BOBA NETWORK

SMART CONTRACT AUDIT

zokyo

August 25th, 2022 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

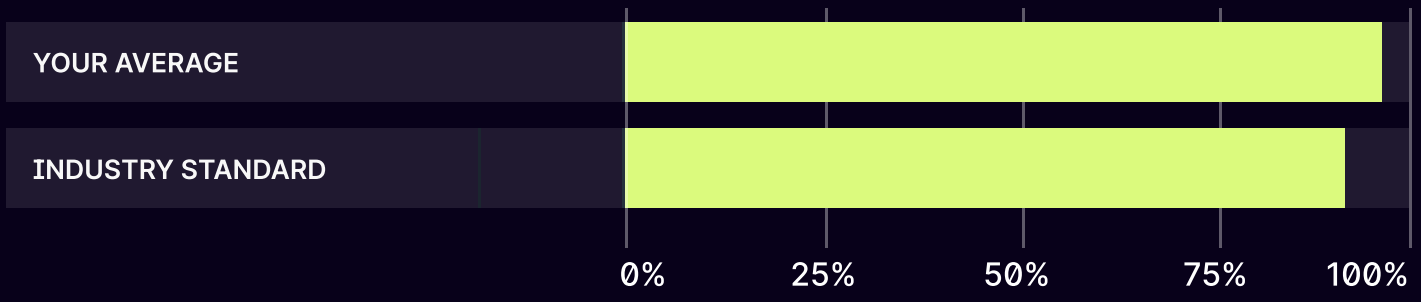
This document outlines the overall security of the Boba Network smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Boba Network smart contract codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit.

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Boba Network team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Boba Network repository.

Repository: https://github.com/bobanetwork/boba/tree/alt-l1/packages/boba/contracts/contracts/lzTokenBridge

Last commit: https://github.com/bobanetwork/boba/commit/3e22e35b8a3224bfcad95136ca7906a396c73c18

Within the scope of this audit, Zokyo auditors have reviewed the following contract(s):

- LzApp.sol
- LzLib.sol
- NonblockingLzApp.sol
- AltL1Bridge.sol
- EthBridge.sol

Throughout the review process, Zokyo Security ensures that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Boba Network smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.	04	Thorough manual review of the codebase, line by line.

Executive Summary

The Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in the “Complete Analysis” section.

There were no critical issues found during the auditing process. Medium, low, and informational issues were found. Not all of them were resolved by the Boba Network team.

All the mentioned findings may have an effect only in the case of specific conditions performed by the contract owner.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the client team and the client team are aware of it, but they have chosen to not solved it. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that it can lead to a significant loss, funds may be lost or allocated incorrectly.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

	Title	Risk
1	Pragma version lock	Low
2	Unused function parameter	Informational
3	Variable declaration after function	Informational
4	Unchecked possible zero address	Medium
5	Config param sanity check	Low
6	Config param sanity check	Low
7	No upgradability pattern used	Informational

COMPLETE ANALYSIS

LOW | ACKNOWLEDGED

Pragma version lock
It's recommended to have the same compiler version that the contracts were tested with the most. This way it reduces the risk of introducing unknown bugs. There are also different versions used throughout the project, for example `^0.8.0` in LzApp.sol and `^0.8.9` in EthBridge.sol.

Recommendation:
Lock pragma versions.

INFORMATIONAL | ACKNOWLEDGED

Unused function parameter
In contract EthBridge.sol, at lines 126 and 127, variables _srcAddress and _nonce are not being used throughout the function.

Recommendation:
Make sure the variables are not actually needed and remove them.

INFORMATIONAL | RESOLVED

Variable declaration after function
In contract LzApp.sol, at line 17 the variable failedMessages is declared after the internal function __NonblockingLzApp_init. Maintain a consistent order of variable and function declarations throughout the project.

Recommendation:
Move the variable declaration to the top of the file, before any function declaration. Do this in all contract files.

MEDIUM | RESOLVED

Unchecked possible zero address

In contract EthBridge.sol, at line 69 in function depositERC20To the _to function parameter can be zero and is not checked. The following function calls will not revert in case of a zero _to address, leading to undefined behavior, such as depositing tokens but not being able to receive them.

```
ftrace | funcSig
69    function depositERC20To(
70        address _l1Token,
71        address _l2Token,
72        address _to,
73        uint256 _amount,
74        address _zroPaymentAddress,
75        bytes memory _adapterParams,
76        bytes calldata _data
77    ) external virtual payable {
78        require(_to != address(0), "EthBridge: zero _to address");
79        _initiateERC20Deposit(_l1Token, _l2Token, msg.sender, _to
80    }
```

Recommendation:

Add a sanity check for the _to address to not be zero and revert otherwise.

LOW | ACKNOWLEDGED

Config param sanity check

In contract LzApp.sol, in the function setTrustedRemote, there's no sanity check for the _srcAddress parameter. This can generate a wrong assignment even if the route can only be accessed by the owner of the contract.

Recommendation:

Add a sanity check to prevent _srcAddress from being an empty slice of bytes.

LOW | ACKNOWLEDGED

Config param sanity check

In contract LzApp.sol, in the function forceResumeReceive, there's no sanity check for the _srcAddress parameter. This can generate a wrong assignment, even if the route can only be accessed by the owner of the contract.

Recommendation:

Add a sanity check to prevent _srcAddress from being an empty slice of bytes.

INFORMATIONAL | ACKNOWLEDGED

No upgradeability pattern used

In contracts LzApp.sol and NonnlockingLzApp.sol you are leaving an empty reserved space, for possible future upgrades and also using the OwnableUpgradeable and Initializer contracts. However, there's no upgradability mechanism used, such as Openzeppelin's Upgradable Proxy.

Recommendation:

If you're not intending to use this pattern, please refactor the contracts to reflect this, otherwise complete the implementation in this direction.

	EthBridge.so l	LzApp.sol	LzLib.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	NonblockingL zApp.sol	AltL1Bridge. sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting Boba Network in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Boba Network contract requirements for details about issuance amounts and how the system handles these.

LzApp

- ✓ Should be able to initialize properly (69ms)
- ✓ Should set the setMinDstGasLookup (43ms)
- ✓ Should force resume recieve
- ✓ Should setReceiveVersion
- ✓ Should setSendVersion
- ✓ Should isTrustedRemote
- ✓ Should be revert erc20 transfer with gas limit too low (108ms)
- ✓ Should transfer erc20 (117ms)
- ✓ Should be get config (52ms)
- ✓ Should revert lzReceive (43ms)
- ✓ Should allow lzReceive (120ms)
- ✓ Should set config
- ✓ should deposit erc20 (64ms)
- ✓ should revert the transaction with LzApp: minGasLimit not set (194ms)
- ✓ should revert the transaction with LzApp: destination chain is not a trusted source (255ms)

NonBlockingLzApp

- ✓ Should be able to allow erc20 withdrawal (63ms)
- ✓ Should retry (133ms)
- ✓ Should revert with NonblockingLzApp: caller must be LzApp

AltL1Bridge

- ✓ Should be able to initialize properly
- ✓ Should be able to allow withdrawals (50ms)
- ✓ Should be able to allow withdrawals to recipient (67ms)
- ✓ Should allow use of custom adapter params
- ✓ Should be able to allow deposits

ETHBridge

- ✓ Should be able to initialize properly
- ✓ Should be able to deposit erc20 (91ms)
- ✓ Should be able to deposit erc20 to recipient (54ms)

✓ Should allow use of custom adapter params
✓ Should be able to allow erc20 withdrawal (56ms)
28 passing (5s)

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
AltL1Bridge.sol	100	100	100	100
EthBridge.sol	100	100	100	100
LzApp.sol	100	100	100	100
LzLib.sol	100	100	100	100
NonblockingLzApp.sol	100	100	100	100
All files	**100**	**100**	**100**	**100**

We are grateful to have been given the opportunity to work with the Boba Network team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo's Security Team recommends that the Boba Network team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo