LI.FI Security Review

DeBridgeDlnFacet(v1.0.0)

Independent Review By:

Sujith Somraaj (somraajsujith@gmail.com)

December 5, 2024

Contents

1 About Researcher

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

In addition to working as an external auditor/security researcher with LI.FI, Sujith is a protocol engineer and security researcher at Superform and Spearbit.

Learn more about Sujith on sujithsomraaj.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/DeBridgeDlnFacet.sol(v1.0.0)

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

This review intends to audit only the new callback functions (ramsesV2SwapCallback, xeiV3SwapCallback, dragonswapV2SwapCallback, agniSwapCallback, fusionXV3SwapCallback, vvsV3SwapCallback, supV3SwapCallback, zebraV3SwapCallback) added, not the entire file under audit (as its an audited fork from sushiswap).

In this period of time a total of 8 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit Hashes	d72cb05510.....a2ef85d06d
Type of Project	
Audit Timeline	November 21, 2024 - November 23, 2024
Methods	Manual Review

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	3
Gas Optimizations	0
Informational	3
Total Issues	**8**

6 Findings

6.1 Medium Risk

6.1.1 Redundant receiver address checks using `validateBridgeData` modifier

Context: DeBridgeDlnFacet.sol#L108, DeBridgeDlnFacet.sol#L138

Description: The functions **startBridgeTokensViaDeBridgeDln** and **swapAndStartBridgeTokensViaDeBridgeDln** bridge tokens using the deBridgeDln facet. These two functions depend on the **validateBridgeData** modifier to ensure the correctness of user input.

The **validateBridgeData** modifier validates the following:

- if **_bridgeData.receiver** is not address(0)
- if **_bridgeData.minAmount** is not zero
- if **_bridgeData.destinationChainId** is not equal to the **block.chainid**

However, some of these checks need to be revised. The **receiver** address used in the bridging comes from the **_deBridgeData**, not the **_bridgeData**, allowing funds to be bridged to a zero address.

Setting an empty receiver not only allows for the funds to be bridged to a wrong invalid address but also makes it non-cancellable.

Recommendation: Consider implementing additional validation checks to ensure the **_deBridgeData** is also validated and sanity-checked to avoid user errors.

LI.FI: Fixed in f561360dde2629e4ff624a87784bc01989b38bc6

Researcher: Verified fix.

6.1.2 Non-EOA receiver can never initiate the cancellation process

Context: DeBridgeDlnFacet.sol#L173

Description: Per deBridge's documentation, to initiate an order cancellation process, the **orderAuthorityAddressDst** has to call the **sendEvmOrderCancel** function on the destination chain's dln.

While creating an order in the source chain, the ** DeBridgeDlnFacet ** specifies the **_deBridgeData.receiver** address as the **orderAuthorityAddressDst** without validating whether it can initiate a cancellation flow.

Hence, bridging funds to a smart contract on the destination chain, without the ability to call **sendEvmOrderCancel**, will permanently lock the funds unless market conditions change and the order created becomes valid for a solver.

Recommendation: Consider accepting a new parameter in the **_deBridgeData** parameter and allow the user to specify the **orderAuthorityAddressDst** as follows:

```
struct DeBridgeDlnData {
  bytes receivingAssetId;
  bytes receiver;
+ bytes orderAuthorityDst;
  uint256 minAmountOut;
}
....
function _startBridge(
    ILiFi.BridgeData memory _bridgeData,
    DeBridgeDlnData calldata _deBridgeData,
    uint256 _fee
) internal {
IDlnSource.OrderCreation memory orderCreation = IDlnSource
  .OrderCreation({
    ....
+   orderAuthorityAddressDst: _deBridgeData.orderAuthorityDst,
    ....
  })
 .....
}
```

LI.FI: Fixed in 8bcb3927ebab0780cb54ba6306f7ecca2a09ff55

Researcher: Verified fix.

6.2 Low Risk

6.2.1 Set `allowedCancelBeneficiarySrc` to be `msg.sender`

Context: DeBridgeDlnFacet.sol#L176

Description: The `OrderCreation` struct in deBridge includes an **allowedCancelBeneficiarySrc** variable, currently set to empty bytes in the `DeBridgeDlnFacet.sol` contract.

When canceling an order, the **orderAuthorityAddressDst** can specify a refund address on the source chain only if **allowedCancelBeneficiarySrc** is not set. This makes refund accuracy dependent on the **orderAuthorityAddressDst** correctly canceling the order and providing a valid refund receiver, which can be cumbersome.

Setting **allowedCancelBeneficiarySrc** to `msg.sender` would ensure that refunds are always sent to the address that initiated the bridge transaction, regardless of any errors from the **orderAuthorityAddressDst**.

For more information regarding the cancellation flow of deBridge, please refer to their documentation.

Recommendation: Consider setting the **allowedCancelBeneficiarySrc** to `msg.sender` while constructing the order inside the **_startBridge** function as follows:

```
function _startBridge(
        ILiFi.BridgeData memory _bridgeData,
        DeBridgeDlnData calldata _deBridgeData,
        uint256 _fee
    ) internal {
        IDlnSource.OrderCreation memory orderCreation = IDlnSource
            .OrderCreation({
                giveTokenAddress: _bridgeData.sendingAssetId,
                giveAmount: _bridgeData.minAmount,
                takeTokenAddress: _deBridgeData.receivingAssetId,
                takeAmount: _deBridgeData.minAmountOut,
                takeChainId: getDeBridgeChainId(
                    _bridgeData.destinationChainId
                ),
                receiverDst: _deBridgeData.receiver,
                givePatchAuthoritySrc: msg.sender,
                orderAuthorityAddressDst: _deBridgeData.receiver,
                allowedTakerDst: "",
                externalCall: "",
-               allowedCancelBeneficiarySrc: ""
+               allowedCancelBeneficiarySrc: msg.sender
            });
    ....
}
```

LI.FI: Fixed in 92d87722236f0d58992b7baac378b42501087f8c

Researcher: Verified fix.

6.2.2 Use deBridge referral code while creating orders to earn deBridge points

Context: DeBridgeDlnFacet.sol#L199, DeBridgeDlnFacet.sol#L191

Description: The **createOrder** function from `DlnSource.sol` accepts four parameters in total. The **_affiliateFee** and **_referralCode** parameters are used by integrating projects to earn **deBridge** points for routing users through them.

```
function createOrder(
   DlnOrderLib.OrderCreation calldata _orderCreation,
   bytes calldata _affiliateFee,
   uint32 _referralCode,
   bytes calldata _permitEnvelope
)
   external
   payable
   returns (bytes32);
```

However, these two parameters are unused in the current **DeBridgeDlnFacet**, leading to the protocol's loss of deBridge points.

Recommendation: Consider using the **_referralCode** while forwarding the order to **DlnSource**, to earn deBridge points.

LI.FI: Fixed in 9f76e21db193af53defe29688cc711385452d7de, 3f1d1037a0f3d6d2839616df830a03227c1ceb88

Researcher: Verified fix.

6.2.3 `initDeBridgeDln` allows re-initialization of the facet

Context: DeBridgeDlnFacet.sol#L79

Description: The **initDeBridgeDln** function in `DeBridgeDlnFacet.sol` initializes the facet. Once the initialization is complete, the **sm.initialized** variable is set to **true**.

To prevent re-initialization, the function should check the **sm.initialized** variable. Due to the lack of this check, the facet can be initialized multiple times, which is an unexpected behavior for the protocol.

Recommendation: Consider updating the **initDeBridgeDln** function as following:

```
function initDeBridgeDln(
    ChainIdConfig[] calldata chainIdConfigs
) external {
    LibDiamond.enforceIsContractOwner();

    Storage storage sm = getStorage();
+   if(sm.initialized) revert AlreadyInitialized();
    .....
}
```

LI.FI: We're fine with this as it allows for upgrading the facet in the future

Researcher: Acknowledged. Since the function is admin-protected, it's not a big risk, but the re-initialization could be prevented, as same functionality can be achieved by other setter methods.

6.3 Informational

6.3.1 Validate zero length arrays in `initDeBridgeDln` function

Context: DeBridgeDlnFacet.sol#L79

Description: The **initDeBridgeDln** function of the `DeBridgeDlnFacet.sol` contract initializes the contract by setting the initial chain IDs. However, this function lacks sanity validations, allowing the facet to be initialized with no chain IDs.

Recommendation: Consider adding a validation to check the array length as follows:

```
function initDeBridgeDln(
  ChainIdConfig[] calldata chainIdConfigs
  ) external {
  if(chainIdConfigs.length == 0) revert();
  ....
}
```

LI.FI: Fixed in 92d87722236f0d58992b7baac378b42501087f8c

Researcher: Verified fix.

6.3.2 Code quality issue: inline documentation

Context: DeBridgeDlnFacet.sol#L240, DeBridgeDlnFacet.sol#L219

Description: The **getDeBridgeChainId** returns the deBridge chain ID for a specified blockchain's native chainId. However, the function's inline documentation states that it returns a Layer 0 chain ID, which seems confusing and inaccurate.

Similarly, the **setDeBridgeChainId** updates the deBridge chain ID for a given chain ID and annotates it as the Layer 0 chain ID.

Recommendation: Consider updating the inline documentation as follows:

```
- /// @notice Gets the Layer 0 chain ID for a given chain ID
+ /// @notice Gets the DeBridge chain ID for a given chain ID
....
- /// @return uint256 of the Layer 0 chain ID
+ /// @return uint256 of the DeBridge chain ID
```

LI.FI: Fixed in 92d87722236f0d58992b7baac378b42501087f8c

Researcher: Verified fix.

6.3.3 Code quality issue: unused import

Context: DeBridgeDlnFacet.sol#L12

Description: The `DeBridgeDlnFacet.sol` imports **InformationMismatch** and **AlreadyInitialized**, but is not used anywhere in the context. Unused imports could impact code quality.

Recommendation: Consider removing the unused imports mentioned above to improve code quality.

LI.FI: Fixed in 92d87722236f0d58992b7baac378b42501087f8c

Researcher: Verified fix.