Linea Burn Mechanism Audit Report

Prepared by Cyfrin

Version 2.2

Lead Auditors

0xStalin

MrPotatoMagic

November 3, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

4.1 Protocol summary

The Linea Burn Mechanism implements a dual-token burn protocol designed to allocate Layer 2 (L2) transaction fees toward operational reimbursement while simultaneously enhancing the strenght of Ethereum (`ETH`) and the Linea (`LINEA`) token.

- **Operational Reimbursement**: Network operators submit invoices for reimbursement, with payments drawn directly from the L2 fee revenue
- **Surplus Allocation and Token Burn**: Following full settlement of operational expenditures, any residual fee surplus is deployed to execute market purchases of `LINEA` tokens. These acquired tokens are subsequently removed from circulation through permanent burning, thereby reducing the `LINEA` token supply

5 Audit Scope

The audit scope was at commit 8285efababe0689aec5f0a21a28212d9d22df22e:

```
contracts/src/operational/L1LineaTokenBurner.sol
contracts/src/operational/RollupRevenueVault.sol
contracts/src/operational/V3DexSwap.sol
contracts/src/operational/interfaces/IL1LineaTokenBurner.sol
contracts/src/operational/interfaces/IRollupRevenueVault.sol
contracts/src/operational/interfaces/IV3DexSwap.sol
```

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Linea Burn Mechanism smart contracts provided by Linea. In this period, a total of 13 issues were found.

During the audit, we identified 1 low severity issue, with the remainder being informational and gas optimizations.

The identified low-severity vulnerability stems from a type inconsistency in the `tickSpacing` field within the swap configuration struct used by `V3DexSwap`. The field is declared as `uint24`, whereas the corresponding parameter in the swap router interface is defined as `int24`. This mismatch may lead to unintended behavior during swap execution, particularly in scenarios involving negative tick values.

The informational issues encompass a series of non-critical enhancements focused on robustness, clarity, and maintainability:

- **Event Emission Gaps**: Certain state transitions lack accompanying event emissions, reducing transparency and complicating off-chain monitoring and auditing
- **Missing Input Validation**: Several functions omit prerequisite checks that would prevent invalid operations or edge-case failures, potentially leading to revert errors or inconsistent state
- **Redundant or Unreachable Code**: The `RollupRevenueVault` contract includes both an initialization function and a reinitialization function intended for proxy upgrades. However, since the proxy was deployed in an already-initialized state, the standard initialization function is permanently unusable and constitutes dead code

Post Audit Notes

All issues have been fully resolved in commit 77b83d2. A minor NatSpec adjustment to rename a variable was applied in commit efe83ff—this change **does not affect** the bytecode or ABI of the contracts.

The bytecode and ABI from the `main` branch have been verified to **exactly match** the versions following the post-audit fixes.

Both commits have been merged into the `main` branch of `linea-monorepo` and are included in the release tag `contract-audit-2025-10-30`.

Post Deployment Notes

The following contracts have been successfully deployed to their respective networks:

- **RollupRevenueVault:** Deployed at `0x84a5ba2c12a15071660b0682b59e665dc2faaedb` on **Linea**
- **V3DexSwapAdapter:** Deployed at `0x30a20a3a9991c939290f4329cb52daac8e97f353` on **Linea**
- **L1LineaTokenBurner:** Deployed at `0x5Ad9369254F29b724d98F6ce98Cb7bAD729969F3` on **Ethereum Mainnet**

The **deployed bytecode** of all three contracts has been **fully verified** and confirmed to **exactly match** the corresponding bytecode generated from the source code at the official release tag:

`contract-audit-2025-10-30`

This verification ensures **deterministic compilation**, **source-to-bytecode integrity**, and **audit alignment** with the reviewed version.

Summary

Project Name	Linea Burn Mechanism
Repository	contract-freeze-2025-10-12
Commit	8285efababe0...
Fix Commit	77b83d2ce990...
Audit Timeline	October 20th - October 24th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Informational	10
Gas Optimizations	2
Total Issues	13

Summary of Findings

[L-1] Tick spacing type mismatch in ExactInputSingleParams	Resolved
[I-1] Inconsistent deadline check in `V3DexSwap.swap`	Resolved
[I-2] Unnecessary implementation of the `RollupRevenueVault::initialize` given that the deployed proxy is already initialized	Resolved
[I-3] emiting `EthReceived` event when `RollupRevenueVault` receives 0 eth	Resolved
[I-4] Lack of validation to prevent receiving less LINEA tokens for the swap than the expected `minAmountOut`	Resolved
[I-5] No event emission when initializing parameters	Resolved
[I-6] Require condition `_minLineaOut > 0` offers no protection against malicious BURNER$_ROLEbehaviour$	Acknowledged
[I-7] Condition `_invoiceAmount != 0` in submitInvoice() can prevent clearing existing debt	Resolved
[I-8] Burn and bridge mechanism can be delayed due to paused token bridge state	Acknowledged
[I-9] Function updateInvoiceArrears() updates `lastInvoiceDate` even when `invoiceArrears` remains unchanged	Acknowledged
[I-10] Function updateInvoiceArrears() does not emit old values	Resolved
[G-1] Unnecessary ETH to WETH conversion during swap	Resolved
[G-2] Redundant call to sinc LINEA token supply on `L1LineaTokenBurner`	Resolved

7 Findings

7.1 Low Risk

7.1.1 Tick spacing type mismatch in ExactInputSingleParams

Description: V3DexSwap uses the Ramses V3 Swap Router to perform WETH to Linea swaps.

However, the ExactInputSingleParams struct definition passed as parameter to the exactInputSingle function differs between the V3DexSwap and the router contract.

V3SwapDex definition:

```
struct ExactInputSingleParams {
    address tokenIn;
    address tokenOut;
    uint24 tickSpacing; <<
    address recipient;
    uint256 deadline;
    uint256 amountIn;
    uint256 amountOutMinimum;
    uint160 sqrtPriceLimitX96;
  }
```

Ramses Router:

```
struct ExactInputSingleParams {
        address tokenIn;
        address tokenOut;
        int24 tickSpacing; <<
        address recipient;
        uint256 deadline;
        uint256 amountIn;
        uint256 amountOutMinimum;
        uint160 sqrtPriceLimitX96;
    }
```

As we can see, the tickSpacing member has differing types: uint24 and int24.

Impact: Due to this, if the POOL_TICK_SPACING value is greater than type(int24).max i.e. 8388607, the value will be interpreted incorrectly as a negative value in the router, causing a revert if such a pool does not exist or a successful swap through an unintended pool that anyone can frontrun create on RamsesV3.

Proof of Concept: Recommended Mitigation: Update the struct definition in V3DexSwap to use int24 for tickSpacing instead of uint24.

Linea: Fixed at commit be1cbc

Cyfrin: Verified.

7.2 Informational

7.2.1 Inconsistent deadline check in `V3DexSwap.swap`

Description: The swap() function in V3DexSwap implements the require check in the snippet below which ensures block.timestamp is strictly less than the `_deadline`.

```
require(_deadline > block.timestamp, DeadlineInThePast());
```

However, the Ramses V3 Swap Router allows swaps to occur even when the block.timestamp is equal to the deadline.

```
modifier checkDeadline(uint256 deadline) {
        if (_blockTimestamp() > deadline) revert Old();
        _;
    }
```

Impact: Due to this inconsistency, if `_deadline` is passed as block.timestamp to the `V3DexSwap.swap` function, the call would revert even though it's a valid value accepted by the router.

Proof of Concept: Recommended Mitigation:

Linea: Fixed at commit e531e7

Cyfrin: Verified.

7.2.2 Unnecessary implementation of the `RollupRevenueVault::initialize` given that the deployed proxy is already initialized

Description: The `RollupRevenueVault` contract will be used to upgrade the implementation of the proxy The proxy is already initialized; therefore, the `_initialized` flag is already set to 1. This means that no function calling the `initializer` modifier can be executed again, not even after the upgrade. This means that the only alternative to initializing the values of the new implementation is to call the `reinitializer` modifier, which is invoked by the `RollupRevenueVault::initializeRolesAndStorageVariables` function.

```
    function initialize(
        ...
@>  ) external initializer {
        ...
    }

    function initializeRolesAndStorageVariables(
        ...
@>  ) external reinitializer(2) {
        ...
        );
    }
```

Proof of Concept: Run the following PoC to verify the upgrade only works when calling the `initializeRole-sAndStorageVariables` function and reverts when calling the `initialize` function.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "forge-std/Test.sol";
import {
    TransparentUpgradeableProxy,
    ITransparentUpgradeableProxy
} from "@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";

import {RollupRevenueVault} from "src/operational/RollupRevenueVault.sol";

import {Vm} from "forge-std/Vm.sol";
```

```
contract RollupRevenueVaultUpgradeSimulationTest is Test {
    ITransparentUpgradeableProxy public proxy;
    RollupRevenueVault public newImpl;
    RollupRevenueVault public vaultUpgraded; // Proxy cast to upgraded interface

    bytes32 ADMIN_SLOT = 0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;
    bytes32 IMPLEMENTATION_SLOT = 0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

    // Existing proxy address on Linea
    address public proxyAddr = 0xFD5FB23e06e46347d8724486cDb681507592e237;

    // Fork URL for Linea mainnet
    string public lineaRpc = "https://linea-mainnet.g.alchemy.com/v2/<ALCHEMY_API_KEY>";

    // Variables for initialization
    address public admin;
    address public invoiceSubmitter;
    address public burner;
    address public invoicePaymentReceiver;
    address public tokenBridge = address(100);
    address public messageService = address(101);
    address public l1LineaTokenBurner = 0x000000000000000000000000000000000000dEaD;
    address public lineaToken = address(102);
    address public dex = address(103);
    uint256 public lastInvoiceDate;

    function setUp() public {
        // Create and select fork of Linea mainnet
        uint256 forkId = vm.createFork(lineaRpc);
        vm.selectFork(forkId);

        // Set the proxy
        proxy = ITransparentUpgradeableProxy(payable(proxyAddr));

        assertGt(address(proxy).balance, 0);

                //@audit-info => Doesn't work because the caller is not the admin :) !
                // admin = proxy.admin();

        // Get the current admin
        admin = getAdminAddress(address(proxy));

        // Set mock/test values for roles and receiver (in production, these would be specific
        ↪  addresses)
        invoiceSubmitter = vm.addr(1); // Placeholder, replace with actual if known
        burner = vm.addr(2); // Placeholder, replace with actual if known
        invoicePaymentReceiver = vm.addr(3); // Placeholder, replace with actual if known
        lastInvoiceDate = block.timestamp; // Use current timestamp for simulation

        // Deploy the new implementation
        newImpl = new RollupRevenueVault();

        // Encode the initialize calldata for upgrade (initialize())
        bytes memory initDataUpgrade_initialize = abi.encodeWithSelector(
            RollupRevenueVault.initialize.selector,
            lastInvoiceDate,
            admin, // Keep the same admin
            invoiceSubmitter,
            burner,
            invoicePaymentReceiver,
            tokenBridge,
            messageService,
            l1LineaTokenBurner,
            lineaToken,
            dex
        );

//@audit => upgrade fails when calling initialize because of the initializer modifier
        vm.prank(admin);
        vm.expectRevert();
        proxy.upgradeToAndCall(address(newImpl), initDataUpgrade_initialize);

        // Encode the reinitialization calldata for upgrade (reinitializer(2))
        bytes memory initDataUpgrade_reinitialize = abi.encodeWithSelector(
            RollupRevenueVault.initializeRolesAndStorageVariables.selector,
            lastInvoiceDate,
            admin, // Keep the same admin
            invoiceSubmitter,
            burner,
            invoicePaymentReceiver,
            tokenBridge,
            messageService,
            l1LineaTokenBurner,
            lineaToken,
            dex
        );

//@audit => upgrade succeeds when calling initializeRolesAndStorageVariables because of the
↪   reinitializer modifier
        vm.prank(admin);
        proxy.upgradeToAndCall(address(newImpl), initDataUpgrade_reinitialize);


        // Cast the proxy to the new interface
        vaultUpgraded = RollupRevenueVault(payable(address(proxy)));
    }

    function testUpgradeSimulation() public {
        // Verify the implementation has been updated
        assertEq(getImplementationAddress(address(proxy)), address(newImpl));

        // // Verify initialization values after upgrade
        assertEq(vaultUpgraded.lastInvoiceDate(), lastInvoiceDate);
        assertEq(vaultUpgraded.invoicePaymentReceiver(), invoicePaymentReceiver);
        assertEq(vaultUpgraded.dex(), dex);
        assertEq(address(vaultUpgraded.tokenBridge()), tokenBridge);
        assertEq(address(vaultUpgraded.messageService()), messageService);
        assertEq(vaultUpgraded.l1LineaTokenBurner(), l1LineaTokenBurner);
        assertEq(vaultUpgraded.lineaToken(), lineaToken);

        // Verify invoice arrears starts at 0
        assertEq(vaultUpgraded.invoiceArrears(), 0);

        // Verify constants
        assertEq(vaultUpgraded.ETH_BURNT_PERCENTAGE(), 20);

        // Verify the proxy still holds its balance (should be unchanged)
        assertGt(address(proxy).balance, 0); // From explorer, ~198 ETH
    }

    function getAdminAddress(address proxy) internal view returns (address) {
        address CHEATCODE_ADDRESS = 0x7109709ECfa91a80626fF3989D68f67F5b1DD12D;
        Vm vm = Vm(CHEATCODE_ADDRESS);

        bytes32 adminSlot = vm.load(proxy, ADMIN_SLOT);
        return address(uint160(uint256(adminSlot)));
    }

    function getImplementationAddress(address proxy) internal view returns (address) {
        address CHEATCODE_ADDRESS = 0x7109709ECfa91a80626fF3989D68f67F5b1DD12D;
        Vm vm = Vm(CHEATCODE_ADDRESS);

        bytes32 implementationSlot = vm.load(proxy, IMPLEMENTATION_SLOT);
        return address(uint160(uint256(implementationSlot)));
    }
}
```

Recommended Mitigation: Remove the `RollupRevenueVault::initialize`. Only `RollupRevenueVault::initializeRolesAndStorageVariables` is required to reinitialize the values for the upgrade.

Linea: Fixed at PR 1604

Cyfrin: Verified. The initialize function has been removed.

7.2.3 emiting `EthReceived` event when `RollupRevenueVault` receives 0 eth

Description: `RollupRevenueVault::receive` && `RollupRevenueVault:fallback` functions emit the `EthReceived` event whenever they are called, regardless of whether there was actually any native sent to the contract.

Especially for the `fallback()`, the event will be emitted whenever the function catches a call that doesn't match any of the functions specified on the ABI.

Recommended Mitigation: Consider skipping the event emission when no native is received on the `fallback` or `receive` functions.

Linea: Fixed at PR 1604

Cyfrin: Verified. Both `receive()` and `fallback()` functions revert if `msg.value` is 0.

7.2.4 Lack of validation to prevent receiving less LINEA tokens for the swap than the expected `minAmountOut`

Description: There is no validation on the `V3DexSwap::swap` to prevent the dex swapper from receiving fewer tokens than the specified ‘minAmountOut‘. It is true that most routers indeed enforce the received `amountOut` to be at least `minAmountOut`. However, full reliance on the router performing this validation poses a potential problem in case the dex swapper is updated to work with a router that does not perform this check.

Recommended Mitigation: Consider validating that the received LINEA tokens for the swap are at least the expected `minAmountOut`.

Linea: Fixed at PR 1604

Cyfrin: Verified. Added a check to verify the caller received at least the specified `_minLineaOut` of LINEA token.

7.2.5 No event emission when initializing parameters

Description: On the constructors of the `L1LineaTokenBurner` and `V3DexSwap` contracts, there are no event emissions to log the values of the parameters that were initialized. The same occurs when reinitializing the values on the `RollupRevenueVault`.

Recommended Mitigation: Consider emitting events to log the values of the initialized parameters.

Linea: Fixed at PR 1604

Cyfrin: Verified.

7.2.6 Require condition `_minLineaOut > 0` offers no protection against malicious BURNER_ROLE behaviour

Description: Function burnAndBridge in RollupRevenueVault is only callable by the BURNER_ROLE. During this call, the BURNER_ROLE can pass in arbitrary `_swapData` to call on the V3DexSwap contract. The swap() function contains the following check:

```
require(_minLineaOut > 0, ZeroMinLineaOutNotAllowed());
```

However, this check provides no protection against malicious behaviour by the BURNER_ROLE since `_minLineaOut` can be passed as 1 wei instead.

Impact: Loss of ETH is intended to be bridged to L1 as part of the burn and bridge mechanism.

Proof of Concept: **Recommended Mitigation:** It is recommended to either implement a configurable slippage percent on the swap amount that `_minLineaOut` should not exceed or consider acknowledging this risk.

Linea: Acknowledged.

Cyfrin: Acknowledged.

7.2.7 Condition `_invoiceAmount != 0` in submitInvoice() can prevent clearing existing debt

Description: Function submitInvoice() implements the following check below:

```
require(_invoiceAmount != 0, ZeroInvoiceAmount());
```

However, if the INVOICE_SUBMITTER role wants to only clear `invoiceArrears` if sufficient ETH balance becomes available, it will not be able to do so. For example:

- Assume at T1, `invoiceArrears` = 1e18 since there is not enough native token balance in the contract.
- At T2, the contract receives 1e18 native token balance, which can be used to clear the existing debt stored in `invoiceArrears`.
- However, `invoiceArrears` cannot be cleared by passing in `_invoiceAmount` as 0 due to the check in submitInvoice().

Impact: Although the INVOICE_SUBMITTER can wait until the next invoice submission, this delays payment of existing debt that could've been paid out sooner.

Proof of Concept: **Recommended Mitigation:** Consider ackowledging this behaviour or implementing either one of the following fixes:

1. Remove the `_invoiceAmount != 0` condition.
2. Implement a separate function that allows clearing invoiceArrears.

Linea: Fixed in PR 1637.

Cyfrin: Verified.

7.2.8 Burn and bridge mechanism can be delayed due to paused token bridge state

Description: Contract RollupRevenueVault provides the BURNER_ROLE with the burnAndBridge() function. In this process, Linea tokens are meant to be bridged to L1 and burned there.

For bridging, the `tokenBridge` service is used, which can revert due to it being paused

Impact: Exeuction of the burn and bridge mechanism can be DOSed.

Proof of Concept: **Recommended Mitigation:** Consider acknowledging the risk here and ensure appropriate measures are taken to handle such a scenario.

Linea: Acknowledged. This is acceptable, as we would then pause our burn job.

Cyfrin: Verified.

7.2.9 Function updateInvoiceArrears() updates `lastInvoiceDate` even when `invoiceArrears` remains unchanged

Description: Function updateInvoiceArrears() allows the DEFAULT_ADMIN_ROLE to update the `invoiceArrears` variable as well as `lastInvoiceDate` accordingly. However, even if the `invoiceArrears` variable remains unchanged, `lastInvoiceDate` can still be updated to a different timestamp.

```
function updateInvoiceArrears(
    uint256 _newInvoiceArrears,
    uint256 _lastInvoiceDate
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_lastInvoiceDate >= lastInvoiceDate, InvoiceDateTooOld());

    invoiceArrears = _newInvoiceArrears;
    lastInvoiceDate = _lastInvoiceDate;

    emit InvoiceArrearsUpdated(_newInvoiceArrears, _lastInvoiceDate);
  }
```

Impact: Variable `lastInvoiceDate` is updated even when `invoiceArrears` is not updated.

Proof of Concept: Recommended Mitigation: Consider disallowing this behaviour by implementing a check to ensure `_newInvoiceArrears` is not equal to `invoiceArrears`. Alternatively, if such behaviour is intended, consider renaming the function to `updateInvoiceArrearsAndLastInvoiceDate`.

Linea: Acknowledged, this accounts for various flexible situations and ways we can correct slow infrastructure billing updates.

Cyfrin: Acknowledged.

7.2.10 Function updateInvoiceArrears() does not emit old values

Description: Function updateInvoiceArrears() should consider emitting previous `invoiceArrears` and `lastInvoiceDate` values to maintain consistency with other setter functions updateL1LineaTokenBurner, updateDex and updateInvoicePaymentReceiver that emit both old and new values in their events.

```
function updateInvoiceArrears(
    uint256 _newInvoiceArrears,
    uint256 _lastInvoiceDate
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
    require(_lastInvoiceDate >= lastInvoiceDate, InvoiceDateTooOld());

    invoiceArrears = _newInvoiceArrears;
    lastInvoiceDate = _lastInvoiceDate;

    emit InvoiceArrearsUpdated(_newInvoiceArrears, _lastInvoiceDate);
  }
```

Impact: Proof of Concept:

Recommended Mitigation: Emit old `invoiceArrears` and `lastInvoiceDate` values in event `InvoiceArrearsUpdated`.

Linea: Fixed in commit 6c65701

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Unnecessary ETH to WETH conversion during swap

Description: In the V3DexSwap contract, ETH is converted to WETH before processing the swap through the router as seen in the snippet below.

```
IWETH9(WETH_TOKEN).deposit{ value: msg.value }();
IWETH9(WETH_TOKEN).approve(ROUTER, msg.value);
```

However, this is not required since the router supports direct ETH to Linea swaps. As we can observe below, the exactInputSingle function is marked as payable to allow direct ETH transfers when the function is called. In the router's execution path when the uniswapV3SwapCallback() calls the pay() function, it would process the router's contract balance if the token is WETH.

```
File: SwapRouter.sol

/// @inheritdoc ISwapRouter
    function exactInputSingle(
        ExactInputSingleParams calldata params
    ) external payable override checkDeadline(params.deadline) returns (uint256 amountOut) {


File: PeripheryPayments.sol

/// @param token The token to pay
    /// @param payer The entity that must pay
    /// @param recipient The entity that will receive payment
    /// @param value The amount to pay
    function pay(
        address token,
        address payer,
        address recipient,
        uint256 value
    ) internal {
        if (token == WETH9 && address(this).balance >= value) {
            // pay with WETH9
            IWETH9(WETH9).deposit{value: value}(); // wrap only what is needed to pay
            IWETH9(WETH9).transfer(recipient, value);
        } else if (payer == address(this)) {
            // pay with tokens already in the contract (for the exact input multihop case)
            TransferHelper.safeTransfer(token, recipient, value);
        } else {
            // pull payment
            TransferHelper.safeTransferFrom(token, payer, recipient, value);
        }
    }
```

Impact: This does not pose a risk, however, it will cost unnecessary gas during swaps.

Proof of Concept: **Recommended Mitigation:** Consider using ETH directly during swaps instead of converting to WETH.

Linea: Fixed at commit a0b875 and dab9ebfd

Cyfrin: Verified. Now there are two versions of the DexSwap, one to directly swap ETH for Linea, and another to swap WETH for Linea. Both have been validated.

7.3.2 Redundant call to sinc LINEA token supply on `L1LineaTokenBurner`

Description: `L1LineaTokenBurner::claimMessageWithProof` is in charge of completing the bridge and burn operation of the `RollupRevenueVault` deployed on the L2.

`L1LineaTokenBurner` claims the message on the L1, receives the bridged tokens, and burns them. It can also burn any tokens that are already on the contract.

The redundancy is in calling `LINEA_TOKEN::syncTotalSupplyToL2` each time `L1LineaTokenBurner::claimMessageWithProof` is called, regardless of how much time has passed since the last burn, or how many LINEA tokens were burnt.

Given that `LINEA_TOKEN::syncTotalSupplyToL2` can be called by anyone at any time, and the function uses the current total supply, `L1LineaTokenBurner` can be optimized not to sync the L2 supply on each call.

Recommended Mitigation: Consider removing the call to `LINEA_TOKEN::syncTotalSupplyToL2` on the `L1LineaTokenBurner::claimMessageWithProof`; instead, explore alternatives to bundle multiple burns of the LINEA token into a single call to `LINEA_TOKEN::syncTotalSupplyToL2`.

- Define a criterion that determines when the L2 supply should be synced. It can occur after a certain amount of LINEA tokens have been burned, or after a specified time period has passed.

Linea: Fixed in commit 43ed33

Cyfrin: Verified.