ackee blockchain

Layer Zero

Solidity Examples NativeProxy

by Ackee Blockchain

August 8, 2022

Contents

1. Document Revisions . . . 4
2. Overview . . . 5
2.1. Ackee Blockchain . . . 5
2.2. Audit Methodology . . . 5
2.3. Review team . . . 6
2.4. Disclaimer . . . 6
3. Executive Summary . . . 7
4. System Overview . . . 8
4.1. Contracts . . . 8
4.2. Actors . . . 9
4.3. Trust model . . . 9
5. Vulnerabilities risk methodology . . . 11
5.1. Finding classification . . . 11
6. Findings . . . 13
M1: Accepting messages from untrusted remotes . . . 15
M2: Constructor data validation . . . 17
L1: Ownable pattern . . . 18
L2: Missing override for ERC165 . . . 20
W1: Usage of `solc` optimizer . . . 21
W2: Recent `solc` version . . . 22
W3: Empty `_srcAddress` can bypass trusted remote check . . . 23
W4: Unused `_lzSend()` function . . . 25
I1: Coding practice . . . 26
I2: Zero token transfer . . . 27
I3: Public functions . . . 28
I4: Unused SafeERC20 . . . 29

Appendix A: How to cite 30
Appendix B: Glossary of terms 31

1. Document Revisions

1.0	Final report	Aug 8, 2022

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 and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzzy testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests.

2.3. Review team

Member's Name	Position
Miroslav Škrabal	Lead Auditor
Jan Šmolík	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.4. Disclaimer

We've put our best effort to find all vulnerabilities in the system, however our findings shouldn't 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

The solidity-examples repository is a collection of LayerZero's example contracts for developers, containing various cross-chain token implementations.

LayerZero engaged Ackee Blockchain to perform a security review of a new example cross-chain token, NativeProxyOFT20.sol, later renamed to `NativeOFT.sol`, with a total time donation of 5 engineering days in a period between 3.8.2022 and 8.8.2022 and the lead auditor was Miroslav Škrabal.

The commit we worked on is the following:

- `b0bd359e8affb782da83915fe06be8b3a7cc34c7`

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

- detecting possible reentrancies in the code;
- ensuring the proper handling of the tokens during the cross-chain messages;
- ensuring access controls are not too relaxed or too strict;
- looking for common issues such as data validation.

The review resulted in 12 findings, ranging from Informational to Medium severity.

Ackee Blockchain recommends LayerZero to address all the reported issues.

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.sol

LzApp is an example base contract for Layer Zero protocol applications. These applications send and receive cross-chain messages through the `lzEndpoint` contract.

NonblockingLzApp.sol

NonblockingLzApp inherits from `LzApp` and defines and implements three additional functions for the handling of the received messages.

NativeProxyOFT20.sol

NativeProxyOFT20 is Layer Zero's example implementation of a cross-chain wrapped native token. On the source chain, where NativeProxyOFT20 is deployed, users can:

- deposit the native tokens and receive NativeProxyOFT20 at a 1:1 ratio;
- withdraw the native tokens back for NativeProxyOFT20 at a 1:1 ratio;
- send NativeProxyOFT20 tokens to a different Layer Zero supported chain.

4.2. Actors

This part describes actors of the system, their roles, and permissions.

Owner of the `NativeProxyOFT20` contract

The owner of the NativeProxyOFT20 contract can call the following functions which change user application config in `LzApp` (since `LzApp` is `NativeProxyOFT20's` parent class):

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

User

A user can:

- deposit and withdraw `NativeProxyOFT20`;
- transfer `NativeProxyOFT20` cross-chain through the Layer Zero protocol.

4.3. Trust model

The owner has several privileges that can influence the usage of the protocol in important ways. Firstly, he can manipulate the trusted remotes mapping. As a result, he potentially can add a malicious trusted remote that would allow him to transfer funds away from the proxy. Another problem with the trusted remotes mapping is that it can be manipulated dynamically. This allows the owner to change the value of the trusted remotes to a new value. As a result,

he can, for example, front-run users' send transactions with his change remote transaction and route the message to a different chain.

5. Vulnerabilities risk methodology

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't 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 *Info*.

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

5.1. Finding classification

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't 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 haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - 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 overriden 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, making clear which solve the underlying issue better (albeit possibly only with architectural changes) than others.

Summary of Findings

	Severity	Impact	Likelihood
M1: Accepting messages from untrusted remotes	Medium	Medium	Low
M2: Constructor data validation	Medium	Medium	Low
L1: Ownable pattern	Low	Low	Low
L2: Missing override for ERC165	Low	Low	Low
W1: Usage of `solc` optimizer	Warning	Warning	N/A
W2: Recent `solc` version	Warning	Warning	N/A

	Severity	Impact	Likelihood
W3: Empty `_srcAddress` can bypass trusted remote check	Warning	Warning	N/A
W4: Unused `_lzSend()` function	Warning	Warning	N/A
I1: Coding practice	Info	Info	N/A
I2: Zero token transfer	Info	Info	N/A
I3: Public functions	Info	Info	N/A
I4: Unused SafeERC20	Info	Info	N/A

Table 2. Table of Findings

M1: Accepting messages from untrusted remotes

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	LzApp.sol, NonblockingLzApp.sol	Type:	Data validation

Description

The LzApp contract uses `trustedRemoteLookup` mapping to validate whether a given remote can be trusted or not. The values to the mapping are set by the contract owner by the function `setTrustedRemote`. If special conditions are met, a call from an untrusted remote can be executed. The `retryMessage` function enables this as it performs no check on whether the given remote is trusted or not. Additionally, it is enabled by the fact that a given remote can be set as untrusted after it was set as trusted.

Vulnerability scenario

1. Owner sets a new trusted remote A using `setTrustedRemote` function.
2. A gets compromised by a malicious actor.
3. A message from A is delivered, but the execution fails inside the `_blockingLzReceive` and the message is thus stored to `failedMessages` mapping.
4. Because A was compromised, the owner sets A as untrusted. He does so by setting the value for key A to the default value.
5. The failed message gets reexecuted by calling `retryMessage`, and this time the call succeeds. As a result, a message from an *untrusted* remote gets executed.

Recommendation

Inside the `retryMessage` function, implement an additional check whether or not a given remote is trusted. This check is needed because a remote can become untrusted while a message is stored in the `failedMessages` buffer.

Go back to Findings Summary

M2: Constructor data validation

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	NativeProxyOFT20.sol	Type:	Data validation

Description

The NativeProxyOFT20 contract does not perform any data validation in its constructor. At least, the `lzEndpoint` argument should be validated.

Exploit scenario

An incorrect value of `_lzEndpoint` is passed to the constructor. Instead of reverting, the call succeeds. If such a mistake is not discovered quickly and the contracts are not redeployed, the protocol can behave in an undefined way.

Recommendation

Add more stringent data validation for `_lzEndpoint`. At the very least, this would include a zero-address check. Ideally, it is recommended to define a getter in the `LzEndpoint` such as `contractId()` that would return a hash of an identifier unique to the (project, contract). An example would be `keccak256("Layer Zero Endpoint")`. Upon constructing the `NativeProxy` the identifier would be retrieved from the passed-in `lzEndpoint` address and compared to the expected value.

Go back to Findings Summary

L1: Ownable pattern

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	NativeProxyOFT20.sol, Ownable.sol	Type:	Access controls

Description

The NativeProxyOFT20 is an `Ownable` contract, and the owner of the contract can call the functions which change the user application config of `LzApp`. The ownership can be transferred and renounced by the owner.

The owner can call `renounceOwnership()` by mistake, and there is no robust verification mechanism for the new proposed owner in `transferOwnership()`.

Vulnerability scenario

The owner accidentally calls `renounceOwnership()` or `transferOwnership()` with a wrong address.

Recommendation

Firstly, we recommend overriding the `renounceOwnership()` method to disable it, if this feature has no intended usage.

With respect to `transferOwnership()`, one of the common and safer approaches to ownership transfer is to use a two-step transfer process.

Suppose Alice wants to transfer the ownership to Bob. The two-step process would have the following steps: Alice proposes a new owner, namely Bob. This proposal is saved to a variable candidate. Bob, the candidate, calls the acceptOwnership function. The function verifies that the caller is the new

proposed candidate, and if the verification passes, the function sets the caller as the new owner. If Alice proposes a wrong candidate, she can change it. However, it can happen, though with a very low probability that the wrong candidate is malicious (most often it would be a dead address). An authentication mechanism can be employed to prevent the malicious candidate from accepting the ownership.

Go back to Findings Summary

L2: Missing override for ERC165

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	NativeProxyOFT20.sol	Type:	Missing implementation

Description

The NativeProxyOFT20 contract inherits from the `ERC165` contract. The `ERC165` contract defines the `supportsInterface` function, which can be used to check what interfaces the given contract implements. However, the `NativeProxyOFT20` contract does not override the mentioned function, although it implements some additional interfaces like the ERC20 interface.

As a result, the NativeProxy contract can return a wrong result when the `supportsInterface` function gets called.

Recommendation

Override the `supportInterface` function and add the equality check for the `IERC20` interface. In the long term, ensure to validate all the relationships to parent contracts, as a similar mistake also manifested in W4: Unused `_lzSend()` function.

Go back to Findings Summary

W1: Usage of `solc` optimizer

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Compiler configuration

Description

The project uses `solc` optimizer. Enabling `solc` optimizer may lead to unexpected bugs.

The Solidity compiler was audited in November 2018, and the audit concluded that the optimizer may not be safe.

One recent bug was discovered in June 2022. The bug caused some assembly memory operations to be removed, even though the values were used later in the execution.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the optimizer. As a result, it is possible to attack the protocol.

Recommendation

Until the `solc` optimizer undergoes more stringent security analysis, opt-out using it. This will ensure the protocol is resilient to any existing bugs in the optimizer.

Go back to Findings Summary

W2: Recent `solc` version

Impact:	Warning	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Compiler configuration

Description

The project uses a too recent compiler version, `0.8.15`.

Vulnerability scenario

A few months after deployment, a vulnerability is discovered in the compiler. As a result, it is possible to attack the protocol.

Recommendation

A good practice is to use a compiler version established for 3-6 months.

Go back to Findings Summary

W3: Empty `_srcAddress` can bypass trusted remote check

Impact:	Warning	Likelihood:	N/A
Target:	LzApp.sol	Type:	Data validation

Listing 1. Excerpt from LzApp.sol.lzReceive

```
25    function lzReceive(uint16 _srcChainId, bytes memory _srcAddress,
   uint64 _nonce, bytes memory _payload) public virtual override {
26        // lzReceive must be called by the endpoint for security
27        require(_msgSender() == address(lzEndpoint), "LzApp: invalid
   endpoint caller");
28
29        bytes memory trustedRemote = trustedRemoteLookup[_srcChainId];
30        // if will still block the message pathway from (srcChainId,
   srcAddress). should not receive message from untrusted remote.
31        require(_srcAddress.length == trustedRemote.length && keccak256
   (_srcAddress) == keccak256(trustedRemote), "LzApp: invalid source
   sending contract");
```

Description

Inside the LzApp in the `lzReceive` function, there is a check of the remote (see Listing 1). However, the `require` inside the receive function does not check whether the argument `_srcAddress` has a default bytes value. If it did and the value returned from the `truestedRemoteLookup` was also the default bytes value, then the `require` would incorrectly pass.

Additionally, calling the `isTrustedRemote` with the default value for the `_srcAddress` argument will incorrectly yield `true`.

Vulnerability scenario

Suppose that it would be possible that the `_srcAddress` argument with the default bytes value was passed to the `lzReceive` function by the `lzEndpoint`. If

a malicious payload was passed (non-zero amount and the attacker's address) to the receive function, it would be possible to drain the `NativeProxyOFT20` funds. That is because at the end of the receive call chain, the function `_creditTo`, which performs the transfer of value, is called.

Recommendation

Implement more careful validation of the `_srAddress`. More specifically, ensure that calling the mentioned functions with the default bytes value will result in a revert.

Go back to Findings Summary

W4: Unused `_lzSend()` function

Impact:	Warning	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Code duplication

Description

The NativeProxyOFT20 defines a `_send()` function.

The LzApp defines an internal `_lzSend()` function.

`_send()` contains the exact same three lines of code which form the `_lzSend()` function:

```
bytes memory trustedRemote = trustedRemoteLookup[_dstChainId];
require(trustedRemote.length != 0, "destination chain is not a trusted source");
lzEndpoint.send{value: msg.value}(_dstChainId, trustedRemote, _payload, _refundAddress, _zroPaymentAddress, _adapterParams);
```

Recommendation

We recommend replacing those three lines of code with a `_lzSend()` call to avoid the unnecessary code repetition:

```
_lzSend(_dstChainId, payload, _refundAddress, _zroPaymentAddress, _adapterParams);
```

This is the way it is done in the `OFT20Core` contract.

Go back to Findings Summary

I1: Coding practice

Impact:	Info	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Code readability

Description

Inside the NativeProxyOFT20 contract, there are a few things that could improve the code readability and understandability.

- function `_send()` is public, although the underscore indicates that it should be private;
- the project uses `uint` instead of `uint256`. Using `uint256` is more explicit and less missleading;
- the name `NativeProxyOFT20` is semantically incorrect, because there is no proxy at all in the contract.

Recommendation

For better code readability:

- rename `_send()` to `send()`;
- use `uint256` instead of `uint`.

The contract has already been renamed from `NativeProxyOFT20` to `NativeOFT`.

Go back to Findings Summary

I2: Zero token transfer

Impact:	Info	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Code logic

Description

The NativeProxyOFT20 allows for sending zero tokens cross-chain.

This is not a security issue, but on the other hand, it does not make any sense to send zero tokens cross-chain.

Recommendation

Implement logic to prevent zero token cross-chain transfers to `_send()` function. One way to achieve that is to add the following line of code:

```
require(_amount > 0, "NativeOFT: Zero token transfer");
```

Go back to Findings Summary

I3: Public functions

Impact:	Info	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Gas optimization

Description

The following functions are declared `public` even though they are not called internally anywhere:

- `estimateSendFee()`
- `sendFrom()`
- `withdraw()`

Recommendation

If functions are not called internally, they should be declared `external`. It helps gas optimization because function arguments do not have to be copied into memory.

Go back to Findings Summary

I4: Unused SafeERC20

Impact:	Info	Likelihood:	N/A
Target:	NativeProxyOFT20.sol	Type:	Unused library

Description

The NativeProxyOFT20 contract uses the `SafeERC20` library for the `IERC20` interface. However, the library is never used.

Recommendation

Consider whether including the `SafeERC20` library is necessary and optionally remove it.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Layer Zero: Solidity Examples NativeProxy, August 8, 2022.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

ackee

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://discord.gg/z4KDUbuPxq