

Severity	INFO
Status	FIXED

Description

In all the audited contracts pragma is not locked.

Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors. Also solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

Recommendation

Lock the pragma, e.g.

```
pragma solidity 0.8.13;
```

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/6

The version of the contract was locked to version `0.8.10`. According to the list of the known bugs in the different versions of the solidity: https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json, this version is the best in terms of the number of known bugs (3 at the moment) and the maturity (more than 6 months since the release).

2.4.2 Public function could be declared external

Severity	INFO
Status	FIXED

Description

`public` functions that are never called by the contract should be declared external to save gas. BridgingManager.sol#L48-L50

```
function isInitialized() public view returns (bool) {
    return _loadState().isInitialized;
}
```

The same goes for `transfer`, `approve`, `transferFrom` in `ERC20Core` and `ERC20Bridged.bridgeMint`.

Recommendation

Use the `external` attribute for functions never called from the contract.

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/15

In the new versions of the solidity, the visibility modifier does not affect the gas costs of the method. In our case, gas costs stay the same also for older solidity versions because the method has no reference types arguments. See details in the documentation: https://docs.soliditylang.org/en/v0.8.10/types.html#reference-types

Despite of this, the visibility of the methods: `transfer`, `approve`, `transferFrom` in the `ERC20Core` contract and `bridgeMint` in the `ERC20Bridged` contract were changed from

`public` to `external` because these methods are not supposed to be called by the inherited contracts.

2.4.3 Import is not used

Severity	INFO
Status	FIXED

Description

IERC20Bridged.sol#L7-L11

```
import {IERC20Metadata} from "./IERC20Metadata.sol";

/// @author psirex
/// @notice Extends the ERC20 functionality that allows the bridge
to mint/burn tokens
interface IERC20Bridged is IERC20 {
```

Recommendation

Use the import, `IERC20Bridged` should extend `IERC20Metadata`

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/7

The `IERC20Metadata` interface was removed from the `IERC20Bridged.sol` file.

2.4.4 Modifier `onlyAdmin` may be misleading

Severity	INFO
Status	FIXED

Description

OssifiableProxy.sol#L93-L97

```
modifier onlyAdmin() {
    address admin = _getAdmin();
    if (admin != address(0) && msg.sender != admin) {
```

```
        revert ErrorNotAdmin();
    }
```

Modifier name suggests that only admin can use a function with it. But if a proxy is ossified anyone can use it. It's not an issue right now because `whenNotOssified` is added everywhere. But it may lead to errors in the future because the behavior may not be expected.

Recommendation

Merge `onlyAdmin` and `whenNotOssified` modifiers. Only check for `msg.sender != admin`.

Update

Lido's response

Fix: https://github.com/lidofinance/lido-l2/pull/8

According to the recommendations, the `whenOssified()` modifier was removed, and its logic was added to the `onlyAdmin()` modifier.

2.4.5 Redundant checks

Severity	INFO
Status	NO_ISSUE

Description

`L1ERC20TokenBridge` and `L2ERC20TokenBridge` support only one pair of tokens. There is no need to pass `l1token` and `l2token` as arguments and then check `onlySupportedL1Token`, `onlySupportedL2Token`. E.g. L1ERC20TokenBridge.sol#L44-L55

```
    function depositERC20(
        address l1Token_,
        address l2Token_,
        uint256 amount_,
        uint32 l2Gas_,
        bytes calldata data_
    )
        external
        whenDepositsEnabled
        onlySupportedL1Token(l1Token_)
```

```
        onlySupportedL2Token(l2Token_)
    {
```

Recommendation

Remove redundant checks to save gas.

Update

Lido's response

The `onlySupportedL1Token` and `onlySupportedL2Token` modifiers are required to prevent unintended usage of the bridge with the wrong pair of tokens. Also, the `l1Token` and `l2Token` arguments can't be removed because it violates the required interface to be compatible with Optimism's bridging UI.

2.4.6 Transaction censoring is possible

Severity	INFO
Status	ACKNOWLEDGED

Description

If `DEPOSITS_ENABLER_ROLE` and `DEPOSITS_DISABLER_ROLE` or `WITHDRAWALS_ENABLER_ROLE` and `WITHDRAWALS_DISABLER_ROLE` are the same entity it's possible to sandwich transactions disallowing withdrawals/deposits for some users which may not be expected.

Recommendation

This roles should not be given to entities that may have incentives to censoring.

Update

Lido's response

The full set of management roles will be granted only to the Lido DAO represented by the Aragon Agent. Additionally, `DEPOSITS_DISABLER_ROLE` and `WITHDRAWALS_DISABLER_ROLE` might be granted to the emergency multisig for fast bridge disabling in case of a bug or vulnerability.

3 Conclusion

The following table contains the total number of issues that were found during audit:

Level	Amount
CRITICAL	1
MAJOR	1
WARNING	1
INFO	6
Total	9

As stated in each particular issue, all issues identified have been correctly fixed or acknowledged by the client, so contracts are assumed as secure to use according to our security criteria and ready to deploy to mainnet. One minor info issue was marked as "no issue" after discussing with the Lido's team.

4 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.

Contacts:

- oxor.io
- ping@oxor.io
- github
- linkedin