LI.FI Security Review

UnitFacet.sol(v1.0.0), LiFiData(v1.0.1), LibAsset(v2.1.3)

Security Researcher

Sujith Somraaj (somraajsujith@gmail.com)

Report prepared by: Sujith Somraaj

October 7, 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 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/UnitFacet.sol(v1.0.0)
- src/Helpers/LiFiData.sol(v1.0.1)
- src/Libraries/LibAsset.sol(v2.1.3)

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 9 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 8 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit	626dd4a9
Audit Timeline	October 06, 2025
Methods	Manual Review
Documentation	High
Test Coverage	Medium-High

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Gas Optimizations	0
Informational	5
Total Issues	**8**

6 Findings

6.1 Low Risk

6.1.1 Centralized trust in backend-provided deposit address

Context: Global

Description: The `UnitFacet.sol` contract unconditionally trusts the depositAddress provided by the backend signer without any on-chain validation. A compromised backend signer, a malicious insider, or a backend vulnerability could result in an attacker-controlled address being used instead of a legitimate Unit Protocol deposit address, leading to the theft of all bridged funds.

Users have no way to verify on-chain that their funds are being sent to a valid Unit Protocol address.

Recommendation: Consider using MPC wallets for off-chain signer security to sign deposit addresses.

In the future, consider redesigning the system to include whitelisting user deposit addresses before transactions, implementing a cooldown period, and adding a challenger mechanism.

LI.FI: Acknowledged. For more context, the Unit already provides MPC-based signing through guardians. However, they use ECDSA with the P-256 curve, which cannot be efficiently verified on-chain using standard Solidity operations, such as ecrecover. To address this, during calldata generation, the backend first verifies the guardian's P-256 signature off-chain, then re-signs the verified message using an EIP-712 signature. Neverthless, the recommendation is passed to the backend team

Researcher: Acknowledged.

6.1.2 No validation that swap output is native token

Context: UnitFacet.sol#L110

Description: The `swapAndStartBridgeTokensViaUnit()` function performs token swaps but does not validate that the final output token is the native asset (ETH). After swapping, the contract unconditionally attempts to transfer native ETH to the deposit address using `LibAsset.transferNativeAsset()`.

Recommendation: Add explicit validation that the final swap output is the native token:

```
function swapAndStartBridgeTokensViaUnit(
    ILiFi.BridgeData memory _bridgeData,
    LibSwap.SwapData[] calldata _swapData,
    UnitData calldata _unitData
)
    external
    payable
    nonReentrant
    refundExcessNative(payable(msg.sender))
    containsSourceSwaps(_bridgeData)
    doesNotContainDestinationCalls(_bridgeData)
    validateBridgeData(_bridgeData)
    onlyAllowSourceToken(_bridgeData, LibAsset.NULL_ADDRESS)
    onlyAllowDestinationChain(_bridgeData, LIFI_CHAIN_ID_HYPERCORE)
{
    _verifySignature(_bridgeData, _unitData);

+  if (_swapData[_swapData.length - 1].receivingAssetId; != LibAsset.NULL_ADDRESS) {
+      revert InvalidSwapOutputToken(finalReceivingAsset);
+   }

    uint256 balanceBefore = address(this).balance;

    _bridgeData.minAmount = _depositAndSwap(
        _bridgeData.transactionId,
        _bridgeData.minAmount,
        _swapData,
        payable(msg.sender)
    );

    _startBridge(_bridgeData, _unitData);
}
```

LI.FI: Acknowledged. We save some gas here. If the swap by accident yields the wrong (non-native) token, then the tx should fail anyway due to insufficient native balance

Researcher: Acknowledged.

6.1.3 Signature verification uses pre-swap amount while transfer uses post-swap amount

Context: UnitFacet.sol#L110

Description: The `swapAndStartBridgeTokensViaUnit()` function verifies the backend signature using the original minAmount parameter, but then executes the bridge transfer using the post-swap minAmount. This discrepancy means that the actual bridged amount differs from what the backend authorized in the signature, thereby breaking the integrity of the signed authorization.

```
function swapAndStartBridgeTokensViaUnit(
    ILiFi.BridgeData memory _bridgeData,  // minAmount = pre swap amount
    LibSwap.SwapData[] calldata _swapData,
    UnitData calldata _unitData
)
    external
    payable
    nonReentrant
    // ... modifiers ...
{
    // STEP 1: Verify signature with ORIGINAL amount
    _verifySignature(_bridgeData, _unitData);

    // STEP 2: Execute swap, UPDATE minAmount with output
    _bridgeData.minAmount = _depositAndSwap(
        _bridgeData.transactionId,
        _bridgeData.minAmount,  pre-swap amount
        _swapData,
        payable(msg.sender)
    );  // Output: Could be either favorable / unfavorable

    // STEP 3: Bridge with NEW post-swap amount
    _startBridge(_bridgeData, _unitData);
}
```

Recommendation: Consider modifying the function to verify the signature with the actual post-swap amount with a slippage tolerance.

LI.FI: Acknowledged. The signature is meant to authorize the swap initiation, not the post-swap result

Researcher: Acknowledged.

6.2 Informational

6.2.1 Missing strict chain ID validations in `_startBridge()` function

Context: UnitFacet.sol#L138

Description: The `_startBridge()` function only enforces minimum amount requirements for Ethereum (chain ID 1) and Plasma (chain ID 9745). On any other chain, the function accepts transfers of any amount, including amounts as low as one wei. This allows users to initiate economically unviable bridges that may fail, result in stuck funds, or waste gas on transactions that the Unit Protocol cannot process.

Recommendation: Consider modifying the function to only allow explicitly supported chains:

```
+ /// @notice Error for unsupported chains
+ error UnsupportedChain();

function _startBridge(
    ILiFi.BridgeData memory _bridgeData,
    UnitData calldata _unitData
) internal {
    if (block.chainid == CHAIN_ID_ETHEREUM) {
        if (_bridgeData.minAmount < 0.05 ether) {
            revert InvalidAmount();
        }
    } else if (block.chainid == CHAIN_ID_PLASMA) {
        if (_bridgeData.minAmount < 15 ether) {
            revert InvalidAmount();
        }
+    } else {
+        // Reject all other chains
+        revert UnsupportedChain();
+    }

    // ... rest of function ...
}
```

LI.FI: Fixed in 40b6c94e

Researcher: Verified fix.

6.2.2 Missing depositAddress validation

Context: UnitFacet.sol#L42

Description: The `UnitFacet.sol` contract does not validate that _unitData.depositAddress is non-zero before transferring native assets. However, the `transferNativeAsset()` function in LibAsset will revert if the destination address to transfer tokens is address(0) protecting users against permanent fund loss.

Recommendation: Add validation in `_startBridge()` before the transfer to fail quick:

```
function _startBridge(
    ILiFi.BridgeData memory _bridgeData,
    UnitData calldata _unitData
) internal {
+    // Add this validation at the start of the function
+    if (_unitData.depositAddress == address(0)) {
+        revert InvalidConfig();
+    }

    // ... rest of the function ...
}
```

LI.FI: Acknowledged, it will revert anyway, and here we can save gas not having this check

Researcher: Acknowledged.

6.2.3 Deadline check after expensive operations

Context: UnitFacet.sol#L141

Description: The signature deadline check is done after massively expensive operations to validate the signature in the _startBridge() function.

However, this check can be moved to the `_verifySignature()` function to save gas and revert earlier than later.

Recommendation: Consider checking the deadline earlier in the flow:

```
function _verifySignature(
  ILiFi.BridgeData memory _bridgeData,
  UnitData calldata _unitData
) internal {
+  if (block.timestamp > _unitData.deadline) {
+      revert SignatureExpired();
+  }

  /// rest of the functions
}
```

LI.FI: Fixed in e1648970

Researcher: Verified fix.

6.2.4 Replace hardcoded values with named constants

Context: UnitFacet.sol#L130, UnitFacet.sol#L135

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

Recommendation: Consider replacing hardcoded magic numbers with named constants.

LI.FI: Fixed in 1600dcb3

Researcher: Verified fix.

6.2.5 No replay protection

Context: UnitFacet.sol#L156

Description: The `UnitFacet.sol` contract does not implement replay protection for backend-signed signatures. Once a valid signature is created with a specific transaction ID, deposit address, receiver, and deadline, it can be used multiple times until the deadline expires.

Currently, the signature verification in `_verifySignature()` only checks:

- BACKEND_SIGNER signed the signature
- The deadline has not expired

There is no mechanism (such as a nonce or signature tracking) to prevent the same signature from being used in multiple transactions.

Given the Unit Protocol's architecture, where protocol addresses are permanently tied to destination addresses:

- No fund theft is possible: If an attacker replays someone else's signature, they would be donating their own ETH to the intended receiver
- User can replay their own signature: This allows them to bridge multiple times, which they could do anyway by requesting a new signature

Recommendation: Consider adding replay protection based on transactionID if this behavior is not desirable.

LI.FI: Fixed in 32a032515

Researcher: Verified fix.