ackee blockchain

LayerZero

LzApp + Tokens Audit

3 May 2022

by Ackee Blockchain

Contents

1. Document Revisions

Revision	Date	Description
1.0	3 May 2022	Initial revision

2. Overview

This document presents our findings in reviewed contracts.

2.1 Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specialized in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run a free certification course Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, Rockaway Blockchain Fund.

2.2 Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Slither is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices. The code architecture is reviewed.
4. **Local deployment + hacking** - contracts are deployed locally and we try to attack the system and break it.
5. **Unit testing** - run unit tests to ensure that the system works as expected. Potentially we write our own unit tests for specific suspicious scenarios.

2.3 Review team

Member's Name	Position
Štěpán Šonský	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4 Disclaimer

We have put our best effort to find all vulnerabilities in the system. However, our findings should not be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Layer Zero engaged Ackee Blockchain to conduct a security review of LzApp and Tokens with a total time donation of 5 engineering days in a period between 27. April - 3. May 2022.

The scope included the following repository with a given commit:

- https://github.com/LayerZero-Labs/solidity-examples
  - contracts/lzApp
  - contracts/tokens

- 87941ce6160f27a4057372e78c552c780baae524

We began our review by using static analysis tools and then took a deep dive into the logic of the contracts. During the review, we paid special attention to:

- checking if nobody can exploit the system,
- ensuring access controls are not too weak,
- checking the architecture,
- checking the code quality and Solidity best practices,
- and looking for common issues such as data validation.

The code quality is solid. However, we've identified a few code duplications so the inheritance could be better designed. Also, the unit test coverage is insufficient in some audited contracts. Although the code is simple, understandable, and contains few comments, it is a good practice to cover the code using NatSpec documentation.

During our in-depth code review, we found a serious issue H1 in the contract logic, which leads to the system's misbehavior. Another high severity issue H2 with a low likelihood can be misused by developers. Also, we have identified a few minor best practices violations.

Ackee Blockchain recommends LayerZero to:

- avoid code duplications,
- increase unit test coverage,
- use NatSpec documentation.

4. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

4.1 Contracts

Contracts we find important for better understanding are described in the following section.

LzApp
LzApp is a generic implementation of `ILayerZeroReceiver`, which interacts with LZ Endpoint and performs `lzReceive()` and `lzSend()` with basic input checks. Also holds `trustedRemoteLookup` map, where trusted sources are stored. The contract also uses OpenZeppelin's `Ownable` pattern. The ownership can be transferred or renounced.

NonblockingLzApp
The default `LzApp` implementation is blocking, so failed messages block the channel. `NonblockingLzApp` wraps the receive logic into the try-catch block and failed messages are stored for the future retry.

BasedOFT
`BasedOFT` implements `OFT` and overrides just `_debitFrom()`, `_creditTo()` and `getType()` functions. This contract is not used in other audited contracts.

PausableOFT
`Pausable` implementation of the `OFT` contract. The owner can pause the contract, so block the cross-chain transactions. This contract is not used in other audited contracts.

ProxyOFT
`ProxyOFT` wraps the existing ERC-20 token and adds `NonBlockingLzApp` cross-chain logic.

OFT
`OFT` extends from `NonBlockingLzApp` and OpenZeppelin `ERC20` implementation, which means the token itself has cross-chain abilities.

ProxyONFT721
`ProxyONFT721` implements `NonBlockingLzApp` and wraps existing ERC-721 non-fungible token and adds LZ cross-chain abilities.

ProxyONFT1155
`ProxyONFT1155` implements `NonBlockingLzApp`, wraps existing ERC-1155 multi-token and adds LZ cross-chain logic.

UniversalONFT721
`ONFT721` implementation with auto-incrementing mint ID and `maxMintId` boundary.

ONFT721
LZ cross-chain NFT, extends from `NonBlockingLzApp` and OpenZeppelin `ERC721` implementation.

ONFT1155
Extends from `NonBlockingLzApp` and OpenZeppelin `ERC1155` multi-token implementation, which means tokens have cross-chain abilities.

4.2 Actors

Owner
The owner has special privileges in the `LzApp` contract, which allows him to `setTrustedRemote()` for other chains. And also to execute these specific functions on the LayerZero Endpoint:

- `setConfig()`
- `setSendVersion()`
- `setReceiveVersion()`
- `forceResumeReceive()`

In the `PausableOFT` contract, the owner can pause cross-chain transactions.

User
User means any external address in the network which can interact with the protocol and call unprotected public/external functions. The user can execute very similar operations as on any other ERC-20 / ERC-721 / ERC-1155 contract. Also the user can call `retryMessage()` on `NonBlockingLzApp`.

4.3 Trust model

In the case of `LzApp`, users have to trust the owner in terms of setting the correct configuration. The owner is potentially able to cause a denial of service by setting the wrong config and versions.

In the `PausableOFT` the owner has the ability to pause cross-chain transactions for all users of that specific token. We do not recommend using this implementation because the owner could potentially abuse this feature.

Generally, users have to trust the owner in all `LzApp` descendants. Also, users have to trust the LayerZero protocol itself, including the owner.

5. Vulnerabilities risk methodology

Each finding contains an *Impact* and *Likelihood* ratings.

If we have found a scenario in which the issue is exploitable, it will be assigned an impact of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we have not found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Informational*.

Low to *High* impact issues also have a *Likelihood* that measures the probability of exploitability during runtime.

5.1 Finding classification

The complete definitions are as follows:

Impact

High
Conditions that activate the issue will lead to undefined or catastrophic consequences for the system.

Medium
Conditions that activate the issue will result in consequences of serious substance.

Low
Conditions that activate the issue will have outcomes on the system that are either recoverable or do not jeopardize its regular functioning.

Warning
The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) but could be a security vulnerability if these were to change slightly. If we have not found a way to exploit the issue given the time constraints, it might be marked as "Warning" or higher, based on our best estimate of whether it is currently exploitable.

Informational
The issue is on the borderline between code quality and security. Examples

include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

High
The issue is exploitable by virtually anyone under virtually any circumstance.

Medium
Exploiting the issue currently requires non-trivial preconditions.

Low
Exploiting the issue requires strict preconditions.

6. Findings

This section contains the list of discovered findings. Unless overridden for purposes of readability, each finding contains:

- a *Description*,
- an *Exploit scenario*, and
- a *Recommendation*

Many times, there might be multiple ways to solve or alleviate the issue, with varying requirements in terms of the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others. Issues can be also acknowledged by developers as not a risk.

Summary of Findings

ID		Type	Impact	Likelihood	Status
H1	Burn address issue	Syntax	High	High	Reported
H2	Condition bypass	Inheritance	High	Low	Reported
W1	Low test coverage	Tests	N/A	N/A	Reported
W2	Code duplication	Best practices	N/A	N/A	Reported
W3	ERC721, ERC1155 reentrancy	Reentrancy	N/A	N/A	Reported
W4	Unresolved TODO	Best practices	N/A	N/A	Reported
W5	Unintended feature - Renounce ownership	Access controls	N/A	N/A	Reported
I1	Public functions can be external	Gas saving	N/A	N/A	Reported
I2	Missing require message	Best practices	N/A	N/A	Reported
I3	Missing zero lenght handling	Data validation	N/A	N/A	Reported
I4	Missing documentation	Documentation	N/A	N/A	Reported
I5	Hardcoded types	Best practices	N/A	N/A	Reported

H1 - Burn address issue

Impact:	High	**Likelihood:**	High
Target:	ONFT721, ONFT1155, ProxyOFT, ProxyONFT721, ProxyONFT1155	**Type:**	Syntax

Description
The syntactic bug, which is present in `_nonblockingLzReceive()` function in contracts `ONFT721`, `ONFT1155`, `ProxyOFT`, `ProxyONFT721` and `ProxyONFT1155`.

```
if (toAddress == address(0x0)) toAddress == address(0xdEaD);
```

There is a comparison operator instead of assignment, so `toAddress` is not being assigned to `address(0xdEaD)` in case that `address(0x0)` is in the `_payload`, so `toAddress` stays `address(0x0)`.

Then `address(0x0)` is passed into the `_afterReceive()` function which calls `_mint()` or `_safeTransfer()`with `address(0x0)` in parameter `to`.

This leads to transaction revert in `_mint()` ("mint to the zero address") or `_safeTransfer()` ("transfer to the zero address") and then stacking `failedMessages` in the parent contract (`NonblockingLzApp`) without being able to successful retry.

Exploiting scenario
This bug in the logic affects all incoming messages which contain `address(0x0)` in the payload.

Recommendation
Use the assignment operator instead of comparison.

```
if (toAddress == address(0x0)) toAddress = address(0xdEaD);
```

H2 - Condition bypass

Impact:	High	**Likelihood:**	Low
Target:	LzApp, NonBlockingLzApp	**Type:**	Inheritance

Description
Conditions in virtual functions can be overridden and bypassed. Example contracts should not give developers such an ability. The affected functions and conditions are:

`LzApp.lzReceive()`:
```
require(_msgSender() == address(lzEndpoint));
```

`NonBlockingLzApp.nonBlockingLzReceive()`:
```
require(_msgSender() == address(this), "LzReceiver: caller must be LzApp");
```

Exploiting scenario
Developers of descendant proxy tokens can override these functions to bypass conditions in receive functions, which could give them ability to drain tokens from the contract.

Recommendation
Remove the `virtual` keyword from `LzApp.lzReceive()` and `NonBlockingLzApp.nonBlockingLzReceive()` to disallow function override.

W1 - Low test coverage

Impact:	Warning	**Likelihood:**	N/A
Target:	/**/*	**Type:**	Tests

Description
Audited contracts has very low test coverage. In some of them test are completely missing (`PausableOFT`, `ProxyOFT`, `ONFT721`, `ONFT1155`). See Appendix A for the test results.

Recommendation
Add more testing scenarios and increase the test coverage. Unit testing is an essential bug prevention and should not be underestimated.

W2 - Code duplication

Impact:	Warning	**Likelihood:**	N/A
Target:	/**/*	**Type:**	Architecture

Description
Contracts contains a lot of duplicated code which can be the source of many bugs. Also, it decreases the readability of the code and effective long-term maintainability. E.g. functions `estimateSendFee()`, `send()`, `_nonBlockingReceive()`...

Recommendation
Review the architecture and refactor contracts using inheritance or libraries to remove code duplications.

W3 - ERC721, ERC1155 reentrancy

Impact:	Warning	**Likelihood:**	N/A
Target:	ONFT721, ONFT1155	**Type:**	Reentrancy

Description

`ERC721` and `ERC1155` implementation can be subject to reentrancy attack using `IERC721Receiver` and `IERC1155Receiver`. We have not found any exploitable scenario in current contracts, so we marked this hypothetical issue as a warning.

Recommendation

Double check your ERC721 and ERC1155 implementations to whether they are not vulnerable to this kind of attack. Eventually, add the `ReentrancyGuard` to public/external functions.

W4 - Unresolved TODO

Impact:	Warning	**Likelihood:**	N/A
Target:	ProxyONFT721	**Type:**	Best practices

Description

We found unresolved TODO in the code of ProxyONFT721.

```
// TODO: to send cross chain tx
```

Recommendation

Resolve the TODO and remove the comment.

W5 - Unintended feature - Renounce ownership

Impact:	Warning	**Likelihood:**	N/A
Target:	*.sol implements Ownable	**Type:**	Access control

Description
The OpenZeppelin's Ownable pattern contains `renounceOwnership()` which sets the owner address to `address(0)`. This could lead to irreversible damage to the contract.

Exploiting scenario
This is not a directly exploitable issue, but can be considered as an unintended feature of the system. This function can be called accidentally or intentionally by a malicious owner.

Recommendation
We recommend using a multisig wallet for the owner to avoid accidental `renounceOwnership()` call. Or it can be handled by overriding the `renounceOwnership()` function in contracts inherited from `Ownable`.

I1 - Public functions can be external

Impact:	Informational	**Likelihood:**	N/A
Target:		**Type:**	Best practices

Description
Some of the contracts' functions are declared as public, but not called from the contract itself or from descendants.

Recommendation
Review functions' visibility and change it to external in cases where it is possible.

I2 - Missing require message

Impact:	Informational	**Likelihood:**	N/A
Target:	LzApp	**Type:**	Best practices

Description

Missing revert message in the `require` statement.

```
require(_msgSender() == address(lzEndpoint));
```

Recommendation

Adding the revert message into the `require` statement is generally good practice to achieve human-readable errors.

I3 - Missing zero length handling

Impact:	Informational	**Likelihood:**	N/A
Target:	ONFT1155, ProxyONFT1155	**Type:**	Data validation

Description
The following code is missing the code path in case `tokenIds.lenght == 0`. This does not have to be an issue, but maybe require/revert can be used here.

```
if (tokenIds.length == 1) {
            _afterReceive(_srcChainId, localToAddress, tokenIds[0],
amounts[0]);
            emit ReceiveFromChain(_srcChainId, localToAddress, tokenIds[0],
amounts[0], _nonce);
        } else if (tokenIds.length > 1) {
            _afterReceiveBatch(_srcChainId, localToAddress, tokenIds,
amounts);
            emit ReceiveBatchFromChain(_srcChainId, localToAddress, tokenIds,
amounts, _nonce);
        }
```

Recommendation
Review the code whether the require/revert can be more suitable in this case.

I4 - Missing documentation

Impact:	Informational	**Likelihood:**	N/A
Target:		**Type:**	Documentation

Description
The code is missing detailed documentation.

Recommendation
Although the code is relatively simple and understandable, we recommend to use NatSpec documentation. High quality documentation has to be essential part of any professional project.

I5 - Hardcoded types

Impact:	Informational	**Likelihood:**	N/A
Target:	OFT, BasedOFT	**Type:**	Best practices

Description
Token types are hardcoded integers (0, 1), which is confusing and not self-explainable. Hardcoded values decrease the code readability and also could lead to bugs.

```
function getType() public view virtual override returns (uint) {
    return 0;
}
```

Recommendation
Use well-named constants instead of hardcoded values.

7. Appendix A - Unit test results

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
examples/	100	62.5	70	100	
ExampleBasedOFT.sol	100	100	0	100	
ExampleOFT.sol	100	100	0	100	
ExampleUniversalONFT721.sol	100	100	0	100	
OmniCounter.sol	100	100	100	100	
PingPong.sol	100	62.5	100	100	
interfaces/	100	100	100	100	
ILayerZeroEndpoint.sol	100	100	100	100	
ILayerZeroReceiver.sol	100	100	100	100	
ILayerZeroUserApplicationConfig.sol	100	100	100	100	
lzApp/	66.67	33.33	64.29	66.67	
LzApp.sol	76.47	50	60	76.47	46,51,55,59
NonblockingLzApp.sol	50	16.67	75	50	42,43,44,46,48
mocks/	85.37	70	54.29	85.88	
ERC1155Mock.sol	66.67	100	75	66.67	20
ERC721Mock.sol	0	100	0	0	14,18,22
LZEndpointMock.sol	89.47	70	59.26	89.87	... 299,303,307
token/oft/	69.57	75	60	72.73	
IOFT.sol	100	100	100	100	
IOFTCore.sol	100	100	100	100	
OFT.sol	69.57	75	60	72.73	... 29,30,38,73
token/oft/extension/	8.7	0	20	8.7	
BasedOFT.sol	66.67	100	75	66.67	14
PausableOFT.sol	0	100	0	0	18,22
ProxyOFT.sol	0	0	0	0	... 8,89,98,106
token/onft/	77.61	55.56	77.78	80.3	
IONFT1155.sol	100	100	100	100	
IONFT721.sol	100	100	100	100	
ONFT1155.sol	76.09	58.33	76.47	77.78	... 42,43,47,51
ONFT721.sol	80.95	50	80	85.71	17,18,22
token/onft/extension/	69.01	58.33	63.33	70.42	
ProxyONFT1155.sol	95.65	62.5	94.44	97.78	171
ProxyONFT721.sol	0	0	0	0	... 55,64,85,90
UniversalONFT721.sol	100	100	100	100	
All files	72.35	56.58	59.57	73.8	

```
BasedOFT:
  ✓ send() - tokens from main to other chain

OFT:
  setting up stored payload
    ✓ hasStoredPayload() - stores the payload
    ✓ getLengthOfQueue() - cant send another msg if payload is blocked
    ✓ retryPayload() - delivers a stuck msg
    ✓ forceResumeReceive() - removes msg
    ✓ forceResumeReceive() - removes msg, delivers all msgs in the queue
    ✓ forceResumeReceive() - emptied queue is actually emptied and doesnt get double counted
    - forceResumeReceive() - the queue being emptied is done in the correct fifo order

PausableOFT:
  - todo

ProxyOFT:
  - todo

ONFT721:
  - todo

ProxyONFT1155:
  ✓ send()
  ✓ sendBatch()
  ✓ estimateSendFee()
  ✓ estimateSendBatchFee()

ProxyONFT721:
  - send()

UniversalONFT721:
  ✓ send() - mint on the source chain and send ONFT to the destination chain
```

ackee blockchain

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq