OpenZeppelin | security

Linea Bridge Audit

Linea

December 7, 2023

Table of Contents

Summary

Type	ZK Rollup
Timeline	From 2023-07-10 To 2023-08-2
Languages	Solidity

Total Issues	33 (20 resolved, 3 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	3 (1 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	9 (4 resolved, 1 partially resolved)
Notes & Additional Information	19 (13 resolved, 2 partially resolved)

Scope

We audited the Consensys/linea-contracts repository at the `f08c1906855198e2dc0413a47dcb38291b7087e5` commit. The OpenZeppelin team has also reviewed the public commit `0949c4096664e613f73ed3ce51c32f97fc56cdef` and confirmed that this contains all of the individual resolutions referenced in this report and is the final version fully reviewed during this audit.

In scope were the following contracts:

```
contracts
├── interfaces
│   ├── IGenericErrors.sol
│   ├── IL1MessageManager.sol
│   ├── IL2MessageManager.sol
│   ├── IMessageService.sol
│   ├── IPauseManager.sol
│   ├── IRateLimiter.sol
│   └── IZkEvmV2.sol
├── messageService
│   ├── l1
│   │   ├── L1MessageManager.sol
│   │   └── L1MessageService.sol
│   ├── l2
│   │   ├── L2MessageManager.sol
│   │   └── L2MessageService.sol
│   ├── lib
│   │   ├── Codec.sol
│   │   ├── PauseManager.sol
│   │   ├── RateLimiter.sol
│   │   ├── Rlp.sol
│   │   ├── TimeLock.sol
│   │   └── TransactionDecoder.sol
│   └── MessageServiceBase.sol
├── tokenBridge
│   ├── BridgedToken.sol
│   ├── interfaces
│   │   └── ITokenBridge.sol
│   └── TokenBridge.sol
├── ZkEvmV2Init.sol
└── ZkEvmV2.sol
```

System Overview

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to provide lower transaction fees than Ethereum's L1 by reducing the costs of transaction execution and verification.

This report presents our findings and recommendations on the smart contracts mentioned in the scope section above. The proof verification will be audited as part of a later report.

Architecture

The rollup is operated by 3 main components:

- **The coordinator**: Responsible for transmitting information between the L1 and the L2.
- **The sequencer**: Responsible for ordering, building and executing blocks of L2 transactions.
- **The prover**: Responsible for proving the correctness of state transitions due to transaction execution by building validity proofs.

These components are currently centralized and operated by the Linea team. However, their decentralization is part of the Linea roadmap.

These components interact with smart contracts on Ethereum that are part of the rollup, namely the `ZkEvmV2`, the `L1/L2MessageService`, and the canonical `TokenBridge` contracts. The `ZkEvmV2` contract provides data availability and verifies proofs to ensure the validity of state transitions during finalization.

The message service contracts are deployed on both L1 and L2, allowing users to send arbitrary messages between Ethereum and Linea. This messaging scheme powers the canonical token bridge, allowing users to bridge tokens between the two layers.

Smart Contracts

The `ZkEvmV2` and the message service contracts are the backbone of the Linea rollup.

As mentioned above, the sequencer builds blocks on L2 and executes the transactions within them. The prover then builds a proof to attest to the correct execution of these transactions. This proof is handed off to the `ZkEvmV2` contract on Ethereum by the coordinator, along with the blocks and the new state root. The data is validated against the proof to check the validity of the state transition. Once the validity proof has been checked, the blocks and the new state root of the rollup are considered finalized.

The message service is deployed on both the Ethereum and Linea chains and enables users to pass arbitrary messages between the two layers. This notably allows users to bridge ETH.

The canonical token bridge uses the message service to bridge other assets. When receiving a new token on the destination chain, the local token bridge deploys a new token contract using a standardized implementation. This new token represents its counterpart and can be bridged back to redeem the original token. It is also possible for the security council to link a separate token contract rather than the standard one in case a custom implementation is desired.

In this implementation, only one side of the bridge is home to the native token in a pair. As the token bridge operates with `safeTransferFrom` and `safeTransfer`, any token standard that supports this interface can be compatible with the token bridge, such as ERC-20 and ERC-777 tokens.

Both the L1 and L2 message services transmit cross-chain messages by emitting events. The coordinator listens to such events and relays the messages to the other side's message service. To incentivize the execution of bridging transactions on the other side, users are encouraged to include a postman fee. Postmen receive these fees in exchange for completing the message reception on the other layer by calling a specific function to claim messages. Users also have the option to claim their own messages. A small additional anti-DOS fee is applied when bridging in the L2 to L1 direction.

The rate limiter and pause manager contracts add limits to the system with the purpose of safeguarding it. Actors with privileged roles on the pause manager can pause different parts of the messaging system, as well as the finalization logic in the `ZkEvmV2` contract. The rate limiter also affects the message service and limits the amount of native ETH that can be transferred over a set amount of time. The token bridge separately implements pausability by inheriting OpenZeppelin's `PausableUpgradeable` contract.

Trust Assumptions & Risks

Several components of the rollup are considered to be inherently trusted. These can pose some risks for users. Such trust assumptions and associated risks include:

- **Coordinator and sequencer implementations**: It is assumed that the coordinator and the sequencer will work as described in the Linea documentation. This notably includes the assumption that the coordinator will remain online, correctly transmitting bridging transactions and finalizing blocks.
- **Censorship resistance**: The protocol is currently centralized, and the coordinator notably has the ability to censor all L2 transactions, including bridging transactions. There is currently no mechanism to force transaction inclusion: the protocol can ignore transactions and still successfully finalize. As such, there is no escape hatch mechanism to force the withdrawal of assets from the rollup.
- **No refund mechanism**: If a bridging transaction does not get successfully executed on the other side, there is currently no mechanism to retrieve funds that were sent with the initial bridging transaction. Such a situation could occur if the coordinator does not relay a transaction or if a transaction reverts on the other side.

Privileged Roles

Multiple privileged roles are present across the rollup and can have a high impact on its operations.

Messaging Service

For the messaging service, both the L1 and L2 inherit from the `PauseManager` contract. This contract uses OpenZeppelin's `AccessControlUpgradeable` in order to administrate the `PAUSE_MANAGER_ROLE`. The `PAUSE_MANAGER_ROLE` is able to pause different functionalities by calling `pauseByType` with one of several pause types: `GENERAL_PAUSE_TYPE`, `L1_L2_PAUSE_TYPE`, `L2_L1_PAUSE_TYPE` or `PROVING_SYSTEM_PAUSE_TYPE`.

The message services also inherit from the `RateLimiter` contract, which limits the amount of native ETH that can be withdrawn from L2. The `RATE_LIMIT_SETTER_ROLE` is the role that is able to change or reset the limit. The `RateLimiter` similarly uses `AccessControlUpgradeable` to administrate the `RATE_LIMIT_SETTER_ROLE`.

Token Bridge

The token bridge inherits OpenZeppelin's `Ownable2StepUpgradeable` contract to carry out privileged actions. The `owner` role is set to the address of the security council during initialization. The `setMessageService`, `setRemoteTokenBridge`, `setReserved`, `removeReserved`, `setCustomContract`, `pause`, and `unpause` functions can only be called by the owner. Notably, the ability of the owner to set the address of the message service and the address of the remote token bridge grant enough power to control or freeze the funds on the token bridge itself.

The token bridge contract is also able to call the `mint` and `burn` functions of the deployed token contracts.

ZkEvmV2

The `ZkEvmV2` contract inherits from the `L1MessageService` and thus contains all the privileges from the message service contracts mentioned above. In addition to these roles, the `ZkEvmV2` also inherits `AccessControlUpgradeable` in order to administrate the `OPERATOR_ROLE`.

The `DEFAULT_ADMIN_ROLE` is able to set the verifier address with the `setVerifierAddress` function, and can finalize blocks without a proof by calling the `finalizeBlocksWithoutProof` function. It is important to note that being able to finalize blocks without a proof effectively grants the `DEFAULT_ADMIN_ROLE` full control over the rollup state. This role is given to the security council on contract initialization.

The `OPERATOR_ROLE` is used by the coordinator and is allowed to relay blocks and proofs from L2 to L1 via the `finalizeBlocks` function.

Critical Severity

C-01 Token Bridge Reentrancy Can Corrupt Token Accounting

The `TokenBridge` contract gets deployed on both L1 and L2, allowing users to bridge tokens between the layers by calling the `bridgeToken` function. To compute the number of tokens that are being sent, the `bridgeToken` function first retrieves its current balance, transfers the tokens from `msg.sender` to itself, and then computes the difference between its new balance and the original one. This is done to handle tokens with unusual logic, such as those having a fee on transfers.

However, this function is vulnerable to reentrancy attacks through token callbacks, for example with ERC-777 tokens. This would allow an attacker to be credited with more tokens than they initially bridged. For example, in the case of an ERC-777-compliant token called "myToken", an attacker could:

1. Register an attacker-owned contract as an `ERC777TokensSender` to the ERC-1820 registry.
2. Call `bridgeToken` with 500 tokens. The `balanceBefore` is 0 and `safeTransferFrom` is called, triggering the ERC-777 callback to the sender.
3. In the ERC-777 `tokensToSend` callback, call `bridgeToken` again with 500 additional tokens. In this subcall, `balanceBefore` is still 0, `safeTransferFrom` is called successfully and `balanceAfter` is 500.
4. The main call will end with a `balanceAfter` of 1000.

Having sent 1000 tokens in total, the attacker would be credited with 1500 tokens on L2. Assuming other users have bridged this token to L2, the attacker could withdraw the 1500 tokens, effectively stealing 500 tokens from other users. Note that some ERC-777 tokens currently have enough liquidity to motivate such an attack, such as Skale, AMP or VRA.

Consider adding reentrancy protection to the `bridgeToken` function, such as OpenZeppelin's `ReentrancyGuard`.

Update: *Resolved at commit bb3691b.*

High Severity

H-01 Rate Limiters Can Lead to a Denial-of-Service Attack

The protocol uses rate limiters to limit its exposure in case of financial loss. These limit the amount of ETH that can be bridged from L2, as well as the amount of ETH that can be claimed on L1.

However, such protocol-wide rate limits can be used to DOS the protocol. For example, a well-funded attacker could deposit and withdraw the rate limit in ETH on each rate limit period. If this were to occur, users' ETH would be stuck on L2 as the rate limit would block withdrawals. Users would also be prevented from adding a postman fee, forcing them to be their own postman to get their transactions included on L1.

Consider removing the rate limiter or introducing more sophisticated limits. If it is kept, consider composing an action plan and incorporating monitoring for potential cases of abuse. Any change, such as more sophisticated rate limits, would have to be end-to-end tested.

Update: *Acknowledged, will resolve. The Linea team stated:*

> *This will be a future enhancement. Current withdrawal limits are placed on the bridge to minimize the impact of any potential malicious actors and will be removed at a later date.*

H-02 Tokens Sharing an Address on L1 and L2 Can Be Set Up to Steal User Funds

When new tokens are bridged, the token metadata is fetched and then sent via the `MessageService` to the destination `TokenBridge`. On the destination layer, a new Beacon Proxy pointing to the `BridgedToken` implementation is then deployed with `create2`.

Other than custom deployments by the Linea team, this pattern is standard for every token, where one layer should host the native token and the other layer should have an associated `BridgedToken` deployed.

Exploiting the asynchronous nature of the bridge, it is possible to circumvent this standard pattern and have two native tokens both deployed and linked on separate layers. With this configuration, it is possible for the controller of the contracts to siphon money from the bridge.

The pattern to deploy two linked native tokens in this invalid state is as follows:

1. Attacker bridges the token with `_recipient` `:= address(0)`. On the origin chain, the `nativeToBridgedToken` mapping will be set to `NATIVE_STATUS`. The message cannot be executed on the destination chain as minting to a `_recipient` of zero will revert.
2. Attacker bridges the token deployed to the same address as the origin chain on the destination chain. Similarly, the `nativeToBridgedToken` mapping will be set to `NATIVE_STATUS`.
3. When users attempt bridging of this token, the bridge will treat the tokens as native, attempting to `safeTransfer` tokens to users on the destination chain. Since the attacker controls the entire supply on the destination layer, the bridge will not have any funds to compensate the user with.
4. Now the attacker is able to mint tokens on the destination layer and bridge back, extracting all of the origin tokens sitting on the token bridge.

It is possible to use either `create` or `create2` to deploy a token on both layers at the same address. The above attack relies on `completeBridging` not being called before step 2 to avoid the token being deployed as a `BridgedToken`. This can be achieved by several means, such as setting `address(0)` as `_recipient` or setting a fee of 0 to disincentivize `completeBridging` from being called by postmen.

Consider only updating mappings in `completeBridging` to ensure atomicity, as well as passing both the local and remote addresses as arguments when bridging to introduce additional checks. Such changes should be end-to-end tested, notably under scenarios where bridging transactions are asynchronous and interleaved.

Update: *Resolved at commit bb3691b.*

H-03 ERC-721 Tokens Can Be Locked on the Token Bridge

The `TokenBridge` contract allows tokens to be bridged across layers with the `bridgeToken` function. When a native token is bridged, the token bridge will call `safeTransferFrom` to move the token from the caller to itself. This `safeTransferFrom` function is defined in OpenZeppelin's `SafeERC20Upgradeable` contract.

While this definition of `safeTransferFrom` is intended to be used solely with ERC-20 tokens, it is possible to move ERC-721 tokens with it. Inside the function call, the low-level `transferFrom` function is actually used to transfer the token. The ERC-721 standard contains this function signature and uses it to transfer tokens.

If a user attempts to bridge an ERC-721 token with the token bridge, the first half of the bridging, where the tokens are collected onto the bridge, will succeed. However, the bridge uses `safeTransfer` when bridging native tokens out, which uses the low-level `transfer` function. Unfortunately, `transfer` is not supported by the ERC-721 standard, resulting in the ERC-721 being locked on the token bridge, with an upgrade being the only way to recover it.

Consider blocking the bridging of ERC-721 tokens, for example by removing the default decimal fallback during metadata collection. By reverting when a contract does not properly support the decimals function selector, all standard ERC-721 contracts would revert before sending any tokens to the contract.

Update: *Acknowledged, will resolve. The Linea team stated:*

> *This will be a future addition.*

Medium Severity

M-01 Tokens Sharing an Address Across L1 and L2 Can Prevent Users From Bridging

Users can call the `bridgeToken` function on the origin layer to start the bridging process. The `completeBridging` function can then be called on the destination layer to complete it. Bridges track tokens internally using their native address, defined as the address of the token on the chain from which it was first bridged. A token can thus only be native on either L1 or L2. If a token with the same address on L1 and L2 is encountered, a choice is made as to which token is the "real" native one by picking the first one that is bridged. It then becomes impossible to bridge a token with the same address in the other direction.

However, this mechanism can be exploited as a DOS attack to prevent users from bridging. This can be done as follows:

1. An attacker deploys a token on L2 with the same address as a token on L1 which has not been bridged yet.

2. The attacker calls `bridgeToken` on L2, followed by `completeBridging` on L1.
3. The token is now considered native to L2, and users can no longer bridge this token in the L1 to L2 direction.

A token can be deployed to the same address on L1 and L2 by using `create2`. At a high level, the issue above arises because of the assumption in the bridge logic that a token cannot be native on both chains. A solution would ideally result in two `bridgedToken` instances being deployed, one on each chain, each associated with its own native token.

Consider using a different prevention method that allows native tokens to exist on both chains at the same address. Such changes would have to be end-to-end tested.

Update: *Resolved at commit bb3691b.*

Low Severity

L-01 Error-Prone and Inconsistent Definition of Gap Variables

The `L2MessageService` contract defines gap variables to support the future addition of state variables when upgrading.

However, while most gaps in the codebase are defined after the other storage variables, the `L2MessageService` contract defines this gap before. This makes it inconsistent with the rest of the codebase in addition to being error-prone for future updates. New variables cannot be added to the end of the existing storage variables without resulting in a storage collision.

Consider defining the gap variable after other storage definitions in the `L2MessageService` contract to be consistent with the rest of the codebase. If this is impossible, consider documenting this issue thoroughly to ensure storage variables are added to the front of the storage definitions in this file to avoid storage collisions.

Update: *Resolved in pull request #9 at commit 2a3c0b0.*

L-02 Incorrect Parent Initializer in BridgedToken

The `BridgedToken` contract inherits from `ERC20PermitUpgradeable`, however, it calls `__EIP712_init` instead of calling its parent initializer `__ERC20Permit_init`. Consider calling `__ERC20Permit_init` to follow best practices and remain compatible with any future changes to the `ERC20PermitUpgradeable` contract's codebase.

Update: *Resolved at commit bb3691b.*

L-03 Pragma Compiler Version Default Can Generate Incompatible Opcodes

Valid compiler versions for the codebase are set via the pragma solidity tag at the top of Solidity files. The codebase uses `pragma solidity ^0.8.19`, requiring the compiler version to be at least 0.8.19 and lower than 0.9.0.

However, Linea currently only supports EVM versions up to London. This notably means that Linea does not support the `push0` opcode which was introduced in Shanghai. Additionally, Shanghai is the default EVM version starting from Solidity compiler version 0.8.20. This means that contracts compiled with 0.8.20 by default might not be supported on Linea.

Consider fixing the pragma version to 0.8.19 in the codebase to limit the likelihood of this occurring, as well as documenting the supported EVM versions.

Update: *Resolved in pull request #6 and commit bb3691b.*

L-04 Pointer Arithmetic Error in RLP Library

During finalization, L1 to L2 transactions that have been transmitted to L2 by the coordinator are marked as received back on L1. To do so, the transactions relaying the message hashes to the L2 message service are RLP-decoded to extract the hashes marked as received during finalization.

However, the RLP library used for decoding includes a pointer arithmetic error when computing the bounds of the RLP-encoded transactions. When decoding long lists, the `endPtr` is computed as `_self.item.memPtr + 1 + lenX` while it should be computed as `_self.item.memPtr + 1 + byteLen + lenX`.

```
Long list RLP-encoded transaction:
|-----------------------|-----------------------------|------------------|
|  length of length     | length of payload in bytes  |      payload     |
|        1 byte         |        (byteLen bytes)      |   (lenX bytes)   |
|-----------------------|-----------------------------|------------------|
^                                                                ^        ^
_self.item.memPtr (initial memPtrStart)                     endPtr   endPtr + byteLen
```

Given the current usage of the library, this arithmetic error will not result in adverse effects. This is because `byteLen` will never exceed 5 bytes in practice, while the abi encoding of the hashes will always be over 64 bytes in length. Still, for correctness and to avoid potential future issues, consider fixing the end pointer calculation.

***Update**: Acknowledged, will resolve. The Linea team stated:*

> *Will resolve, using extensive sample testing for generic use.*

L-05 Validators Can Steal Refunds of EOA Users When Claiming Messages

When a message with empty calldata and an EOA as a recipient is claimed on L1, part of the fees is reimbursed to the destination address. This refund is computed as `_feeInWei - deliveryFee` where `deliveryFee = min(_feeInWei, (startingGas + REFUND_OVERHEAD_IN_GAS - gasleft()) * tx.gasprice)`.

However, this way of computing the refund is vulnerable to validators claiming messages and inflating `tx.gasprice` by setting a high transaction tip. A validator could inflate the tip to where the `deliveryFee` is equal to the maximum fee set on the message. A user setting a high fee to ensure their transaction is picked up by a postman would effectively get their refund stolen.

Consider documenting this behavior. Generally, users should not be encouraged to rely on the refund mechanism as it is susceptible to leaks from MEV.

***Update:** Acknowledged, not resolved. The Linea team stated:*

> *This will be a potential future enhancement. Documentation will be updated as per the recommendation.*

L-06 Functions Lack Pausability

The `TokenBridge` contract can be paused to prevent some functions from being called for a period of time. This functionality is intended as a contingency plan to prevent loss of funds in case something unexpected happens.

However, several functions can still be called while the contract is paused, notably the `completeBridging` and `confirmDeployment` functions.

Consider making these functions pausable.

Update: *Partially resolved at commit bb3691b. The `completeBridging` function is now pausable. The Linea team stated:*

> *`confirmDeployment` is a Linea function with no impact on funds.*

L-07 Funds Can Be Locked in the Bridge

In order to bridge funds, users can call the `bridgeToken` function on the origin chain. Later, on the destination chain, the `completeBridging` function can be called to transfer the bridged tokens to the `_recipient`. It is possible to have funds effectively stuck on one side of the bridge if `completeBridging` is unable to successfully execute.

When the destination token is an instance of the `bridgedToken` contract, the `TokenBridge` will `mint` and `burn` tokens directly from a user's account instead of using `transferFrom`. The `TokenBridge` has the proper permissions to `mint` and `burn` from the `bridgedToken` instances. However, the zero address is specifically protected in the `mint` function, resulting in a revert when attempting to `mint` to the zero address.

Even though sending funds to the zero address should only be done when attempting to lock up tokens, in this scenario funds would be locked on the `TokenBridge` and not on the zero address itself.

Consider reverting in the `bridgeToken` function when the `_recipient` is specified to be the zero address.

Update: *Resolved at commit bb3691b.*

L-08 Denial-of-Service Protection Not Initialized

The `L2MessageService` contract defines a state variable called `minimumFeeInWei` as a minimum fee to pay to transfer L2-to-L1 messages. Such a fee is intended as a way to "address DOS protection".

However, this fee is not set in the initializer, meaning that its value is initially 0 until `setMinimumFee` is called.

If `minimumFeeInWei` is required to address a DOS risk that could be present from the launch of the protocol, consider initializing it in the initializer.

Update: *Acknowledged, will resolve. The Linea team stated:*

> *This will be a future enhancement for testnet deployments. Mainnet has the fee set, so it includes Denial-of-Service protection.*

L-09 Bridge Tokens Default to 18 Decimals

When a new token is bridged, its metadata is collected and sent alongside the first transfer so that a new `BridgedToken` can be deployed on the destination layer. To collect this metadata, static calls are made to the `name()`, `symbol()` and `decimals()` methods of the token address. Similarly to Hermez's bridge, default values are assumed in case the metadata could not be fetched/decoded.

However, assuming default values for decimals may surprise users. For instance, in the case of a token with 0 decimals being defaulted to 18 on L2, a user bridging 100 tokens may not expect to receive only `100 * 10^-18` tokens on the target chain.

While it is possible to deploy a contract with the same metadata as the original on the destination chain and ask the Linea team to call the `setCustomContract` function to link the two contracts through the bridge, this option becomes unavailable once the token has been initially bridged.

Consider removing default values for metadata and reverting on unsupported tokens. These tokens would then have to be bridged following the second method underlined above. Whether or not this suggestion is implemented, any potential consequences for users should be properly documented.

Update: *Acknowledged, not resolved. The Linea team stated:*

> *This will be a potential future enhancement. Documentation will be updated as per the recommendation.*

Notes & Additional Information

N-01 Bridging Completion May Be Blocked by Reserve Status

After initiating a bridging transaction of a token for the first time, a postman can call `completeBridging` on the destination bridge. Once they do, a new instance of `BridgedToken` is deployed behind a beacon proxy.

However, if the `nativeToBridge` mapping for the token on the destination chain is set to `RESERVED_STATUS` when executing `completeBridging`, the transaction will fail. This is because a `mint` call to `address(0x111)` (`RESERVED_STATUS`) will revert. This could lead to tokens being temporarily stuck on the origin bridge until administrators call `removeReserved` on the destination chain to set the status back to `EMPTY`.

This situation could happen if an administrator mistakenly calls `setReserved` for a token on a chain it is not native on.

To limit the likelihood of such a scenario happening, consider adding an additional warning in the NatSpec of the `setReserved` function that `_token` should be a native token on the chain.

Update: *Resolved in pull request #8 at commit 952bf56.*

N-02 Custom Error Redefinition

The abstract contract `MessageServiceBase` defines the `ZeroAddressNotAllowed` error, which is also defined in the `IGenericError` interface.

Consider reusing `IGenericError` in `MessageServiceBase` to avoid defining it twice.

Update: *Resolved in pull request #13 at commit d6c4399.*

N-03 Custom Errors Recommendations

Some unused errors and potential improvements were detected:

- Some errors ([1] [2] [3] [4] [5]) are defined but never used. Consider removing them.
- In the `IL1MessageManager` interface, the `MessageDoesNotExistOrHasAlreadyBeenClaimed` error could return the `messageHash` as an argument to be consistent with the other errors and make debugging easier for users.
- In the `ZkEvmV2` contract, the `BlockTimestampError` error could include the `blockInfo.l2BlockTimestamp` and `block.timestamp`.
- In the `TransactionDecoder` library, the `UnknownTransactionType` error could include the version byte.

Consider updating these errors to improve the clarity of the codebase.

Update: *Resolved in pull request #13.*

N-04 Hardcoded Values Are Error-Prone

When a new message is claimed, `L1/2MessageService` exposes a `_messageSender` state variable that can be queried to get the cross-domain address. `_messageSender`'s default value is hard-coded to `address(123456789)`.

Consider using a constant instead of a hard-coded value to be clearer and less error-prone. For the same reasons, consider setting the constant permit function selector to `IERC20PermitUpgradeable.permit.selector` instead of computing the `keccak256` hash of a string.

Update: *Resolved in pull request #14 at commit 985ee5f.*

N-05 Inconsistent Licenses

Some files have an `Apache-2.0` license while others have an `AGPL-3.0` license.

Consider standardizing the license across the codebase.

Update: *Acknowledged, not resolved. The Linea team stated:*

> *Licenses are decided on a per-contract level and were purposely set.*

N-06 Multi-Line Comments Do Not Consistently Follow NatSpec Format

Some multi-line NatSpec comments end with a `**/` instead of the recommended `*/`.

Similarly, there is an instance of a multi-line comment starting with `/*` rather than the recommended `/**`.

Consider fixing these instances to be consistent in case external services depend on a strict NatSpec format.

Update: *Resolved in pull request #9 at commit 4f6e4ea.*

N-07 Incorrect or Misleading Documentation

Throughout the codebase, a few instances of incorrect or misleading documentation were identified:

- In the `IMessageService` interface, a comment has been duplicated for two different events.
- In the `IL1MessageManager` interface, the NatSpec for the `MessageDoesNotExistOrHasAlreadyBeenClaimed` error could mention the possibility that the message does not exist.
- In the `L1MessageManager` contract, there is a comment indicating that "There is a uint216 worth of storage layout here". However, this comment is incorrect as the variables above are constant and do not occupy storage space. Consider removing it. This also occurs in the `L2MessageManager`.
- In the `TokenBridge` contract, the default returned value for token names is "UNKNOWN" and not "NOT_VALID_ENCODING".
- In the `IRateLimiter` interface, the NatSpec around the `resetRateLimitAmount` function describes the `_amount` parameter as being "New message hashes". Consider editing this comment to better describe that it sets the maximum amount of ETH that can be withdrawn over a period.
- In the `TokenBridge`, the NatSpec for the `_safeName` function incorrectly documents two arguments when there should only be one.
- In the `IZkEvmV2` interface, the NatSpec for the `finalizeBlocks` should be replaced by this one.

Clean and accurate documentation helps users and developers understand the codebase. Consider updating the identified instances of incorrect or misleading documentation.

Update: *Resolved in pull request #12 and commit bb3691b.*

N-08 Lack of Indexed Event Parameters

Throughout the codebase, several events do not have their parameters indexed. For instance:

- In the `IPauseManager` interface, the `Paused` and `UnPaused` events could have the `pauseType` parameter indexed.
- In the `ITokenBridge` interface, most events could have their parameters indexed.

Consider indexing event parameters to improve the ability of off-chain services to search and filter for specific events.

Update: *Resolved in pull request #15 at commits c921621 and bb3691b.*

N-09 Missing Event Parameter

We have identified a few instances of events which would benefit from additional event parameters:

- The `NewTokenDeployed` event is emitted to signal that a new bridged token has been deployed by the `TokenBridge`. Consider adding the address of the native token to the emitted parameters of the event.
- The `VerifierAddressChanged` event only emits the address of the new verifier and could benefit from also emitting the old address.
- The `BlockFinalized` event could emit a flag indicating whether the block was finalized using a proof.

Consider adding the suggested parameters to their respective events to improve the clarity and transparency of the records.

Update: *Partially resolved in pull request #15. The `BlockFinalized` event remains unchanged.*

N-10 Only One Permit Standard Supported

The `TokenBridge` supports permit-style approvals via the `_permit` function, invoked when calling the external `bridgeTokenWithPermit` function. This function supports ERC-2612, a permit standard now widely used among ERC-20 and ERC-712 tokens.

However, there are permit standards other than ERC-2612. Most notably, Dai from MakerDAO uses a different signature for their permit function.

In order to support permit functionality on more tokens, consider adding other permit methods to the permit function of the `TokenBridge`.

Update: *Acknowledged, will resolve. The Linea team stated:*

> *This will be a future addition.*

N-11 Typographical Errors

We identified the following typographical errors in the codebase:

- "checked," [1] [2] should be "checked."
- "store" should be "stored".
- "claim" should be "claims".
- "reserved" [1] [2] [3] [4] should be "reserve".
- "are" [1] [2] should be "have".
- "subject to a by a time lock" should be "subject to a delay by a time lock".
- "updated" should be "updates".
- "achoring" should be "anchoring".
- "EIP29230" should be "EIP2930".
- "neccesary" should be "necessary".

To improve the overall consistency and readability of the codebase, consider updating the identified errors and applying an automated spelling/grammar checker to the codebase to identify further instances.

Update: *Partially resolved in pull request #9.*

N-12 L2 Timestamp Not Initialized in ZkEvm Contract

The `ZkEvmV2` contract exposes a `currentTimestamp` state variable which is used to store the last L2 block timestamp.

However, this variable is not initialized alongside the `currentL2BlockNumber` and `stateRootHashes` variables. This means that on initialization, the block number might not match the timestamp as the timestamp is set to 0. In the case of a migration, this could break the assumption that the rollup timestamps are strictly increasing.

Consider adding an argument to the `initialize` function to initialize the `currentTimestamp` variable.

Update: *Acknowledged, will resolve. The Linea team stated:*

> *This will be a future enhancement for testnet deployments. Mainnet has the timestamp set.*

N-13 Unused Import

In `L2MessageService.sol`, the import of `CodecV2` is unused and could be removed.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #9.*

N-14 Codec Function Is Not Memory Safe

The `_extractXDomainAddHashes` function is used to extract the message hashes from the calldata of transactions calling `addL1L2MessageHashes` on L2.

However, this function modifies its input bytes array `_calldataWithSelector` argument in-place by changing its length and making it invalid. This is not an issue with the way the code is currently used as the function is called on a local memory copy which is not used for anything else.

Consider documenting the fact that the input of this function is modified and should not be reused.

Update: *Resolved in pull request #9 at commit 2c71574.*

N-15 Lack of Documentation in `bridgeTokenWithPermit` Function

The `TokenBridge` contract exposes a `bridgeTokenWithPermit` function to allow users to approve and bridge in a single transaction. This function calls an internal `_permit` function which calls the ERC-2612 `permit` function on the token address.

However, the `_permit` function passing does not mean that the allowance has been updated or that the user has been authenticated. For example, calling `_permit` with the address of WETH on layer 1 would pass without being supported because of its silent fallback method.

While there is currently no issue with the way the function is being used, consider documenting a warning not to rely on this function for authentication.

Update: *Resolved in pull request #8.*

N-16 Variable Naming

The array `hashOfRootHashes` in the `ZkEvmV2` contract could be renamed to `blockRootHashes`, as it does not contain the hash of root hashes, but the individual root hashes.

Consider renaming the variable to improve the clarity of the codebase.

Update: *Resolved in pull request #9 at commit ac6185e.*

N-17 Gas Optimizations

- In the `TokenBridge` contract, the `nativeToken` variable is set on each loop but remains unused.
- There are multiple instances of array lengths being evaluated on each loop ([1] [2] [3] [4] [5] [6] [7]). These could be written to memory to save gas.
- In the `RLPReader` library, the constant `LIST_SHORT_START_MAX` is unused and could be removed.

Consider updating these to save gas during the operation/deployment of the protocol.

Update: *Acknowledged, not resolved. The Linea team stated:*

> *There will be very few items on `TokenBridge` and the gas savings are negligible or null. In the `zkEvmV2` contract, the high volume of data, in practice, is more costly due to exponential memory costs.*

N-18 Lack of Security Contact

Providing a specific security contact (such as an email or ENS) within a smart contract significantly simplifies the process for individuals to communicate should a vulnerability be identified in the code. Additionally, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the creators of those libraries to make contact, inform the code owners about the problem, and provide mitigation instructions.

Consider adding a NatSpec comment on top of contract definitions with a security contact. Using the `@custom:security-contact` tag is recommended as it has been adopted by the Openzeppelin Wizard and the ethereum-lists.

***Update:** Resolved in pull request #11 at commit 001bf37.*

N-19 `BlockData` Struct Lacks Documentation

The `BlockData` struct is an important part of the proving system but is not documented. Consider adding documentation describing the meaning of its different components to improve the clarity of the codebase.

***Update:** Resolved in pull request #10.*

Conclusion

During this three-and-a-half-week audit, one critical and three high-severity vulnerabilities were identified. Three of these affected the token bridge, while the other one was related to the potential abuse of the rate limiter. These were promptly communicated to the Linea team and resolved/acknowledged.

As it currently stands, we found the codebase to be clear and well-documented. Several recommendations to further improve its readability and facilitate future development and integrations were made. Working with the Linea team has been a great experience. We thank them for their responsiveness as well as for providing us with extensive specifications for the rollup and the message service.