LI.FI Security Review

Glacis Facet (v1.0.0)

Independent Review By:

Sujith Somraaj (somraajsujith@gmail.com)

February 19, 2025

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 security researcher at Spearbit and a former founding-engineer at Superform, an yield aggregator with over $100M in TVL.

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

- GlacisFacet.sol(v1.0.0)
- IGlacisAirlift.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 1 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 5 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Type of Project	
Audit Timeline	February 11, 2025
Methods	Manual Review

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Gas Optimizations	0
Informational	4
Total Issues	**5**

6 Findings

6.1 Low Risk

6.1.1 Validate `GlacisData.refundAddress` is non zero

Context: GlacisFacet.sol#L112

Description: The `_glacisData.refundAddress` parameters in `startBridgeTokensViaGlacis` and `swapAndStartBridgeTokensViaGlacis` are not validated. This address is crucial for receiving refunds from the Glacis bridge and should be validated like `bridgeData.receiver`.

Recommendation: Consider adding a sanity check to ensure `_glacisData.refundAddress` is set.

```
    function _startBridge(
        ILiFi.BridgeData memory _bridgeData,
        GlacisData calldata _glacisData
    ) internal {
        // Approve the Airlift contract to spend the required amount of tokens.
        // The `send` function assumes that the caller has already approved the token transfer,
        // ensuring that the cross-chain transaction and token transfer happen atomically.
        LibAsset.maxApproveERC20(
            IERC20(_bridgeData.sendingAssetId),
            address(airlift),
            _bridgeData.minAmount
        );

+    if(_glacisData.refundAddress == address(0)) revert("Invalid refund address");

        airlift.send{ value: _glacisData.nativeFee }(
            _bridgeData.sendingAssetId,
            _bridgeData.minAmount,
            bytes32(uint256(uint160(_bridgeData.receiver))),
            _bridgeData.destinationChainId,
            _glacisData.refundAddress
        );

        emit LiFiTransferStarted(_bridgeData);
    }
```

LI.FI: Fixed in f5cdbc279f0f15ed469650d5b9b4185c0c668547

Researcher: Verified fix

6.2 Informational

6.2.1 Validate if `msg.value` is enough to cover for `_glacisData.nativeFee`

Context: GlacisFacet.sol#L107

Description: The functions `startBridgeTokensViaGlacis` and `swapAndStartBridgeTokensViaGlacis` do not have any validations on `msg.value`, which allows users to siphon native tokens from the diamond contract (if it has any).

Malicious users can make the diamond pay for their `_glacisData.nativeFee` from its balance.

Recommendation: Consider ensuring that the `msg.value` paid by the user is enough for `_glacisData.nativeFee`

LI.FI: We prioritize gas savings over additional checks that lead to failing transactions anyway.

Researcher: Acknowledged

6.2.2 Add `noNativeAsset` modifier

Context: GlacisFacet.sol#L66, GlacisFacet.sol#L43

Description: Both bridging functions, `startBridgeTokensViaGlacis` and `swapAndStartBridgeTokensViaGlacis`, do not facilitate native asset bridging, as the Glacis bridge does not support this functionality either.

As a result, the bridging encounters issues further down the stack during the external call to the Glacis bridge contract, as indicated in the logs below:

```
    [49404] GlacisFacet::fallback{value: 100000002793980000000}(BridgeData({ transactionId:
↪  0x0000000000000000000000000000000000000000000000000000000000000000, bridge: "glacis",
↪  integrator: "", referrer: 0x0000000000000000000000000000000000000000, sendingAssetId:
↪  0x0000000000000000000000000000000000000000, receiver:
↪  0x000000000000000000000000000000abC654321, minAmount: 100000000000000000000 [1e21],
↪  destinationChainId: 10, hasSourceSwaps: false, hasDestinationCall: false }), GlacisData({
↪  refundAddress: 0x317F8d18FB16E49a958Becd0EA72f8E153d25654, nativeFee: 2793980000000 [2.793e12]
↪  }))
    [44392] TestGlacisFacet::startBridgeTokensViaGlacis{value: 100000002793980000000}(BridgeData({
↪  transactionId: 0x0000000000000000000000000000000000000000000000000000000000000000, bridge:
↪  "glacis", integrator: "", referrer: 0x0000000000000000000000000000000000000000, sendingAssetId:
↪  0x0000000000000000000000000000000000000000, receiver:
↪  0x000000000000000000000000000000abC654321, minAmount: 100000000000000000000 [1e21],
↪  destinationChainId: 10, hasSourceSwaps: false, hasDestinationCall: false }), GlacisData({
↪  refundAddress: 0x317F8d18FB16E49a958Becd0EA72f8E153d25654, nativeFee: 2793980000000 [2.793e12]
↪  })) [delegatecall]
       [10007] 0xD9E7f6f7Dc7517678127D84dBf0F0b4477De14E0::send{value:
↪  2793980000000}(0x0000000000000000000000000000000000000000, 100000000000000000000 [1e21],
↪  0x0000000000000000000000000000000000000000000000000000000abc654321, 10,
↪  0x317F8d18FB16E49a958Becd0EA72f8E153d25654)
          [5040] 0xA9f837bE490d9d34141f2ED02845692c5309a177::send{value:
↪  2793980000000}(0x0000000000000000000000000000000000000000, 100000000000000000000 [1e21],
↪  0x0000000000000000000000000000000000000000000000000000000abc654321, 10,
↪  0x317F8d18FB16E49a958Becd0EA72f8E153d25654) [delegatecall]
             ← [Revert] custom error 0xc22b5d63
          ← [Revert] custom error 0xc22b5d63
       ← [Revert] custom error 0xc22b5d63
    ← [Revert] custom error 0xc22b5d63
```

Recommendation: Consider adding the `noNativeAsset` modifier to the above function to fail fast with meaningful errors. However, this addition might increase gas costs (~40 GAS) on the usual path, which is insignificant.

LI.FI: Fixed in f9276e33393986022f90b48fd0c5a025fa9702b6

Researcher: Verified fix

6.2.3 Ensure the last token after swaps is the required bridging token in `swapAndStartBridgeTokensViaGlacis`

Context: GlacisFacet.sol#L66

Description: The `swapAndStartBridgeTokensViaGlacis` function first swaps an asset to the `_bridgeData.sendingAsset`, allowing the swapped asset to be bridged through Glacis. However, there is no validation to ensure that the swap's final output token is the required asset for bridging, which may lead to failures further down the stack without a proper error message.

Recommendation: Consider adding the following check:

```
function swapAndStartBridgeTokensViaGlacis(
        ILiFi.BridgeData memory _bridgeData,
        LibSwap.SwapData[] calldata _swapData,
        GlacisData calldata _glacisData
    )
        external
        payable
        nonReentrant
        refundExcessNative(payable(msg.sender))
        containsSourceSwaps(_bridgeData)
        doesNotContainDestinationCalls(_bridgeData)
        validateBridgeData(_bridgeData)
    {
+       if(_swapData[_swapData.length - 1].receivingAssetId != _bridgeData.sendingAssetId) revert("Asset
↪  mismatch");
        _bridgeData.minAmount = _depositAndSwap(
            _bridgeData.transactionId,
            _bridgeData.minAmount,
            _swapData,
            payable(msg.sender),
            _glacisData.nativeFee
        );
        _startBridge(_bridgeData, _glacisData);
    }
```

LI.FI: This has been reported on other facets as well. Since our contract does not hold funds, the transaction would simply fail in that case. We prioritize gas saving over unnecessary checks.

Researcher: Acknowledged

6.2.4 Unused `SafeERC20` import

Context: GlacisFacet.sol#L5

Description: The `GlacisFacet.sol` contains an unused import, `SafeERC20`. The file is imported but is not used anywhere in the code.

Recommendation: Consider removing the unused import / if required, use it accordingly.

LI.FI: Fixed in dad7806ed0c2aeba5f68c56ea63108852987dd9f

Researcher: Verified fix