WorldCoin Bridge Linea Security Review

Conducted By Pelz

Contents

About Pelz

I am an independent blockchain security researcher with extensive experience auditing over 10,000 lines of smart contract code, having uncovered numerous vulnerabilities across various projects. My goal is to provide my clients with the best possible security audit journey, ensuring thorough and professional assessments. While 100% security of your project can't be guaranteed, I strive to identify critical issues and offer solutions to enhance your project's resilience.

You can explore my portfolio at GitHub or connect with me on X.

Disclaimer

While I strive to deliver the most comprehensive and accurate security assessments, no audit can guarantee 100% security. This report is based on the information provided and my independent findings at the time of the audit. New vulnerabilities may emerge, and I recommend continuous monitoring and additional security reviews as your project evolves. The responsibility for addressing and resolving any identified issues ultimately lies with the project owners and developers.

About Worldcoin Bridge Linea

The LineaStateBridge contract is an essential part of the cross-chain infrastructure, enabling secure and efficient communication between Ethereum and the Linea blockchain. Its primary role is to act as an intermediary, retrieving state data, such as Merkle tree roots, from Ethereum and transmitting it to Linea. This process ensures that state updates are consistently propagated across both blockchains.

The contract also manages transaction fees on both Ethereum and Linea, requiring periodic transactions by users or operators to maintain the bridge's operations and ensure data consistency across the networks.

Project Audit Scope

The objective of this audit is to thoroughly review the codebase of the MeProtocol smart contracts. The goal is to ensure the contracts are secure, functionally correct, and of high quality, with a focus on identifying potential

vulnerabilities and ensuring robustness in the overall design and implementation.

Commit Hash: 036c98b265a973723e60d58461020b29cdf59916

Total SLOC: 300

Risk Classification

Severity	Impact:High	Impact:Medium	Impact:Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - Leads to a significant loss of assets in the protocol or significantly harm a group of users
- Medium - Only a small amount of funds is lost or core contract functionality is broken or affected
- Low - Can lead to any kind of unexpected behaviour with no major impact

Likelihood

- High - Attack path is possible with reasonable assumptions that mimic on chain conditions and the cost of the attack is relatively low compared to the value lost or stolen
- Medium - Only a conditionally incentivized attack vector but still likely
- Low - Has too many or too unlikely assumptions

Actions Required For Severity Levels

- High - Must fix (before deployment, if not already deployed)
- Medium - Should fix
- Low - Could fix

Executive Summary

During this audit, Pelz reviewed the Worldcoin Bridge Linea contracts and identified three issues throughout the assessment process.

Findings Count

Severity	Amount
High	1
Medium	1
Low	1
Total Findings	**3**

Findings Summary

ID	Title	Severity	Status
[H-01]	**Potential Fund Loss Due to Excess msg.value in Linea Messaging Service**	High	Resolved
[M-01]	**Incorrect Ownership Validation Based on isLocal Flag in CrossDomainOwnableLinea Contract**	Medium	Resolved
[L-01]	**Redundant Reinitialization of Zero Values in Constructor Leading to Gas Wastage**	Low	Acknowledged

Findings

[H-01] Potential Fund Loss Due to Excess `msg.value` in Linea Messaging Service

Severity

High

Description

The `LineaStateBridge.sol` contract uses the Linea messaging service to send messages from Layer 1 (L1) to Layer 2 (L2) while forwarding `msg.value` to cover the transaction fees. The fee required for automatic claiming on L2 is passed as part of the message call, while the remaining value (if any) is forwarded to the destination address.

However, the issue arises because certain functions in the contract, such as `transferOwnershipLinea`, `propagateRoot`, and others, accept `msg.value` even when no extra value beyond the required fee is necessary. If a user sends more than the required fee, the excess value will be lost and locked permanently, as the destination addresses do not require or expect this extra value, and there is no mechanism to recover it.

This problem affects multiple functions:

- `transferOwnershipLinea` at L147
- `propagateRoot` at L126
- Several other functions like L165 and L183 also forward `msg.value` without validating the amount.

Impact

Users sending transactions to the contract could inadvertently send more value than necessary, leading to permanent loss of funds. Since the excess value has no utility in the target addresses (such as ownership transfers or message propagation), it becomes unrecoverable and locked in the contract or at the receiving address.

Location of Affected Code

- L147
- L126
- L165
- L183

Recommendation

To prevent accidental loss of funds, introduce a check that ensures `msg.value` matches the expected fee. If the provided value is different from the required fee, the transaction should revert.

For example, in the `propagateRoot` function, the fix would look like this:

```
function propagateRoot() external payable {
    uint256 latestRoot = IWorldIDIdentityManager(worldIDAdd
ress).latestRoot();

    bytes memory message = abi.encodeCall(ILineaWorldID.rec
eiveRoot, (latestRoot));

    // Ensure that the correct fee is sent with the transac
tion
    if (msg.value != _feePropagateRoot) {
        revert IncorrectMessageFeeSent();
    }

    IMessageService(messageServiceAddress).sendMessage{ val
ue: msg.value }(
        lineaWorldIDAddress,
        _feePropagateRoot,
        message
    );

    emit RootPropagated(latestRoot);
}
```

Similar checks should be added to other functions like `transferOwnershipLinea` to ensure no excess value is sent and locked inappropriately.

[M-01] Incorrect Ownership Validation Based on `isLocal` Flag in `CrossDomainOwnableLinea` Contract

Severity

Medium

Description

The `CrossDomainOwnableLinea` contract uses the `isLocal` flag to determine how ownership validation should be performed. According to the comment, when `isLocal` is `true`, the contract should use the overridden cross-domain ownership check, and when `isLocal` is `false`, it should use the inherited `Ownable` contract's `_checkOwner` function.

However, the current implementation does not follow this logic. Instead of using `super._checkOwner()` when `isLocal` is `false`, the code still directly checks the owner against `msg.sender`. This mismatch between the comment and the actual behavior can lead to confusion about how ownership is validated, potentially causing unexpected access control issues.

Impact

Due to this discrepancy, ownership validation might not be handled as intended. Specifically:

- When `isLocal` is `false`, instead of falling back to the standard `Ownable` logic, the contract checks the ownership through custom logic.
- This could lead to security assumptions being broken, particularly if developers rely on the comment to understand how ownership checks are conducted.

Location of Affected Code

- L14-L16
- L73-L89

Recommendation

Update the code to align with the comment's intention:

- When `isLocal` is `false`, use `super._checkOwner()` to validate ownership using the inherited `Ownable` logic.

Alternatively, if the current implementation is correct and the comment is inaccurate:

- Correct the comment to accurately describe the current logic, ensuring that it reflects how ownership checks are performed in the contract.

This will eliminate confusion and ensure the contract behaves as expected based on its documentation.

[L-01] Redundant Reinitialization of Zero Values in Constructor Leading to Gas Wastage

Severity

Low

Description

In the `LineaStateBridge.sol` contract, the `DEFAULT_LINEA_FEE` is set to 0, which is meant to signal that certain transactions will have to be claimed manually on Layer 2 (L2). In the constructor, various internal fee-related variables are initialized to this default value of 0:

```
_feePropagateRoot = DEFAULT_LINEA_FEE;
_feeSetRootHistoryExpiry = DEFAULT_LINEA_FEE;
_feeTransferOwnership = DEFAULT_LINEA_FEE;
_feeSetMessageService = DEFAULT_LINEA_FEE;
```

Since these variables are of type `uint256`, which defaults to 0 in Solidity when left uninitialized, explicitly setting them to 0 again is redundant. This results in unnecessary gas consumption during contract deployment.

Explicit initialization to zero is only necessary if there is a plan to change the `DEFAULT_LINEA_FEE` to a non-zero value. Otherwise, leaving these variables uninitialized would save gas during deployment.

Impact

This redundancy leads to slightly higher gas consumption during the contract's deployment without adding any benefit. Although the impact on deployment cost is minor, optimizing gas usage is always important in Ethereum smart contract development, especially for contracts deployed multiple times.

Location of Affected Code

- L44
- L114-L117

Recommendation

Remove the redundant initializations of the fee-related variables in the constructor, as these variables default to 0. The lines where `_feePropagateRoot`, `_feeSetRootHistoryExpiry`, `_feeTransferOwnership`, and `_feeSetMessageService` are explicitly set to `DEFAULT_LINEA_FEE` (0) should be removed unless there are future plans to change `DEFAULT_LINEA_FEE` to a non-zero value. This would save gas and simplify the contract.