LUDO

SMART CONTRACT AUDIT REPORT

Certification

LUDO TOKEN FOR EVM (ETHEREUM and BASE)

xaudits

SMART CONTRACT AUDITS

Oct 24th, 2025 / v.0.1

Audited source code version:

45fa402f1870602225843ed16d36a8ad08378fb1

xAudits.io – SC Audit Report - Table of contents

1. Executive summary

Ludo (https://www.ludo.com/) is a cross-chain reputation and trust layer transforming Web3 activity into real rewards and actionable discovery.

The Ludo token smart contract inherits LayerZero's **OFT** (an ERC-20 with cross-chain send/receive) and OpenZeppelin **Ownable**. The constructor wires the LayerZero **Endpoint V2** and the **delegate/owner** and mints 600,000,000 LUDO tokens to a set of company wallets on Ethereum Mainnet.

Key properties:

- **No post-deployment mint function is exposed by this contract.** There's only a one-time `_mint` inside the constructor, which cannot be called again, ensuring a fixed supply.
- **No custom admin functions are added.** However, **admin/config functions are inherited** from OFT/OApp (e.g., `setPeer`, `setDelegate`, config setters) and are gated by onlyOwner. These must be treated as privileged operations.
- The contract is immutable (non-upgradeable) with mint function disabled, so the rules are locked for all holders.
- Uses well-known, audited libraries: LayerZero OFT and OpenZeppelin Ownable/ERC-20 implementation.

External deps.

- @openzeppelin/contracts/access/Ownable.sol (industry-standard ownership)
- @layerzerolabs/oft-evm/contracts/OFT.sol (LayerZero V2 Omnichain Fungible Token)

2. Methodology

The audit process includes both automated and manual analysis techniques and testing, consisting of:
- Static and dynamic code analysis
- Business logic review
- Testing for known vulnerability patterns
- Review against known attack vectors
- Verification of gas optimization and efficiency
- Manual and Stress Testing of different integration layers per request
- Corner case scenarios

3. Findings - Structure and Organization

Some sections are more important than others. The most critical areas are at the top, and the less critical sections are at the bottom. The issues in these sections have been fixed or addressed and will show by the "Resolved" or "Unresolved" tags. Each case is written so you can understand how serious it is, with an explanation of whether it is a risk of exploitation or unexpected behavior.

These issues can have a dangerous effect on the ability of the contract to work correctly.

These issues significantly affect the ability of the contract to work correctly.

These issues affect the ability of the contract to operate correctly but do not hinder its behavior.

These issues have a minimal impact on the contract's ability to operate.

These issues do not impact the contract's ability to operate.

Detailed Findings (Issues)

I1. Privileged operations inherited from OFT/OApp require governance procedures

Severity: INFORMATIONAL

Description:
Although this contract adds no new admin functions, OFT/OApp exposes onlyOwner ops such as setPeer, setDelegate, enforced options, library/DVN configs, etc. These are **critical cross-chain configuration functions** that determine:

- which contracts are trusted across chains
- which messaging libraries are used
- and what enforced options/rate limits exist

Misconfiguration (or being compromised) can:

- disable bridging between chains
- route messages to a wrong peer
- or disrupt token transfers permanently

Possible fix to research:
Assign Owner and Delegate to governance-controlled Safe Multisig.
Deploy with _delegate set to a Gnosis Safe (or simpler just transfer ownership immediately after).

Response:
To be using a Multisig.

Status:
Accepted and Closed.

I2. Improve readability with enum-based recipient getter

Severity: INFORMATIONAL

Description:
Currently, allocation recipients are stored in a fixed-length array (`address[6] public allocationRecipients`). While this allows index-based access (e.g., `allocationRecipients(0)`), it requires knowing the numeric index for each enum value. This reduces readability and makes it harder for dashboards or external integrations to display data clearly.

Possible fix to research:

```
function allocationRecipient(Allocation a) external view returns (address) {
    return allocationRecipients[uint256(a)];
}
```

Response:
NA.

Status:
Not addressed.

I3. Duplicate Allocation Recipients Not Prevented

Severity: INFORMATIONAL

Description:
The constructor allows setting multiple allocation categories (e.g., Team, Treasury) to the same address. If the same wallet is accidentally used for multiple allocations, it will receive the combined token amount. While this may be intentional in some cases, it is often a configuration error and should be prevented at deployment.

Possible fix to research:
Add a constructor-time validation that ensures all recipient addresses are unique. This prevents accidental double allocation due to human error in deployment scripts.

Response:
NA.

Status:
Not addressed.

I4. Gas Optimization: Use `unchecked { ++i; }` in Bounded Loops

Severity: INFORMATIONAL

Description:
Solidity 0.8+ automatically checks for integer overflows, adding a small gas cost per arithmetic operation. In fixed-length loops (e.g., iterating from `i = 0` to `< 6`), overflow cannot occur, so these checks are redundant.

Possible fix to research:
Wrap the loop increment in an unchecked block to skip overflow verification. This slightly reduces gas usage without impacting safety.

```
for (uint256 i = 0; i < recipients.length; ) {
    // logic...
    unchecked { ++i; }
}
```

Response:
NA.

Status:
Not addressed.

4. Recommendations

No critical issues were found, Ludo engineering team has demonstrated a solid understanding of web3 technicalities so far, most of their high critical business logic being in web2 backend.
Based on the audit findings, the following recommendations are made for the long-term project's overall security posture:
- Implement fixes for all critical and high-severity issues before deployment (if any)
- Conduct continuous security monitoring post-launch
- Perform periodic re-audits after major updates or contract changes

5. Disclaimer

This audit report is provided strictly for informational purposes.

The authors make no warranties, representations, or guarantees, express or implied, regarding the accuracy, completeness, or security of the reviewed code or the findings presented herein.

This document does not represent an endorsement, certification, or guarantee of the code’s functionality, safety, or suitability for production use.

No part of this report should be interpreted as legal, financial, or investment advice.

Users of this report are solely responsible for their own assessment, testing, and implementation decisions.
All information is provided “as is,” without any warranty of any kind. The authors and affiliated entities shall not be liable for any damages, losses, or other consequences resulting from reliance on this report or its contents.