



# LayerZero Examples

# Audit

---



Presented by:

**OtterSec**

**Robert Chen**

**Shiva Shankar**

**Vishvesh Rao**

[contact@osec.io](mailto:contact@osec.io)

[r@osec.io](mailto:r@osec.io)

[sh1v@osec.io](mailto:sh1v@osec.io)

[vishvesh@osec.io](mailto:vishvesh@osec.io)



# Contents

<b>01 Executive Summary</b>	<b>2</b>
Overview . . . . .	2
Key Findings . . . . .	2
<b>02 Scope</b>	<b>3</b>
<b>03 Findings</b>	<b>4</b>
<b>04 General Findings</b>	<b>5</b>
OS-LZR-SUG-00   Reorganize Payload Fields . . . . .	6
OS-LZR-SUG-01   Fee Constraints Edge Case . . . . .	7
OS-LZR-SUG-02   Gas Limit Ambiguity . . . . .	8
OS-LZR-SUG-03   Redundant Length Check . . . . .	9
OS-LZR-SUG-04   Missing Initialization Check . . . . .	10
OS-LZR-SUG-05   Missing Event Emissions . . . . .	11
OS-LZR-SUG-06   Verify Array Lengths . . . . .	12
 <b>Appendices</b>	
<b>A Vulnerability Rating Scale</b>	<b>13</b>
<b>B Procedure</b>	<b>14</b>

# 01 | **Executive Summary**

## Overview

LayerZero engaged OtterSec to perform an assessment of various LayerZero programs under our retainer. This is an ongoing engagement, starting November 11th. For more information on our auditing methodology, see [Appendix B](#).

As part of this engagement, we reviewed LayerZero's [OFT v2 upgrade](#), [upgradable contract update](#), and [oNFT extensions](#).

## Key Findings

Over the course of this audit engagement, we produced 7 findings total.

In particular, we found a minor edge case with fee constraint validation ([OS-LZR-SUG-01](#)), a missing initialization check ([OS-LZR-SUG-04](#)), along with recommendations around reorganizing payload fields ([OS-LZR-SUG-00](#)), gas limit ambiguity ([OS-LZR-SUG-02](#)), and redundant length checks ([OS-LZR-SUG-03](#)).

Overall, we commend the LayerZero team for being responsive and knowledgeable throughout the audit.

## 02 | Scope

The source code was delivered to us in a git repository at [github.com/LayerZero-Labs/solidity-examples](https://github.com/LayerZero-Labs/solidity-examples). This audit was performed up to and against [09abd22](#).

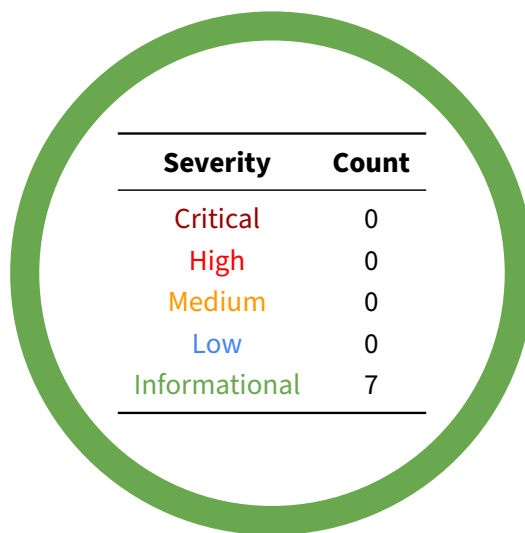
A brief description of the programs is as follows.

Name	Description
solidity-examples	Solidity code examples building on top of LayerZero.

## 03 | Findings

Overall, we report 7 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	0
High	0
Medium	0
Low	0
Informational	7

## 04 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-LZR-SUG-00	Reorder OFT payload fields to mitigate the impact of variable length decoding.
OS-LZR-SUG-01	Potential edge case for fee constraints.
OS-LZR-SUG-02	The gas limit check is ambiguous and can lead to unintended behaviour.
OS-LZR-SUG-03	The length check can be safely removed.
OS-LZR-SUG-04	Failure to check for address initialization in <code>isTrustedRemote</code> may lead to unintended behaviour.
OS-LZR-SUG-05	<code>ONFT721Core.sol</code> and <code>DistributedONFT721.sol</code> fails to emit events for certain state changes.
OS-LZR-SUG-06	The constructor of <code>DistributedONFT721.sol</code> lacks checks for array lengths.

## OS-LZR-SUG-00 | Reorganize Payload Fields

### Description

Currently, the OFT payload is encoded via

```
return abi.encodePacked(
    PT_SEND_AND_CALL,
    _toAddress,
    _amountSD,
    _addressToBytes32(_from),
    _dstGasForCall,
    _payload
);tc
```

SOLIDITY

Note that the security of this encoding depends on the correct size of `_toAddress`. Otherwise, the critical field `_amountSD` may get confused with bytes of the `_from` field.

### Remediation

As a defence-in-depth measure, reorder the fields. More specifically, putting `_amountSD` first in the encoding will likely mitigate future variable length encoding attacks.

## OS-LZR-SUG-01 | Fee Constraints Edge Case

### Description

Within `Fee.sol`, when using `setDefaultFeeBp` or `setFeeBp` to set the `defaultFeeBp`, it is important to ensure that the fee is less than and not equal to `BP_DENOMINATOR`. If `defaultFeeBp` is set equal to `BP_DENOMINATOR`, transactions will fail.

### Remediation

Update the check to ensure that the `_feeBp` is only less than `BP_DENOMINATOR`.

*Fee.sol*

SOLIDITY

```
function setDefaultFeeBp(uint16 _feeBp) public virtual onlyOwner {
    require(_feeBp < BP_DENOMINATOR, "Fee: fee bp must be < BP_DENOMINATOR");
    defaultFeeBp = _feeBp;
    emit SetDefaultFeeBp(defaultFeeBp);
}

function setFeeBp(uint16 _dstChainId, bool _enabled, uint16 _feeBp) public
    ↪ virtual onlyOwner {
    require(_feeBp < BP_DENOMINATOR, "Fee: fee bp must be < BP_DENOMINATOR");
    chainIdToFeeBps[_dstChainId] = FeeConfig(_feeBp, _enabled);
    emit SetFeeBp(_dstChainId, _enabled, _feeBp);
}
```



## OS-LZR-SUG-02 | Gas Limit Ambiguity

### Description

contracts/contracts-upgradable/lzApp/LzAppUpgradeable.sol

SOLIDITY

```
function _checkGasLimit(uint16 _dstChainId, uint16 _type, bytes memory
    ↪ _adapterParams, uint _extraGas) internal view virtual {
    uint providedGasLimit = _getGasLimit(_adapterParams);
    uint minGasLimit = minDstGasLookup[_dstChainId][_type] + _extraGas;
    require(minGasLimit > 0, "LzApp: minGasLimit not set");
    require(providedGasLimit >= minGasLimit, "LzApp: gas limit is too low");
```

The gas limit check has three scenarios.

1. minDstGasLookup[\_dstChainId][\_type] != 0 and \_extraGas == 0
2. minDstGasLookup[\_dstChainId][\_type] == 0 and \_extraGas != 0
3. minDstGasLookup[\_dstChainId][\_type] != 0 and \_extraGas != 0

contracts/contracts-upgradable/lzApp/LzAppUpgradeable.sol

SOLIDITY

```
function setMinDstGas(uint16 _dstChainId, uint16 _packetType, uint _minGas)
    ↪ external onlyOwner {
    require(_minGas > 0, "LzApp: invalid minGas");
    minDstGasLookup[_dstChainId][_packetType] = _minGas;
    emit SetMinDstGas(_dstChainId, _packetType, _minGas);
}
```

After changing the minimum gas, the admin cannot revert it back to zero, rendering scenario 2. unusable. This leads to unintended double gas payments.

### Remediation

SOLIDITY

```
require(minDstGasLookup[_dstChainId][_type] > 0)
```

If the intention is to disallow minDstGasLookup[\_dstChainId][\_type] from zero, then modify the gas limit check to reflect this requirement. Otherwise, the admin must be able to change the minimum gas to zero.

## OS-LZR-SUG-03 | Redundant Length Check

contracts/contracts-upgradable/lzApp/LzAppUpgradeable.sol

SOLIDITY

```
function getTrustedRemoteAddress(uint16 _remoteChainId) external view returns
↳ (bytes memory) {
    bytes memory path = trustedRemoteLookup[_remoteChainId];
    require(path.length != 0, "LzApp: no trusted path record");
    return path.slice(0, path.length - 20); // the last 20 bytes should be
↳ address(this)
```

If `path.length` is zero, subtracting 20 from `path.length` will result in an arithmetic underflow, causing the program to revert. Therefore, removing the redundant length check is possible if returning an error message is not needed.

### Remediation

Remove the redundant length check.

## OS-LZR-SUG-04 | Missing Initialization Check

### Description

*contracts/contracts-upgradable/lzApp/LzAppUpgradeable.sol*

SOLIDITY

```
function isTrustedRemote(uint16 _srcChainId, bytes calldata _srcAddress)
↳ external view returns (bool) {
    bytes memory trustedSource = trustedRemoteLookup[_srcChainId];
    return keccak256(trustedSource) == keccak256(_srcAddress);
}
```

The function fails to verify whether `srcAddress_` is initialized. Consequently, a function call with a currently uninitialized source chain ID and an empty source address would return true, contrary to expectations.

### Remediation

Add an initialization check to `isTrustedRemote`.

## OS-LZR-SUG-05 | Missing Event Emissions

### Description

No event is emitted in the following setter functions in `ONFT721Core.sol` and `DistributedONFT721.sol`, resulting in state changes.

- `setMinGasToTransferAndStore(...)`
- `setDstChainIdToTransferGas(...)`
- `setDstChainIdToBatchLimit(...)`

### Remediation

Ensure all setter functions which modify state emit events.

### Patch

Resolved in commit [c3f8c7d](#).

## OS-LZR-SUG-06 | Verify Array Lengths

### Description

The `DistributedONFT721.sol` constructor fails to check the length of `_indexArray` and `_valueArray`, which are passed in as arguments to the constructor.

*DistributedONFT721.sol*

SOLIDITY

```
    constructor(string memory _name, string memory _symbol, uint256
↳   _minGasToTransfer, address _layerZeroEndpoint, uint[] memory _indexArray,
↳   uint[] memory _valueArray) ONFT721(_name, _symbol, _minGasToTransfer,
↳   _layerZeroEndpoint){
        uint _indexArrayLength = _indexArray.length;
        for(uint i; i < _indexArrayLength;) {
            tokenIds[_indexArray[i]] = _valueArray[i];
            unchecked{++i;}
        }
    }
```

### Remediation

Verify that `_indexArray` and `_valueArray` are of the same length.

### Patch

Resolved in commit [c3f8c7d](#).

# A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the [General Findings](#) section.

---

<b>Critical</b>	<p>Vulnerabilities that immediately lead to loss of user funds with minimal preconditions</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Misconfigured authority or access control validation</li><li>• Improperly designed economic incentives leading to loss of funds</li></ul>
<b>High</b>	<p>Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Loss of funds requiring specific victim interactions</li><li>• Exploitation involving high capital requirement with respect to payout</li></ul>
<b>Medium</b>	<p>Vulnerabilities that could lead to denial of service scenarios or degraded usability.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Malicious input that causes computational limit exhaustion</li><li>• Forced exceptions in normal user flow</li></ul>
<b>Low</b>	<p>Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Oracle manipulation with large capital requirements and multiple transactions</li></ul>
<b>Informational</b>	<p>Best practices to mitigate future security risks. These are classified as general findings.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>• Explicit assertion of critical internal invariants</li><li>• Improved input validation</li></ul>

---

## B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.