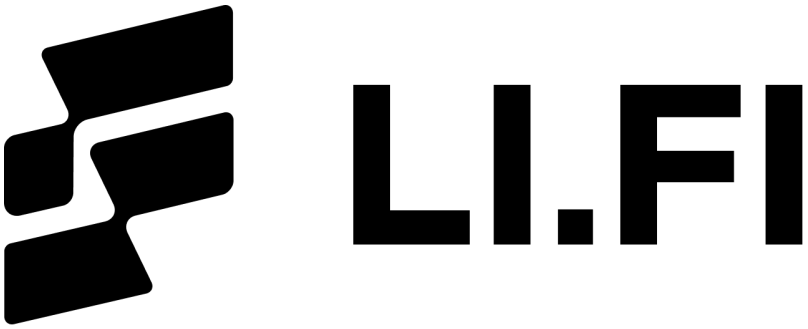

LI.FI Security Review

EcoFacet.sol(v1.0.0), IEcoPortal(v1.0.0)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

October 10, 2025

Contents

1 About Researcher

Sujith Somraaj is a distinguished security researcher and protocol engineer with over eight years of comprehensive experience in the Web3 ecosystem.

In addition to working as a Lead Security Researcher at Spearbit, Sujith is also the security researcher and advisor for leading bridge protocol LI.FI and also is a former founding engineer and current CISO at Superform, a yield aggregator with over $170M in TVL.

Sujith has experience working with protocols / funds including Coinbase, Layerzero, Edge Capital, Berachain, Optimism, Ondo, Sonic, Monad, Blast, ZkSync, Decent, Drips, SuperSushi Samurai, DistrictOne, Omni-X, Centrifuge, Superform-V2, Tea.xyz, Paintswap, Bitcorn, Sweep n' Flip, Byzantine Finance, Variational Finance, Satsbridge, Rova, Horizen, Earthfast and Angles

Learn more about Sujith on sujithsomraaj.xyz or on cantina.xyz

2 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of that given smart contract(s) or blockchain software. i.e., the evaluation result does not guarantee against a hack (or) the non existence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, I always recommend proceeding with several audits and a public bug bounty program to ensure the security of smart contract(s). Lastly, the security audit is not an investment advice.

This review is done independently by the reviewer and is not entitled to any of the security agencies the researcher worked / may work with.

3 Scope

- src/Facets/EcoFacet.sol(v1.0.0)
- src/Interfaces/IEcoPortal.sol(v1.0.1)

4 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

4.1 Impact

High leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.

Medium global losses <10% or losses to only a subset of users, but still unacceptable.

Low losses will be annoying but bearable — applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

4.2 Likelihood

High almost certain to happen, easy to perform, or not easy but highly incentivized

Medium only conditionally possible or incentivized, but still relatively likely

Low requires stars to align, or little-to-no incentive

4.3 Action required for severity levels

Critical Must fix as soon as possible (if already deployed)

High Must fix (before deployment if not already deployed)

Medium Should fix

Low Could fix

5 Executive Summary

Over the course of 15 hours in total, LI.FI engaged with the researcher to audit the contracts described in section 3 of this document ("scope").

In this period of time a total of 10 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	8d0e66a
Audit Timeline	October 05, 2025 - October 10, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	4
Gas Optimizations	2
Informational	4
Total Issues	**10**

6 Findings

6.1 Low Risk

6.1.1 Missing validation allows bypassing receiver address validation for evm chains

Context: EcoFacet.sol#L240

Description: In the `_validateEcoData()` function, when `receiver == NON_EVM_ADDRESS` is set for a non-Solana destination, the code only validates that encodedRoute exists but does not reject the invalid configuration:

```
if (receiver == NON_EVM_ADDRESS) {
    if (_ecoData.nonEVMReceiver.length == 0) revert InvalidReceiver();

    if (isSolanaDestination) {
        // Solana-specific validation
        if (_ecoData.solanaATA == bytes32(0)) revert InvalidConfig();
        if (_ecoData.encodedRoute.length != 319)
            revert InvalidReceiver();
        _validateSolanaReceiver(_ecoData);
    } else {
        // Does not reject invalid usage of NON_EVM_ADDRESS!
        if (_ecoData.encodedRoute.length == 0) revert InvalidConfig();
    }
}
```

According to the contract logic and comments:

- Solana requires NON_EVM_ADDRESS (truly non-EVM chain)
- TRON uses regular EVM receiver addresses (EVM-compatible)
- Other EVM chains use regular EVM receiver addresses

However, if a user incorrectly sets receiver = NON_EVM_ADDRESS for TRON or any other EVM-compatible chain, the validation passes as long as encodedRoute is provided.

Recommendation: Add an explicit revert in the else block to reject non-EVM addresses for non-Solana chains:

```
if (receiver == NON_EVM_ADDRESS) {
    if(!isSolanaDestination) revert InvalidConfig();

    if (_ecoData.nonEVMReceiver.length == 0) revert InvalidReceiver();
    if (_ecoData.solanaATA == bytes32(0)) revert InvalidConfig();
    if (_ecoData.encodedRoute.length != 319)
            revert InvalidReceiver();
     _validateSolanaReceiver(_ecoData);
}
```

LI.FI: Fixed in ca999e8

Researcher: Verified fix

6.1.2 Excessive native tokens are not refunded and can be permanently locked

Context: EcoFacet.sol#L123, EcoFacet.sol#L89

Description: Both `startBridgeTokensViaEco()` and `swapAndStartBridgeTokensViaEco()` functions are marked as payable and include the noNativeAsset(_bridgeData) modifier, which ensures that the bridging asset is not native (i.e., only ERC20 tokens are bridged).

However, there are issues with how native tokens sent to these functions are handled:

1. `startBridgeTokensViaEco()` accepts but never uses native tokens. Since no swaps occur and no native tokens are forwarded to the portal, any native tokens sent to this function will be permanently locked in the contract with no way to retrieve them.
2. `swapAndStartBridgeTokensViaEco()` may not refund excess native tokens. While this function performs swaps that might consume native tokens for gas or swap fees, any excess native tokens sent beyond what's needed for the swaps will remain in the contract and not be refunded to the user. This can result in users losing funds if they send more ETH than required.

Recommendation:

1. For `startBridgeTokensViaEco()`:

Since this function performs no swaps and the Eco portal is called with value: 0, native tokens serve no purpose. Make the function non-payable to prevent accidental loss of funds:

2. For `swapAndStartBridgeTokensViaEco()`:

Add the `refundExcessNative()` modifier to refund an excessive native tokens sent.

LI.FI: Fixed in 8ee81f6

Researcher: Verified fix.

6.1.3 Missing lower bound validation for solana address length

Context: EcoFacet.sol#L283

Description: In the `_validateSolanaReceiver()` function, the validation logic fails to enforce the minimum length requirement for Solana addresses as documented in the code comments:

```
function _validateSolanaReceiver(EcoData calldata _ecoData) private pure {
    // Validate the nonEVMReceiver length for Solana addresses
    // Solana addresses are base58-encoded and should be between 32-44 characters
    if (
        _ecoData.nonEVMReceiver.length == 0 ||
        _ecoData.nonEVMReceiver.length > 44
    ) {
        revert InvalidReceiver();
    }
    // ... rest of validation
}
```

The comment explicitly states that Solana addresses "should be between 32-44 characters," but the code only validates:

- Zero-length check (redundant, as discussed separately)
- Upper bound: > 44 characters

Missing validation: Lower bound check for < 32 characters.

This means a user could provide a **nonEVMReceiver** with a length between 1 and 31 characters (e.g., 10 characters), and it would pass validation despite being an invalid Solana address format.

Recommendation: Add the lower bound validation to ensure the address length is within the valid range:

```
function _validateSolanaReceiver(EcoData calldata _ecoData) private pure {
    // Validate the nonEVMReceiver length for Solana addresses
    // Solana addresses are base58-encoded and should be between 32-44 characters
    if (
        _ecoData.nonEVMReceiver.length < 32 ||
        _ecoData.nonEVMReceiver.length > 44
    ) {
        revert InvalidReceiver();
    }

    // Extract the Solana recipient address from a Borsh-encoded Route struct
    // ... rest of validation
}
```

LI.FI: Fixed in 94b00df

Researcher: Verified fix.

6.1.4 Misleading BridgeToNonEVMChain event emission for evm chains

Context: EcoFacet.sol#L213

Description: In the `_startBridge()` function, the event emission check is misleading because it only checks if nonEVMReceiver has data:

```
if (_ecoData.nonEVMReceiver.length > 0) {
    emit BridgeToNonEVMChain(
        _bridgeData.transactionId,
        _bridgeData.destinationChainId,
        _ecoData.nonEVMReceiver
    );
}
```

The problem is that `_validateEcoData()` does not enforce that nonEVMReceiver must be empty when bridging to regular EVM chains with an EVM receiver address. The validation only checks:

If receiver == NON_EVM_ADDRESS, then nonEVMReceiver.length must be > 0 If receiver != NON_EVM_ADDRESS, it validates the receiver address but never checks that nonEVMReceiver should be empty

This means a user could bridge to a regular EVM chain (e.g., Ethereum, Polygon) with a normal EVM receiver address, but still populate nonEVMReceiver with arbitrary data.

The current code would then emit the BridgeToNonEVMChain event even though the destination is actually an EVM chain, creating misleading event logs.

Recommendation: The event should only be emitted when actually bridging to a non-EVM address. Consider replacing the length checks to chain id based checks:

```
- if (_ecoData.nonEVMReceiver.length > 0) {
+ if (_bridgeData.destinationChainId == LIFI_CHAIN_ID_SOLANA) {
```

LI.FI: Fixed in e3249ef

Researcher: Verified fix.

6.2 Gas Optimization

6.2.1 Redundant zero-Length check for nonEVMReceiver in solana validation

Context: EcoFacet.sol#L239

Description: In the _validateEcoData function, there is a redundant check for nonEVMReceiver.length == 0 that occurs in two places:

First check in `_validateEcoData()`:

```
if (receiver == NON_EVM_ADDRESS) {
    if (_ecoData.nonEVMReceiver.length == 0) revert InvalidReceiver();

    if (isSolanaDestination) {
        if (_ecoData.solanaATA == bytes32(0)) revert InvalidConfig();
        if (_ecoData.encodedRoute.length != 319)
            revert InvalidReceiver();
        _validateSolanaReceiver(_ecoData);  // Called here
    }
    // ...
}
```

Second check in `_validateSolanaReceiver()`:

```
function _validateSolanaReceiver(EcoData calldata _ecoData) private pure {
    // Validate the nonEVMReceiver length for Solana addresses
    // Solana addresses are base58-encoded and should be between 32-44 characters
    if (
        _ecoData.nonEVMReceiver.length == 0 ||  // Redundant check
        _ecoData.nonEVMReceiver.length > 44
    ) {
        revert InvalidReceiver();
    }
    // ... rest of validation
}
```

The `_validateSolanaReceiver()` function is only called after already verifying that nonEVMReceiver.length > 0 in `_validateEcoData()`. This makes the zero-length check inside _validateSolanaReceiver redundant and wasteful in terms of gas consumption.

Recommendation: Remove the redundant zero-length check from _validateSolanaReceiver and only validate the upper bound:

```
function _validateSolanaReceiver(EcoData calldata _ecoData) private pure {
    // Validate the nonEVMReceiver length for Solana addresses
    // Solana addresses are base58-encoded and should be between 32-44 characters
    // Note: Zero-length is already validated in _validateEcoData before calling this function
    if (_ecoData.nonEVMReceiver.length > 44) {
        revert InvalidReceiver();
    }

    // Extract the Solana recipient address from a Borsh-encoded Route struct
    // ... rest of validation
}
```

LI.FI: Fixed in 94b00df

Researcher: Verified fix.

6.2.2 Unnecessary variable assignment in `startBridgeTokensViaEco()` increases gas cost

Context: EcoFacet.sol#L104

Description: In the `startBridgeTokensViaEco()` function, an intermediate variable depositAmount is declared and assigned a value that is only used once:

```
function startBridgeTokensViaEco(
    ILiFi.BridgeData memory _bridgeData,
    EcoData calldata _ecoData
)
    external
    payable
    nonReentrant
    validateBridgeData(_bridgeData)
    doesNotContainSourceSwaps(_bridgeData)
    doesNotContainDestinationCalls(_bridgeData)
    noNativeAsset(_bridgeData)
{
    _validateEcoData(_bridgeData, _ecoData);

    // Deposit includes the solver reward for ERC20
    uint256 depositAmount = _bridgeData.minAmount + _ecoData.solverReward;

    LibAsset.depositAsset(_bridgeData.sendingAssetId, depositAmount);

    _startBridge(_bridgeData, _ecoData);
}
```

The depositAmount variable stores the sum of _bridgeData.minAmount and _ecoData.solverReward, but this value is immediately consumed in the next line and never referenced again. This creates unnecessary stack operations (MSTORE/MLOAD) that increase gas consumption without providing any benefit in terms of code readability or reusability.

Recommendation: Consider removing the intermediate variable and pass the expression directly to the function call, which saves approximately 100 GAS units.

LI.FI: Fixed in 50e26d5

Researcher: Verified fix.

6.3 Informational

6.3.1 Missing critical parameter validation in `_buildReward()` function

Context: EcoFacet.sol#L169-L170

Description: The `_buildReward()` function constructs the IEcoPortal.Reward struct without validating critical parameters, prover and rewardDeadline from _ecoData.

Recommendation: Add validation for these critical parameters in the _validateEcoData function or at the beginning of _buildReward:

```
function _validateEcoData(
    ILiFi.BridgeData memory _bridgeData,
    EcoData calldata _ecoData
) private pure {
    // Validate prover address
    if (_ecoData.prover == address(0)) revert InvalidConfig();

    // Validate reward deadline is in the future
    if (_ecoData.rewardDeadline == 0 || _ecoData.rewardDeadline <= block.timestamp) {
        revert InvalidConfig();
    }

    address receiver = _bridgeData.receiver;
    // ... rest of existing validation
}
```

LI.FI: Fixed in 0c8f825

Researcher: Verified fix.

6.3.2 Redundant explicit zero value in portal call

Context: EcoFacet.sol#L206

Description: In the `_startBridge()` function, the call to PORTAL.publishAndFund explicitly specifies { value: 0 }:

```
PORTAL.publishAndFund{ value: 0 }(
    destination,
    _ecoData.encodedRoute,
    reward,
    false
);
```

The explicit { value: 0 } specification is redundant because the default value for any external call is 0 when not specified. This adds unnecessary bytecode and marginally increases gas costs without providing any functional benefit or improving code clarity.

Recommendation: Remove the explicit { value: 0 } specification to simplify the code and slightly reduce gas costs.

LI.FI: Fixed in 82fbf2b

Researcher: Verified fix.

6.3.3 Redundant `receiverAddress` in EcoData

Context: EcoFacet.sol#L64

Description: The `receiverAddress` parameter in `EcoData` is redundant and should always be equal to the `receiver` in `BridgeData`. The contract only uses the `EcoData.receiverAddress` for validation, not for any actual bridge execution, which could be safely removed.

Recommendation: Consider removing the `receiverAddress` parameter in `EcoData` to save additional gas costs.

LI.FI: Fixed in 967d424a

Researcher: Verified fix.

6.3.4 Replace hardcoded values with named constants

Context: EcoFacet.sol#L210, EcoFacet.sol#L171, EcoFacet.sol#L23

Description: The `EcoFacet.sol` contract uses hardcoded literals in critical calls instead of named constants, which obscures intent and makes future changes error-prone:

- In `_buildReward()`, nativeAmount is hardcoded to 0, silently enforcing "no native reward"
- In `_startBridge()`, publishAndFund{ value: 0 }(..., false) the intent is published with a hardcoded `false` flag for the allowPartial parameter
- In `_validateEcoData()`, the minimum length value is hardcoded to 319

Recommendation: Define descriptive, file-scoped constants (or configurable parameters) and replace magic literals with them.

LI.FI: Fixed in dd905c0

Researcher: Verified fix.