LI.FI Security Review

Auditors

Gerard Persoon, Lead Security Researcher

Jonah1005, Lead Security Researcher

DefSec, Security Researcher

Blockdev, Security Researcher

Report prepared by: Pablo Misirov

April 11, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

LI.FI is a cross-chain bridge aggregation protocol that supports any-2-any swaps by aggregating bridges and connecting them to DEX aggregators.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of lifinance contracts according to the specific commit. Any modifications to the code will require a new security review.

3 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

3.1 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.

3.2 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

3.3 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

4 Executive Summary

Over the course of 5 days in total, LI.FI engaged with Spearbit to review the lifinance contracts protocol. In this period of time a total of **76** issues were found.

Summary

Project Name	LI.FI
Repository	contracts
Commit	spearbit-audit-march-2023
Type of Project	Bridge Aggregator, Bridge
Audit Timeline	March 13 - March 17
Two week fix period	March 17 - April 1
Retainer period	1

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	4	4	0
Medium Risk	8	5	3
Low Risk	18	9	9
Gas Optimizations	13	12	1
Informational	33	26	7
Total	**76**	**56**	**20**

5 Findings

5.1 High Risk

5.1.1 `Receiver` doesn't always reset allowance

Severity: High Risk

Context: Receiver.sol#L224-L297

Description: The function `_swapAndCompleteBridgeTokens()` of `Receiver` reset the approval to the executor at the end of an ERC20 transfer. However it there is insufficient gas then the approval is not reset.

This allows the executor to access any tokens (of the same type) left in the `Receiver`.

```
function _swapAndCompleteBridgeTokens(...) ... {
    ...
    if (LibAsset.isNativeAsset(assetId)) {
        ...
    } else { // case 2: ERC20 asset
        ...
        token.safeIncreaseAllowance(address(executor), amount);
        if (reserveRecoverGas && gasleft() < _recoverGas) {
            token.safeTransfer(receiver, amount);
            ...
            return; // no safeApprove 0
        }
        try executor.swapAndCompleteBridgeTokens{...} ...
        token.safeApprove(address(executor), 0);
    }
}
```

Recommendation: Only increase the allowance if sufficient gas is available, for example in the following way

```
function _swapAndCompleteBridgeTokens(...) ... {
    ...
    if (LibAsset.isNativeAsset(assetId)) {
        ...
    } else { // case 2: ERC20 asset
        ...
-       token.safeIncreaseAllowance(address(executor), amount);
        if (reserveRecoverGas && gasleft() < _recoverGas) {
            token.safeTransfer(receiver, amount);
            ...
            return;
        }
+       token.safeIncreaseAllowance(address(executor), amount);
        try executor.swapAndCompleteBridgeTokens{...} ...
        token.safeApprove(address(executor), 0);
    }
}
```

LiFi: Fixed in PR 247.

Spearbit: Verified.

5.1.2 `CelerIMFacet` incorrectly sets `RelayerCelerIM` as receiver

Severity: High Risk

Context: CelerIMFacet.sol#L171-L177

Description: When assigning a `bytes memory` variable to a new variable, the new variable points to the same memory location. Changing any one variable updates the other variable. Here is a PoC as a foundry test

```
function testCopy() public {
  Pp memory x = Pp({
      a: 2,
      b: address(2)
  });
  Pp memory y = x;
  y.b = address(1);
  assertEq(x.b, y.b);
}
```

Thus, when `CelerIMFacet._startBridge()` updates `bridgeDataAdjusted.receiver`, `_bridgeData.receiver` is implicitly updated too. This makes the receiver on the destination chain to be the relayer address.

```
// case 'yes': bridge + dest call - send to relayer
ILiFi.BridgeData memory bridgeDataAdjusted = _bridgeData;
bridgeDataAdjusted.receiver = address(relayer);
(bytes32 transferId, address bridgeAddress) = relayer
.sendTokenTransfer{ value: msgValue }(bridgeDataAdjusted, _celerIMData);
// call message bus via relayer incl messageBusFee
relayer.forwardSendMessageWithTransfer{value: _celerIMData.messageBusFee}(
    _bridgeData.receiver,
    uint64(_bridgeData.destinationChainId),
    bridgeAddress,
    transferId,
    _celerIMData.callData
);
```

Recommendation: Remove `bridgeDataAdjusted` and work with `_bridgeData` as follows:

```
// case 'yes': bridge + dest call - send to relayer
-ILiFi.BridgeData memory bridgeDataAdjusted = _bridgeData;
-bridgeDataAdjusted.receiver = address(relayer);
+address receiver = _bridgeData.receiver;
+_bridgeData.receiver = address(relayer);
(bytes32 transferId, address bridgeAddress) = relayer
-.sendTokenTransfer{ value: msgValue }(bridgeDataAdjusted, _celerIMData);
+.sendTokenTransfer{ value: msgValue }(_bridgeData, _celerIMData);
// call message bus via relayer incl messageBusFee
relayer.forwardSendMessageWithTransfer{value: _celerIMData.messageBusFee}(
-  _bridgeData.receiver
+   receiver,
    uint64(_bridgeData.destinationChainId),
    bridgeAddress,
    transferId,
    _celerIMData.callData
);
+_bridgeData.receiver = receiver;
```

LiFi: Fixed in PR 252.

Spearbit: Verified.

5.1.3 Max approval to any address is possible

Severity: High Risk

Context: HopFacetOptimized.sol#L33-L45

Description: `HopFacetOptimized.setApprovalForBridges()` can be called by anyone to give max approval to any address for any ERC20 token. Any ERC20 token left in the Diamond can be stolen.

```
function setApprovalForBridges(address[] calldata bridges,address[] calldata tokensToApprove) external {
    ...
    LibAsset.maxApproveERC20(..., type(uint256).max);
    ...
}
```

Recommendation: Add authorization to the function `setApprovalForBridges()` so only the owner can call it. For example in the following way

```
function setApprovalForBridges(address[] calldata bridges,address[] calldata tokensToApprove) external {
+   LibDiamond.enforceIsContractOwner();
    ...
    LibAsset.maxApproveERC20(..., type(uint256).max);
    ...
}
```

LiFi: Fixed in PR 244.

Spearbit: Verified.

5.1.4 Return value of low-level `.call()` not checked

Severity: High Risk

Context: Receiver.sol#L209, Receiver.sol#L238, Receiver.sol#L257,

Description: Low-level primitive `.call()` doesn't revert in caller's context when the callee reverts. If its return value is not checked, it can lead the caller to falsely believe that the call was successful. `Receiver.sol` uses `.call()` to transfer the native token to `receiver`. If `receiver` reverts, this can lead to locked ETH in `Receiver` contract.

Recommendation: Check the return value and revert if `false` is returned.

```
-receiver.call{ value: amount }("");
+(bool success, ) = receiver.call{ value: amount }("");
+require(success)
```

LiFi: Fixed in PR 244.

Spearbit: Verified.

5.2 Medium Risk

5.2.1 Limits in `LIFuelFacet`

Severity: Medium Risk

Context: LIFuelFacet.sol#L72-L101

Description: The facet `LIFuelFacet` is meant for small amounts, however, it doesn't have any limits on the funds sent. This might result in funds getting stuck due to insufficient liquidity on the receiving side.

```
function _startBridge(...) ... {
    ...
    if (LibAsset.isNativeAsset(_bridgeData.sendingAssetId)) {
        serviceFeeCollector.collectNativeGasFees{...}(...);
    } else {
        LibAsset.maxApproveERC20(...);
        serviceFeeCollector.collectTokenGasFees(...);
        ...
    }
}
```

Recommendation: Consider enforcing limits in `LIFuelFacet`.

LiFi: Limits apply to all bridges and depend on the liquidity on the receiving chain. This information is usually not available on the source chain. For sure some high fixed limits could be added but they don't really check that there is that much liquidity available.

Checking limits and applying them to the calls is handled by the backend.

Spearbit: Acknowledged.

5.2.2 The optimal version `_depositAndSwap()` isn't always used

Severity: Medium Risk

Context: SwapperV2.sol#L138-L221, AmarokFacet.sol#L97-L102, CelerIMFacet.sol#L127-L132, AllBridgeFacet.sol#L79-L84, SquidFacet.sol#L112-L117

Description: The function `_depositAndSwap()` of `SwapperV2` has two versions. The second version keeps `_nativeReserve` that is meant for fees. Several facets don't use this version although their bridge does require native fees. This could result in calls reverting due to insufficient native tokens left.

```
    function _depositAndSwap(...) ... // 4 parameter version

    /// @param _nativeReserve Amount of native token to prevent from being swept back to the caller
    function _depositAndSwap(..., uint256 _nativeReserve) ... // 5 parameter version
```

Recommendation: Use the 5 parameter version of `_depositAndSwap()` where applicable.

LiFi: CelerIMFacet is changed to use the 5 parameter version. Other facets don't need this change since we don't bridge native tokens via those facets. Solved in PR 256.

Spearbit: Verified.

5.2.3 `setContractOwner()` is insufficient to lock down the owner

Severity: Medium Risk

Context: MakeLiFiDiamondImmutable.s.sol#L14-L17. LibDiamond.sol#L70-L75, OwnershipFacet.sol#L66-L73

Description: The function `transferOwnershipToZeroAddress()` is meant to make the Diamond immutable. It sets the contract owner to `0`. However, the contract owner can still be changed if there happens to be a `pendingOwner`. In that case `confirmOwnershipTransfer()` can still change the contract owner.

```
function transferOwnershipToZeroAddress() external {
    // transfer ownership to 0 address
    LibDiamond.setContractOwner(address(0));
}
function setContractOwner(address _newOwner) internal {
    DiamondStorage storage ds = diamondStorage();
    address previousOwner = ds.contractOwner;
    ds.contractOwner = _newOwner;
    emit OwnershipTransferred(previousOwner, _newOwner);
}
function confirmOwnershipTransfer() external {
    Storage storage s = getStorage();
    address _pendingOwner = s.newOwner;
    if (msg.sender != _pendingOwner) revert NotPendingOwner();
    emit OwnershipTransferred(LibDiamond.contractOwner(), _pendingOwner);
    LibDiamond.setContractOwner(_pendingOwner);
    s.newOwner = LibAsset.NULL_ADDRESS;
}
```

Recommendation: Possible solutions

- first call `cancelOwnershipTransfer()` (and ignore reverts)
- reset `s.newOwner` of the OwnershipFacet
- also remove `OwnershipFacet` (but then function `owner()` is no longer accessible)

LiFi: Solved in PR 250.

Spearbit: Verified.

5.2.4 `Receiver` does not verify address from the originator chain

Severity: Medium Risk

Context: Receiver.sol#L254 Receiver.sol#L282

Description: The `Receiver` contract is designed to receive the cross-chain call from `libDiamond` address on the destination chain. However, it does not verify the source chain address. An attacker can build a malicious `_callData`. An attacker can steal funds if there are left tokens and there are allowances to the `Executor`. Note that the tokens may be lost in issue: "Arithemetic underflow leading to unexpected revert and loss of funds in `Receiver` contract". And there may be allowances to `Executor` in issue "`Receiver` doesn't always reset allowance"

Recommendation: This is a tricky issue. Recommend to fixes other issues, especially making sure to clear the allowance.

The bridges we're integrating do not always allow users to verify the source address. Take `Amarok` for example, technically we can verify the sender's address:

```
    /// @notice Completes a cross-chain transaction with calldata via Amarok facet on the receiving
↪   chain.
    /// @dev This function is called from Amarok Router.
    /// @param _transferId The unique ID of this transaction (assigned by Amarok)
    /// @param _amount the amount of bridged tokens
    /// @param _asset the address of the bridged token
    /// @param * (unused) the sender of the transaction
    /// @param * (unused) the domain ID of the src chain
    /// @param _callData The data to execute
    function xReceive(
        bytes32 _transferId,
        uint256 _amount,
        address _asset,
        address _sender,
        uint32,
        bytes memory _callData
    ) external nonReentrant onlyAmarokRouter {
    if (_sender != address(diamond) {
       revert UnAuthorized();
   }
}
```

However, there's a special feature of Amarok. The `_sender` address would be `address(0)` if this is a fast path. If we revert such transactions, all the cross-chain transfers through `Amarok` would take about 30 minutes and this impacts UX.

LiFi: Solved by solving the related issues.

Spearbit: Verified.

5.2.5 Arithemetic underflow leading to unexpected revert and loss of funds in `Receiver` contract.

Severity: Medium Risk

Context: Receiver.sol#L254 Receiver.sol#L282

Description: The `Receiver` contract is designed to gracefully return the funds to users. It reserves the gas for recovering gas before doing swaps via `executor.swapAndCompleteBridgeTokens`. The logic of reserving gas for recovering funds is implemented at Receiver.sol#L236-L258

```
contract Receiver is ILiFi, ReentrancyGuard, TransferrableOwnership {
// ...
        if (reserveRecoverGas && gasleft() < _recoverGas) {
            // case 1a: not enough gas left to execute calls
            receiver.call{ value: amount }("");
// ...
        }

        // case 1b: enough gas left to execute calls
        try
            executor.swapAndCompleteBridgeTokens{
                value: amount,
                gas: gasleft() - _recoverGas
            }(_transactionId, _swapData, assetId, receiver)
        {} catch {
            receiver.call{ value: amount }("");
        }
// ...
}
```

The `gasleft()` returns the remaining gas of a call. It is continuously decreasing. The second query of `gasleft()` is smaller than the first query. Hence, if the attacker tries to relay the transaction with a carefully crafted gas where `gasleft() >= _recoverGas` at the first quiry and `gasleft() - _recoverGas` reverts. This results in the token loss in the `Receiver` contract.

Recommendation: Recommend to cache the `gasleft()`

```
        if (LibAsset.isNativeAsset(assetId)) {
            // case 1: native asset
+.          uint256 cacheGasleft = gasleft();
-            if (reserveRecoverGas && gasleft() < _recoverGas) {
+            if (reserveRecoverGas && cacheGasleft < _recoverGas) {

                // case 1a: not enough gas left to execute calls
                receiver.call{ value: amount }("");

                emit LiFiTransferRecovered(
                    _transactionId,
                    assetId,
                    receiver,
                    amount,
                    block.timestamp
                );
                return;
            }

            // case 1b: enough gas left to execute calls
            try
                executor.swapAndCompleteBridgeTokens{
                    value: amount,
-                    gas: gasleft() - _recoverGas
+                    gas: cacheGasleft - recoverGas
                }(_transactionId, _swapData, assetId, receiver)
            {} catch {
                receiver.call{ value: amount }("");
            }
        } else {
```

LiFi: Solved in PR 249.

Spearbit: Verified.

5.2.6 Use of name to identify a token

Severity: Medium Risk

Context: CelerIMFacet.sol#L79-L83

Description: `startBridgeTokensViaCelerIM()` uses the token name to identify `cfUSDC` token. Another token with the same name can pass this check. An attacker can create a scam token with the name "cfUSDC" and a function `canonical()` returning a legit ERC20 token address, say WETH. If this token is passed as `_bridgeData.sendingAssetId`, `CelerIMFacet` will transfer WETH.

```
if (
    keccak256(
        abi.encodePacked(
            ERC20(_bridgeData.sendingAssetId).symbol()
        )
    ) == keccak256(abi.encodePacked(("cfUSDC")))
) {
    // special case for cfUSDC token
    asset = IERC20(
        CelerToken(_bridgeData.sendingAssetId).canonical()
    );
}
```

Recommendation: Store `cfUSDC` address as a constant variable. Use `onlyAllowSourceToken` modifier to verify `_bridgeData.sendingAssetId` matches that address.

LiFi: Fixed in PR 254.

Spearbit: Verified.

5.2.7 Unvalidated `destination` address in Gravity faucet

Severity: Medium Risk

Context: GravityFacet.sol#L101

Description: In the Gravity faucet, there is an issue related to the validation of the `_gravityData.destinationAddress` address. The code does not validate if the provided destination address is in the valid `bech32` format.

This can potentially cause issues when sending tokens to the destination address. If the provided address is not in the `bech32` format, the tokens can be locked. Also, it can lead to confusion for the end-users as they might enter an invalid address and lose their tokens without any warning or error message.

Recommendation: To mitigate this issue, it is recommended to add a validation check for the `_gravityData.destinationAddress` address. The validation should ensure that the provided address is in the valid `bech32` format. This will help prevent the loss of tokens and provide better error handling for the end users.

Here are some libraries that might be used:

- Bech32.sol
- pStake's Bech32.sol

LiFi: We validate the address on the backend side, and we don't want to double check in the contract.

Spearbit: Acknowledged.

5.2.8 Hardcode or whitelist the Thorswap `vault` address

Severity: Medium Risk

Context: ThorSwapFacet.sol#LL104C27-L104C32

Description: The issue with this code is that the `depositWithExpiry` function allows the user to enter any arbitrary vault address, which could potentially lead to a loss of tokens. If a user enters an incorrect or non-existent `vault` address, the tokens could be lost forever. There should be some validation on the `vault` address to ensure that it is a valid and trusted address before allowing deposits to be made to it.

- Router

```
    // Deposit an asset with a memo. ETH is forwarded, ERC-20 stays in ROUTER
    function deposit(address payable vault, address asset, uint amount, string memory memo) public
↪   payable nonReentrant{
        uint safeAmount;
        if(asset == address(0)){
            safeAmount = msg.value;
            bool success = vault.send(safeAmount);
            require(success);
        } else {
            require(msg.value == 0, "THORChain_Router: unexpected eth");  // protect user from
↪   accidentally locking up eth
            if(asset == RUNE) {
                safeAmount = amount;
                iRUNE(RUNE).transferTo(address(this), amount);
                iERC20(RUNE).burn(amount);
            } else {
                safeAmount = safeTransferFrom(asset, amount); // Transfer asset
                _vaultAllowance[vault][asset] += safeAmount; // Credit to chosen vault
            }
        }
        emit Deposit(vault, asset, safeAmount, memo);
    }
```

Recommendation: Hardcode or whitelist the `vault` address.

Asgard Vault Addresses can be seen from here.

LiFi: LiFi Team claims that they are validating the address on the backend side.

Spearbit: Acknowledged.

5.3 Low Risk

5.3.1 Check enough native assets for fee

Severity: Low Risk

Context: SquidFacet.sol#L126-L175, CelerIMFacet.sol#L156-L187, DeBridgeFacet.sol#L119-L148, ArbitrumBridgeFacet.sol#L81-L117

Description: The function `_startBridge()` of `SquidFacet` adds `_squidData.fee` to `_bridgeData.minAmount`. It has verified there is enough native asset for `_bridgeData.minAmount`, but not for `_squidData.fee`. So this could use native assets present in the Diamond, although there normally shouldn't be any native assets left.

A similar issue occurs in:

- CelerIMFacet
- DeBridgeFacet

```
function _startBridge(...) ... {
    ...
    uint256 msgValue = _squidData.fee;
    if (LibAsset.isNativeAsset(address(sendingAssetId))) {
            msgValue += _bridgeData.minAmount;
    } ...
    ...
    squidRouter.bridgeCall{ value: msgValue }(...);
    ...
}
```

Recommendation: Consider checking enough native asset is sent for the fee.

Note: `ArbitrumBridgeFacet` has extra logic, which might be used elsewhere.

LiFi: We know it. We make no token in the Diamond and tx will revert if there's no enough native asset.

Spearbit: Acknowledged.

5.3.2 No check on native assets

Severity: Low Risk

Context: HopFacetOptimized.sol#L77-L92, GnosisBridgeL2Facet.sol#L39-L53, MultichainFacet.sol#L145-L166

Description: The functions `startBridgeTokensViaHopL1Native()` , `startBridgeTokensViaXDaiBridge()` and `startBridgeTokensViaMultichain()` don't check `_bridgeData.minAmount <= msg.value`. So this could use native assets that are still in the Diamond, although that normally shouldn't happen. This might be an issue in combination with reentrancy.

```
function startBridgeTokensViaHopL1Native(...) ... {
    _hopData.hopBridge.sendToL2{ value: _bridgeData.minAmount }( ... );
    ...
}
function startBridgeTokensViaXDaiBridge(...) ... {
    _startBridge(_bridgeData);
}
function startBridgeTokensViaMultichain(...) ... {
    if (!LibAsset.isNativeAsset(_bridgeData.sendingAssetId))
        LibAsset.depositAsset(_bridgeData.sendingAssetId,_bridgeData.minAmount);
    } // no check for native assets
    _startBridge(_bridgeData, _multichainData);
}
```

Recommendation: Consider sufficient native asset is supplied.

LiFi: We do this on purpose as no token should be in the contract. The transaction will revert if not enough tokens are there. So we don't need to check manually, and save gas.

Spearbit: Acknowledged.

5.3.3 Missing `doesNotContainDestinationCalls()`

Severity: Low Risk

Context: LIFuelFacet.sol#L27-L42

Description: The functions `startBridgeTokensViaLIFuel()` and `swapAndStartBridgeTokensViaLIFuel()` doesn't have `doesNotContainDestinationCalls()`.

```
function startBridgeTokensViaLIFuel(...)
    external
    payable
    nonReentrant
    refundExcessNative(payable(msg.sender))
    doesNotContainSourceSwaps(_bridgeData)
    validateBridgeData(_bridgeData) {
    ...
}
```

Recommendation: Consider adding `doesNotContainDestinationCalls()`.

LiFi: Fixed in PR 273.

Spearbit: Verified.

5.3.4 Race condition in `_startBridge` of `LIFuelFacet`.

Severity: Low Risk

Context: LIFuelFacet.sol#L72-L101, PeripheryRegistryFacet.sol#L34-L41

Description: If the mapping for `FEE_COLLECTOR_NAME` hasn't been set up yet, then `serviceFeeCollector` will be `address(0)` in function `_startBridge` of `LIFuelFacet`. This might give unexpected results.

```
function _startBridge(...) ... (
    ...
    ServiceFeeCollector serviceFeeCollector = ServiceFeeCollector(
            LibMappings.getPeripheryRegistryMappings().contracts[FEE_COLLECTOR_NAME]
        );
    ...
}
function getPeripheryContract(string calldata _name) external view returns (address) {
    return LibMappings.getPeripheryRegistryMappings().contracts[_name];
}
```

Recommendation: Consider calling `getPeripheryContract()` instead of accessing the `PeripheryRegistryMappings` directly. In `getPeripheryContract()` consider reverting if the value is empty.

LiFi: Keep this as it is. It will revert when it tries to call `collectNativeGasFees` or `collectTokenGasFees`

Spearbit: Acknowledged

5.3.5 Sweep tokens from Hopfacets

Severity: Low Risk

Context: HopFacet.sol, HopFacetOptimized.sol

Description: The Hop bridges `HopFacet` and `HopFacetOptimized` don't check that `_bridgeData.sendingAssetId` is the same as the bridge token. So this could be used to sweep tokens out of the Diamond contract.

Normally there shouldn't be any tokens left at the Diamond, however, in this version there are small amounts left: Etherscan LiFiDiamond.

Recommendation: Check `_bridgeData.sendingAssetId==bridge.token()`, in `_startBridge`, after `bridge` has been determined.

LiFi: We get the bridge from the sendingAssetId, which means the `bridge.token()` is `_bridgeData._sendingAssetId`.

```
        address sendingAssetId = _bridgeData.sendingAssetId;
        Storage storage s = getStorage();
        IHopBridge bridge = s.bridges[sendingAssetId];
```

Also, we make no token in the Diamond, so I think we can leave it as now.

Spearbit: Acknowledged.

5.3.6 Missing `emit` in `_swapAndCompleteBridgeTokens` of `Receiver`

Severity: Low Risk

Context: Receiver.sol#L224-L297

Description: In function `_swapAndCompleteBridgeTokens` the `catch` of ERC20 tokens does an `emit`, while the comparable `catch` of native assets doesn't do an `emit`.

```
function _swapAndCompleteBridgeTokens(...) ... {
    ...
    if (LibAsset.isNativeAsset(assetId)) {
        ..
        try ... {} catch {
            receiver.call{ value: amount }("");
            // no emit
        }
        ...
    } else { // case 2: ERC20 asset
        ...
        try ... {} catch {
            token.safeTransfer(receiver, amount);
            emit LiFiTransferRecovered(...);
        }
    }
}
```

Recommendation: Add an `emit` to the `catch` of native assets.

LiFi: Fixed in PR 281.

Spearbit: Verified.

5.3.7 Spam events in `ServiceFeeCollector`

Severity: Low Risk

Context: ServiceFeeCollector.sol#L44-L110

Description: The contract `ServiceFeeCollector` has several functions that collect fees and are permissionless. This could result in spam events, which might confuse the processing of the events.

```
function collectTokenGasFees(...) ... {
    ...
    emit GasFeesCollected(tokenAddress, receiver, feeAmount);
}
function collectNativeGasFees(...) ... {
    ...
    emit GasFeesCollected(LibAsset.NULL_ADDRESS, receiver, feeAmount);
}
function collectTokenInsuranceFees(...) ... {
    ...
    emit InsuranceFeesCollected(tokenAddress, receiver, feeAmount);
}
function collectNativeInsuranceFees(...) ... {
    ...
    emit InsuranceFeesCollected(LibAsset.NULL_ADDRESS,receiver,feeAmount);
}
```

Recommendation: If spam events are relevant, consider adding authorization to the functions.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.3.8 Function `depositAsset()` allows `0` amount of native assets

Severity: Low Risk

Context: LibAsset.sol#L107-L116

Description: The function `depositAsset()` disallows `amount == 0` for ERC20, however it does allow `amount == 0` for native assets.

```
function depositAsset(address assetId, uint256 amount) internal {
    if (isNativeAsset(assetId)) {
        if (msg.value < amount) revert InvalidAmount();
    } else {
        if (amount == 0) revert InvalidAmount();
        ...
    }
}
```

Recommendation: Consider also checking `amount == 0` for native assets.

LiFi: Fixed in PR 279.

Spearbit: Verified.

5.3.9 Inadequate expiration time check in `ThorSwapFacet`

Severity: Low Risk

Context: ThorSwapFacet.sol#L108

Description: According to Thorchain, the expiration time for certain operations should be set to +60 minutes. However, there is currently no check in place to enforce this requirement. This oversight may lead to users inadvertently setting incorrect expiration times, potentially causing unexpected behavior or issues within the `ThorSwapFacet`.

Recommendation: Consider adding a validation check within the `ThorSwapFacet` to ensure that expiration times are set to +60 minutes.

LiFi: Fixed with PR 278.

Spearbit: Verified.

5.3.10 Insufficient validation of `bridgedTokenSymbol` and `sendingAssetId`

Severity: Low Risk

Context: SquidFacet.sol#L146-L166

Description: During the code review, It has been noticed that the facet does not adequately check the correspondence between the `bridgedTokenSymbol` and `sendingAssetId` parameters. This oversight could allow for a random token to be sent to the Diamond, while still bridging another available token within the Diamond, even when no tokens should typically be left in the Diamond.

Recommendation: Consider implementing validation mechanisms to ensure that the `bridgedTokenSymbol` and `sendingAssetId` correspond before executing any asset transfers.

LiFi: Fixed with PR 282.

Spearbit: Verified.

5.3.11 Check for `destinationChainId` in `CBridgeFacet`

Severity: Low Risk

Context: CBridgeFacet.sol#L94-L128, Validatable.sol#L11-L22

Description: Function `_startBridge()` of `CBridgeFacet` contains a check on `destinationChainId` and contains conversions to `uint64`.

If both `block.chainid` and `_bridgeData.destinationChainId)` fit in an `uint64` then the checks of `modifier validateBridgeData` are already sufficient. When `_bridgeData.destinationChainId > type(uint64).max` then this never reverts:

```
if (uint64(block.chainid) == _bridgeData.destinationChainId)
    revert CannotBridgeToSameNetwork();
```

Then in the rest of the code it takes the truncated varion of the `destinationChainId` via `uint64(_bridgeData.destinationChainId)`, which can be any value, including `block.chainid`. So you can still bridge to the same chain.

```
function _startBridge(ILiFi.BridgeData memory _bridgeData,CBridgeData memory _cBridgeData) private {
    if (uint64(block.chainid) == _bridgeData.destinationChainId)
        revert CannotBridgeToSameNetwork();
    if (...) {
        cBridge.sendNative{ value: _bridgeData.minAmount }(... ,
↪   uint64(_bridgeData.destinationChainId),...);
    } else {
        ...
        cBridge.send(..., uint64(_bridgeData.destinationChainId), ...);
    }
}
modifier validateBridgeData(ILiFi.BridgeData memory _bridgeData) {
    ...
    if (_bridgeData.destinationChainId == block.chainid) {
        revert CannotBridgeToSameNetwork();
    }
    ...
}
```

Recommendation: If there are situations where `block.chainid` and `_bridgeData.destinationChainId)` could be too large for `uint64` then verify this. Remove the redundant check `if (uint64(block.chainid) == _bridgeData.destinationChainId)`.

LiFi: Fixed with PR 275.

Spearbit: Verified.

5.3.12 Absence of `nonReentrant` in HopFacetOptimized facet

Severity: Low Risk

Context: HopFacetOptimized.sol#L12, ERC20Proxy.sol

Description: HopFacetOptimized is a facet-based smart contract implementation that aims to optimize gas usage and streamline the execution of certain functions. It doesn't have the checks that other facets have:

```
nonReentrant
refundExcessNative(payable(msg.sender))
containsSourceSwaps(_bridgeData)
doesNotContainDestinationCalls(_bridgeData)
validateBridgeData(_bridgeData)
```

Most missing checks are done on purpose to save gas. However, the most important check is the `nonReentrant` modifier. On several places in the Diamond it is possible to trigger a reentrant call, for example via ERC777

tokens, custom tokens, native tokens transfers. In combination with the complexity of the code and the power of `ERC20Proxy.sol` it is difficult to make sure no attacks can occur.

Recommendation: Consider doing one of the following:

- add `nonReentrant` to the functions of `HopFacetOptimized.sol`
- seperate `HopFacetOptimized.sol` from the Diamond contracts;
  - to keep one address for settings allowances by the users consider using `ERC20Proxy` or permit2 of uniswap.

To prevent stuck native assets consider removing the `payable` keyword from the following functions because they normally wouldn't receive native assets. Note: this does cost some extra gas.

- `swapAndStartBridgeTokensViaHopL1Native()`
- `swapAndStartBridgeTokensViaHopL2Native()`

LiFi: Acknowledged.

- We think it's still safe without nonReentrant modifier since we make no token in the Diamond
- Keeping the `payable` keyword is useful in situations where the amount is partly paid in ERC20 tokens and partly in native tokens.

Spearbit: Acknowledged.

5.3.13 Revert for excessive approvals

Severity: Low Risk

Context: HopFacetOptimized.sol#L42

Description: Certain tokens, such as UNI and COMP, undergo a reversal if the value input for approval or transfer surpasses uint96.

Both aforementioned tokens possess unique logic in their approval process that sets the allowance to the maximum value of uint96 when the approval amount equals uint256(-1). Note: Hop currently doesn't support these token so set to low risk.

Recommendation: To address this issue, it is advised to implement a standardized validation mechanism for the approval processes. This mechanism should verify and restrict the input values to be within the allowable uint96 range, preventing revert occurrences.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.3.14 Inconsistent transaction failure/stuck due to missing validation of global fixed native fee rate and execution fee

Severity: Low Risk

Context: DeBridgeFacet.sol#L124

Description: The current implementation of the facet logic does not validate the global fixed native fee rate and execution fee, which can lead to inconsistent transaction failures or getting stuck in the process. This issue can arise when the fee rate is not set correctly or there are discrepancies between the fee rate used in the smart contract and the actual fee rate. This can result in transactions getting rejected or stuck, causing inconvenience to users and affecting the overall user experience.

Recommendation: To avoid this issue, it is recommended to include a validation step for the global fixed native fee rate in the transaction logic. This can be done by fetching the current fee rate from a debridge gate and comparing it with the fee rate used in the transaction.

uint protocolFee = deBridgeGate.globalFixedNativeFee;

LiFi: Fixed with PR 253.

Spearbit: Verified.

5.3.15 Incorrect value emitted

Severity: Low Risk

Context: LibSwap.sol#L76-L78

Description: `LibSwap.swap()` emits the following

```
emit AssetSwapped(
    transactionId,
    _swap.callTo,
    _swap.sendingAssetId,
    _swap.receivingAssetId,
    _swap.fromAmount,
    newBalance > initialReceivingAssetBalance // toAmount
        ? newBalance - initialReceivingAssetBalance
        : newBalance,
    block.timestamp
);
```

It will be difficult to interpret the value emitted for `toAmount` as the observer won't know which of the two values has been emitted.

Recommendation: Depending on the intended definition of `toAmount`, reconsider emitting one of the two values.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.3.16 Storage slots derived from hashes are prone to pre-image attacks

Severity: Low Risk

Context: AxelarFacet.sol#L17, HopFacet.sol#L19, MultichainFacet.sol#L24, OptimismBridgeFacet.sol#L26, OwnershipFacet.sol#L16, ReentrancyGuard.sol#L10, LibAccess.sol#L12, LibAllowList.sol#L12, LibDiamond.sol#L12, LibMappings.sol#L12-L1

Description: Storage slots manually constructed using keccak hash of a string are prone to storage slot collision as the pre-images of these hashes are known. Attackers may find a potential path to those storage slots using the keccak hash function in the codebase and some crafted payload.

Recommendation: Subtract 1 from the keccak hash before assigning it to `NAMESPACE`:

```
bytes32 internal constant STARGATE_NAMESPACE =
    bytes32(uint(keccak256("com.lifi.library.mappings.stargate")) - 1);
```

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.3.17 Incorrect arguments compared in `SquidFacet`

Severity: Low Risk

Context: SquidFacet.sol#L69

Description: `startBridgeTokensViaSquid ()` reverts if `(_squidData.sourceCalls.length > 0) != _bridgeData.hasSourceSwaps`. Here, `_squidData.sourceCalls` is an argument passed to Squid Router, and `_bridgeData.hasSourceSwaps` refers to source swaps done by `SquidFacet`. Ideally, `_bridgeData.hasSourceSwaps` should be `false` for this function (though it's not enforced) which means `_squidData.sourceCalls.length` has to be `0` for it to successfully execute.

Recommendation: Remove this check.

LiFi: Fixed in PR 258.

Spearbit: Verified.

5.3.18 Unsafe casting of bridge amount from `uint256` to `uint128`

Severity: Low Risk

Context: MakerTeleportFacet.sol#L118

Description: The issue with the code is that it performs an unsafe cast from a `uint256` value to a `uint128` value in the call to `initiateTeleport()` function. The `_bridgeData.minAmount` parameter passed to this function is of type `uint256`, but it is cast to `uint128` without any checks, which may result in a loss of precision or even an overflow.

```
    function _startBridge(ILiFi.BridgeData memory _bridgeData) internal {
        LibAsset.maxApproveERC20(
            IERC20(dai),
            address(teleportGateway),
            _bridgeData.minAmount
        );

        teleportGateway.initiateTeleport(
            l1Domain,
            _bridgeData.receiver,
+           uint128(_bridgeData.minAmount)
        );
```

Recommendation: To address this issue, it is recommended to use a `SafeCast` library that provides safe casting methods with appropriate checks to avoid any potential issues with loss of precision or overflows. The `SafeCast` library can be used to safely cast the `uint256` value to `uint128` before passing it to the `initiateTeleport()` function.

LiFi: Fixed with PR 257.

Spearbit: Verified.

5.4 Gas Optimization

5.4.1 Cache `s.anyTokenAddresses[_bridgeData.sendingAssetId]`

Severity: Gas Optimization

Context: MultichainFacet.sol#L203-L245

Description: The function `_startBridge()` of `MultichainFacet` contains the following expresssion. This retrieves the value for `s.anyTokenAddresses[_bridgeData.sendingAssetId]` twice. It might save some gas to first store this in a tmp variable.

```
s.anyTokenAddresses[_bridgeData.sendingAssetId] != address(0) ?
↪   s.anyTokenAddresses[_bridgeData.sendingAssetId]: ...
```

Recommendation: Consider storing `s.anyTokenAddresses[_bridgeData.sendingAssetId]` in tmp variable.

LiFi: Fixed with PR 293.

Spearbit: Verified.

5.4.2 DeBridgeFacet permit seems unusable

Severity: Gas Optimization

Context: DeBridgeFacet.sol#L119-L148

Description: The function `deBridgeGate.send()` takes a parameter `permit`. This can only be used if it's signed by the Diamond, see DeBridgeGate.sol#L654-L662. As there is no code to let the Diamond sign a permit, this function doesn't seem usable.

```
function _startBridge(...) ... {
    ...
    deBridgeGate.send{ value: nativeAssetAmount }(..., _deBridgeData.permit, ...);
    ...
}
```

Recommendation: Check the conclusion and consider removing `_deBridgeData.permit`.

LiFi: Fixed with PR 291.

Spearbit: Verified.

5.4.3 Redundant checks in CircleBridgeFacet

Severity: Gas Optimization

Context: CircleBridgeFacet.sol#L65-L87, Validatable.sol#L24-L39

Description: The function `swapAndStartBridgeTokensViaCircleBridge()` contains both `noNativeAsset()` and `onlyAllowSourceToken()`. The check `noNativeAsset()` is not necessary as `onlyAllowSourceToken()` already verifies the `sendingAssetId` isn't a native token.

```
function swapAndStartBridgeTokensViaCircleBridge(...) ... {
    ...
    noNativeAsset(_bridgeData)
    onlyAllowSourceToken(_bridgeData, usdc) {
    ...
}
modifier noNativeAsset(ILiFi.BridgeData memory _bridgeData) {
    if (LibAsset.isNativeAsset(_bridgeData.sendingAssetId)) {
        revert NativeAssetNotSupported();
    }
    _;
}
modifier onlyAllowSourceToken(ILiFi.BridgeData memory _bridgeData, address _token) {
    if (_bridgeData.sendingAssetId != _token) {
        revert InvalidSendingToken();
    }
    _;
}
```

Recommendation: Consider to remove `noNativeAsset()`.

LiFi: Fixed with PR 255 & PR 294.

Spearbit: Verified.

5.4.4 Redundant check on `_swapData`

Severity: Gas Optimization

Context: MakerTeleportFacet.sol#L89-L91

Description: This check is not present in the majority of the facets

```
if (_swapData.length == 0) {
    revert NoSwapDataProvided();
}
```

Ultimately, it's not required as `_depositAndSwap()` reverts when length is 0.

Recommendation: Consider removing this check to save gas.

LiFi: Fixed in PR 276.

Spearbit: Verified.

5.4.5 Duplicate checks done

Severity: Gas Optimization

Context: ArbitrumBridgeFacet.sol#L131, AmarokFacet.sol#L67

Description: In highlighted cases, a check has been done multiple times at different places:

- `validateBridgeData` modifier on `ArbitrumBridgeFacet._startBridge()` does checks already done by functions from which it's called.
- `depositAsset()` does some checks already done by `AmarokFacet.startBridgeTokensViaAmarok()`.

Recommendation:

- Remove `validateBridgeData` modifier on `ArbitrumBridgeFacet._startBridge()` function.
- `LibAsset.depositAsset()` performs several checks already done in modifiers present in `AmarokFacet`. While it's unsafe to remove those checks from `depositAsset()` as it's used at different places, to save

gas you can create its "unsafe" version that removes those checks. You need to be careful in integrating that version.

LiFi:

- Removed `validateBridgeData` modifier PR 288.
- We don't want to maintain multiple versions of `depositAsset()` method.

Spearbit: Verified and acknowledged.

5.4.6 `calldata` can be used instead of `memory`

Severity: Gas Optimization

Context: LibAsset.sol#L120, AcrossFacet.sol#L78, CBridgeFacet.sol#L70, CBridgeFacet.sol#L96, CelerIMFacet.sol#L116, CelerIMFacet.sol#L158, GravityFacet.sol#L90, HopFacet.sol#L149, MultichainFacet.sol#L205, SquidFacet.sol#L128, RelayerCelerIM.sol#L157-L15, ServiceFeeCollector.sol#L122

Description: When the incoming argument is constant, `calldata` can be used instead of `memory` to save gas on copying it to memory. This remains true for individual array elements.

Recommendation: Consider replacing `memory` with `calldata` for highlighted code above. Also, consider tracing those arguments through function calls to ensure it's not being copied to `memory`.

LiFi: Fixed in PR 268.

Spearbit: Verified.

5.4.7 Further gas optimizations for `HopFacetOptimized`

Severity: Gas Optimization

Context: HopFacetOptimized.sol#L15-L22, HopFacetOptimized.sol#L50-L72, ILiFi.sol#L7-L18, LibAsset.sol#L91-L105

Description: For the contract `HopFacetOptimized` it is very important to be gas optimized. Especially on Arbritrum, it is relatively expensive due to the calldata.

```
struct HopData {
    uint256 bonderFee;
    uint256 amountOutMin;
    uint256 deadline;
    uint256 destinationAmountOutMin;
    uint256 destinationDeadline;
    IHopBridge hopBridge;
}
struct BridgeData {
    bytes32 transactionId;
    string bridge;
    string integrator;
    address referrer;
    address sendingAssetId;
    address receiver;
    uint256 minAmount;
    uint256 destinationChainId;
    bool hasSourceSwaps;
    bool hasDestinationCall;
}
function startBridgeTokensViaHopL1ERC20(
    ILiFi.BridgeData calldata _bridgeData,
    HopData calldata _hopData
) external {
    // Deposit assets
    LibAsset.transferFromERC20(...);
    _hopData.hopBridge.sendToL2(...);
    emit LiFiTransferStarted(_bridgeData);
}
function transferFromERC20(...) ... {
    if (assetId == NATIVE_ASSETID) revert NullAddrIsNotAnERC20Token();
    if (to == NULL_ADDRESS) revert NoTransferToNullAddress();
    IERC20 asset = IERC20(assetId);
    uint256 prevBalance = asset.balanceOf(to);
    SafeERC20.safeTransferFrom(asset, from, to, amount);
    if (asset.balanceOf(to) - prevBalance != amount)
        revert InvalidAmount();
}
```

Recommendation: Consider shrinking the HopData and BridgeData. The `transactionId` might be enough to track the transfer. A signature from the API could be added to verify the authenticity and prevent spam. In that case, adding a `nonce` is useful to prevent replays. Some `uint256` might be replaced by smaller types. Consider having a dedicated version of `transferFromERC20()` with fewer checks. Note this will increase some risks.

Consider having a standalone version of the `HopFacetOptimized` bridge to remove any overhead from the Diamond and its proxy. In that case consider having a vanity address with leading `0`s for `HopFacetOptimized`, see create2crunch. Consider having a very limited number of functions and make sure the most important functions have the lowest value for their signature. This will reduce some gas.

Remove redundant event : HopFacetOptimized.sol#L26

Note: the `_depositAndSwap()` is complicated and gas intensive and could be optimized by a redesign.

LiFi: Several optimizations applied in `HopFacetPacked.sol` via PR 283.

Spearbit: Verified.

5.4.8 `payable` keyword can be removed for some bridge functions

Severity: Gas Optimization

Context: CircleBridgeFacet.sol#L48, MakerTeleportFacet.sol#L59, GnosisBridgeFacet.sol#L43

Description: For the above highlighted functions, the native token is never forwarded to the underlying bridge. In these cases, `payable` keyword and related modifier `refundExcessNative(payable(msg.sender))` can be removed to save gas.

Recommendation: Remove `payable` keyword and `refundExcessNative(payable(msg.sender))` modifier for the highlighted functions.

LiFi: Fixed in PR 265.

Spearbit: Verified.

5.4.9 `AmarokData.callTo` can be removed

Severity: Gas Optimization

Context: AmarokFacet.sol#L29, AmarokFacet.sol#L122-L124

Description: `AmarokFacet`'s final receiver can be different from `_bridgeData.receiver`

```
address receiver = _bridgeData.hasDestinationCall
    ? _amarokData.callTo
    : _bridgeData.receiver;
```

Since both `_amarokData.callTo` and `_bridgeData.receiver` are passed by the caller, `AmarokData.callTo` can be removed, and `_bridgeData.receiver` can be assumed as the final receiver.

Recommendation: Remove `callTo` field from `AmarokData` struct, and apply this diff

```
-address receiver = _bridgeData.hasDestinationCall
-    ? _amarokData.callTo
-    : _bridgeData.receiver;

// initiate bridge transaction
connextHandler.xcall{ value: _amarokData.relayerFee }(
    _amarokData.destChainDomainId,
-    receiver,
+   _bridgeData.receiver
```

LiFi: Fixed in PR 263.

Spearbit: Verified.

5.4.10 Use `requiredEther` variable instead of adding twice

Severity: Gas Optimization

Context: ArbitrumBridgeFacet.sol#L144-L178

Description: The `cost` and `nativeAmount` are added twice to calculate the `requiredEther` variable, which can lead to increased gas consumption.

Recommendation: Use `requiredEther` variable in the `_startNativeBridge` function.

```
        if (isNativeTransfer) {
-           _startNativeBridge(_bridgeData, _arbitrumData, _cost);
+           _startNativeBridge(_bridgeData, _arbitrumData, requiredEther);
        } else {
            _startTokenBridge(_bridgeData, _arbitrumData, _cost);
        }

....

    function _startNativeBridge(
        ILiFi.BridgeData memory _bridgeData,
        ArbitrumData calldata _arbitrumData,
        uint256 requiredEther
    ) private {
        inbox.unsafeCreateRetryableTicket{
-           value: _bridgeData.minAmount + cost
+           value: requiredEther
        }
```

LiFi: Fixed with PR 262.

Spearbit: Verified.

5.4.11 `refundExcessNative` modifier can be gas-optimized

Severity: Gas Optimization

Context: SwapperV2.sol#L118-L126

Description: The highlighted code above can be gas-optimized by removing 1 `if` condition.

Recommendation: Replace the highlighted code with

```
if (finalBalance > initialBalance) {
    LibAsset.transferAsset(
        LibAsset.NATIVE_ASSETID,
        _refundReceiver,
        finalBalance - initialBalance
    );
}
```

LiFi: Fixed in PR 260.

Spearbit: Verified.

5.4.12 `BridgeData.hasSourceSwaps` can be removed

Severity: Gas Optimization

Context: ILiFi.sol#L16

Description: The field `hasSourceSwaps` can be removed from the struct `BridgeData`. `_swapData` is enough to identify if source swaps are needed.

Recommendation: Delete the field `BridgeData.hasSourceSwaps` and all the resulting checks like `containsSourceSwaps` and `doesNotContainSourceSwaps` modifiers.

LiFi: We need `hasSourceSwaps` in `BridgeData` for off-chain analysis.

Spearbit: Acknowledged.

5.4.13 Unnecessary validation argument for native token amount

Severity: Gas Optimization

Context: ServiceFeeCollector.sol#L56, ServiceFeeCollector.sol#L90

Description: Since both `msg.value` and `feeAmount` is controlled by the caller, you can remove `feeAmount` as an argument and assume `msg.value` is what needs to be collected.

This will save gas on comparing these two values and refunding the extra.

Recommendation: Remove `feeAmount` as an argument for `collectNativeInsuranceFees()` and `collectNativeGasFees()`

```
-function collectNativeInsuranceFees(uint256 feeAmount, address receiver)
+function collectNativeInsuranceFees(address receiver)
    external
    payable
{
-    if (msg.value < feeAmount) revert NotEnoughNativeForFees();
-    uint256 remaining = msg.value - (feeAmount);
-    // Prevent extra native token from being locked in the contract
-    if (remaining > 0) {
-        (bool success, ) = payable(msg.sender).call{ value: remaining }(
-            ""
-        );
-        if (!success) {
-            revert TransferFailure();
-        }
-    }
    emit InsuranceFeesCollected(
        LibAsset.NULL_ADDRESS,
        receiver,
-       feeAmount
+       msg.value
    );
}
```

LiFi: Fixed in PR 259.

Spearbit: Verified.

5.5 Informational

5.5.1 Restrict access for cBridge refunds

Severity: Informational

Context: WithdrawFacet.sol#L28-L56

Description: cBridge refunds need to be triggered from the contract that sent the transaction to cBridge. This can be done using the `executeCallAndWithdraw` function. As the function is not cBridge specific it can do any calls for the Diamond contract. Restricting what that function can call would allow more secure automation of refunds.

Recommendation: Add a dedicated refund function to the Diamond similar to RelayerCelerIM.sol#L443-L477.

LiFi: PR 243.

Spearbit: Verified.

5.5.2 Stargate now supports multiple pools for the same token

Severity: Informational

Context: StargateFacet.sol#L25-L28

Description: The Stargate protocol now supports multiple pools for the same token on the same chain, each pool may be connected to one or many other chains. It is not possible to store a one-to-one token-to-pool mapping.

Recommendation: Pass pools in callData to ensure support for all pools.

LiFi: Fixed in PR 237.

Spearbit: Verified.

5.5.3 Expose `receiver` in `GenericSwapFacet` facet

Severity: Informational

Context: GenericSwapFacet.sol#L60-L69

Description: Other than the bridge facets the swap facet does not emit the receiver of a transaction yet.

Recommendation: Emit a new event containing that information.

LiFi: Added in PR 236.

Spearbit: Fix is verified with PR 236.

5.5.4 Track the destination chain on `ServiceFeeCollector`

Severity: Informational

Context: ServiceFeeCollector.sol#L44-L56

Description: `ServiceFeeCollector` collects gas fees to send to the destination chain. For example

```
/// @param receiver The address to send gas to on the destination chain
function collectTokenGasFees(
    address tokenAddress,
    uint256 feeAmount,
    address receiver
)
```

However, the destination chain is never tracked in the contract.

Recommendation: Consider adding the destination chain ID to the functions `collectTokenGasFees()` and `collectNativeGasFees()`.

LiFi: Fixed in PR 289.

Spearbit: Verified.

5.5.5 `Executor` can reuse `SwapperV2` functions

Severity: Informational

Context: Executor.sol#L32, SwapperV2.sol#L30-L34, Executor.sol#L241, SwapperV2.sol#L30

Description: `Executor.sol`'s `noLeftOvers` and `_fetchBalances()` is copied from `SwapperV2.sol`.

Recommendation: Consider reusing these functions from `SwapperV2`, perhaps by extracting these functions in `LibSwap`.

LiFi: We will leave as is for now.

Spearbit: Acknowledged.

5.5.6 Consider adding `onERC1155Received`

Severity: Informational

Context: Executor.sol#L270

Description: In addition to ERC721, NFTs can be created using ERC1155 standard. Since, the use case of purchasing an NFT has to be supported, support for ERC1155 tokens can be added.

Recommendation: Consider adding `onERC1155Received` function to `Executor.sol` using `ERC1155Holder`. You can also use `ERC721Holder.sol` to replace the current `onERC721Received()` function.

LiFi: Fixed in PR 295.

Spearbit: Verified.

5.5.7 `SquidFacet` uses a different string encoding library

Severity: Informational

Context: SquidFacet.sol#L157

Description: `SquidFacet` uses an OZ library to convert address to string, whereas the underlying bridge uses a different library.

Fuzzing showed that these implementations are equivalent.

Recommendation: This issue is just to highlight this difference. Consider commenting about this difference, or changing the library.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.5.8 Assembly in `StargateFacet` can be replaced with Solidity

Severity: Informational

Context: StargateFacet.sol#L295-L313

Description: The function `toBytes()` contains assembly code that can also be replaced with solidity code. Also, see how-to-convert-an-address-to-bytes-in-solidity.

```
function toBytes(address _address) private pure returns (bytes memory) {
    bytes memory tempBytes;
    assembly {
        let m := mload(0x40)
        _address := and(_address,0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
        mstore(add(m, 20),xor(0x140000000000000000000000000000000000000000, _address) )
        mstore(0x40, add(m, 52))
        tempBytes := m
    }
    return tempBytes;
}
```

Recommendation: Consider using `abi.encodePacked(_address))`.

LiFi: Fixed with PR 290.

Spearbit: Verified.

5.5.9 Doublecheck `quoteLayerZeroFee()`

Severity: Informational

Context: StargateFacet.sol#L168-L218

Description: The function quoteLayerZeroFee uses `msg.sender` to determine a fee, while `_startBridge()` uses `_bridgeData.receiver` to execute ‘ router.swap. This might give different results.

```
function quoteLayerZeroFee(...) ... {
    return router.quoteLayerZeroFee( ... , toBytes(msg.sender) );
}
function _startBridge(...)
    ...
    router.swap{ value: _stargateData.lzFee }(..., toBytes(_bridgeData.receiver), ... );
    ...
}
```

Recommendation: Doublecheck using `msg.sender` in `quoteLayerZeroFee()` results in the right fee.

LiFi: Fixed with PR 292.

Spearbit: Verified.

5.5.10 Missing `modifier refundExcessNative()`

Severity: Informational

Context: GnosisBridgeFacet.sol#L59-L83, GnosisBridgeL2Facet.sol#L58-L82

Description: The function `swapAndStartBridgeTokensViaXDaiBridge()` of `GnosisBridgeFacet()` and `GnosisBridgeL2Facet()` don't have the `modifier refundExcessNative()`. While other Facets have such a modifier.

Recommendation: Doublecheck if the `modifier refundExcessNative()` has to be added.

LiFi: Fixed with PR 280.

Spearbit: Verified.

5.5.11 Special case for `cfUSDC` tokens in `CelerIMFacet`

Severity: Informational

Context: CelerIMFacet.sol#L63-L149

Description: The function `startBridgeTokensViaCelerIM()` has a special case for `cfUSDC` tokens, whereas `swapAndStartBridgeTokensViaCelerIM()` doesn't have this.

```
function startBridgeTokensViaCelerIM(...) ... {
        if (!LibAsset.isNativeAsset(_bridgeData.sendingAssetId)) {
            if (...)  {   // special case for cfUSDC token
                asset = IERC20(CelerToken(_bridgeData.sendingAssetId).canonical());
            } else {
                ...
            }
        }
         ...
    }
function swapAndStartBridgeTokensViaCelerIM(...) ... {
    ...
    if (!LibAsset.isNativeAsset(_bridgeData.sendingAssetId)) {
        // no special case  for cfUSDC token
    }
    ...
}
```

Recommendation: Doublecheck if `swapAndStartBridgeTokensViaCelerIM()` also should have a special case for `cfUSDC` tokens.

LiFi: Fixed with PR 254.

Spearbit: Verified.

5.5.12 External calls of `SquidFacet`

Severity: Informational

Context: SquidFacet.sol#L126-L175

Description: The functions `CallBridge` and `CallBridgeCall` do random external calls. This is done via a separate contract multicall SquidMulticall. This might be used to try reentrancy attacks.

```
function _startBridge(...) ... {
    ...
    squidRouter.bridgeCall{ value: msgValue }(...);
    ...
    squidRouter.callBridgeCall{ value: msgValue }(...);
    ...
}
```

Recommendation: This adds to the case to have `nonReentrant` modifiers.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.5.13 Missing test coverage for `triggerRefund` Function

Severity: Informational

Context: RelayerCelerIM.sol#L449

Description: The current test suite does not include test cases for the `triggerRefund` function. This oversight may lead to undetected bugs or unexpected behavior in the function's implementation.

Recommendation: Add the new test cases to the existing test suite and run the tests to confirm that the `triggerRefund` function operates as expected.

LiFi: Fixed with PR 296.

Spearbit: Verified.

5.5.14 Implicit assumption in `MakerTeleportFacet`

Severity: Informational

Context: MakerTeleportFacet.sol#L108-L122

Description: The function `_startBridge()` of `MakerTeleportFacet` has the implicit assumption that `dai` is an ERC20 token. However on `GnosisChain` the native asset is (x)dai. Note: `DAI` on `GnosisChain` is an `ERC20`, so unlikely this would be a problem in practice.

```
function _startBridge(ILiFi.BridgeData memory _bridgeData) internal {
    LibAsset.maxApproveERC20( IERC20(dai),...);
    ...
}
```

Recommendation: Enforce `dai !=0` or at least add a comment.

LiFi: Acknowledged, `MakerTeleportFacet` is used to bridge `Dai(0xDA10009cBd5D07dd0CeCc66161FC93D7c9000da1)` from Optimism and Arbitrum to mainnet.

Spearbit: Acknowledged

5.5.15 Robust allowance handling in `maxApproveERC20()`

Severity: Informational

Context: LibAsset.sol#L52-L67

Description: Some tokens, like USDT, require setting the approval to `0` before setting it to another value. The function `SafeERC20.safeIncreaseAllowance()` doesn't do this.

Luckily `maxApproveERC20()` sets the allowance so high that in practice this never has to be increased.

```
function maxApproveERC20(...) ... {
    ...
    uint256 allowance = assetId.allowance(address(this), spender);
    if (allowance < amount)
            SafeERC20.safeIncreaseAllowance(IERC20(assetId), spender, MAX_UINT - allowance);
    }
```

Recommendation: Consider setting the allowance first to `0` in `maxApproveERC20()`.

LiFi: Fixed with PR 286.

Spearbit: Verified.

5.5.16 Unused re-entrancy guard

Severity: Informational

Context: RelayerCelerIM.sol#L21

Description: The RelayerCelerIM.sol#L21 includes a redundant re-entrancy guard, which adds an extra layer of protection against re-entrancy attacks. While re-entrancy guards are crucial for securing contracts, this particular guard is not used.

Recommendation: Consider reviewing the contract and delete re-entrancy guard if it's not needed.

LiFi: Fixed with PR 272.

Spearbit: Verified.

5.5.17 Redundant duplicate import in the `LIFuelFacet`

Severity: Informational

Context: LIFuelFacet.sol#L13

Description: The current LIFuelFacet.sol contains a redundant duplicate import. Identifying and removing duplicate imports can streamline the contract and improve maintainability.

Recommendation: Consider removing the duplicate checks.

```
  import { LibMappings } from "../Libraries/LibMappings.sol";
  import { Validatable } from "../Helpers/Validatable.sol";
- import { LibMappings } from "../Libraries/LibMappings.sol";
```

LiFi: Fixed in PR 270.

Spearbit: Verified.

5.5.18 Extra checks in `executeMessageWithTransfer()`

Severity: Informational

Context: RelayerCelerIM.sol#L67-L111

Description: The function `executeMessageWithTransfer()` of `RelayerCelerIM` ignore the first parameter. It seems this could be used to verify the origin of the transaction, which could be an extra security measure.

```
* @param * (unused) The address of the source app contract
function executeMessageWithTransfer(address, ...) ... {
}
```

Recommendation: Check if it is useful to verify the value of the first parameter or `executeMessageWithTransfer()`.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.5.19 Variable visibility is not uniform

Severity: Informational

Context: ThorSwapFacet.sol#L19, SynapseBridgeFacet.sol#L21

Description: In the current facets, state variables like router/messenger visibilities are not uniform, with some variables declared as public while others are private.

thorchainRouter => is defined as `public`.

synapseRouter => is defined as `public`.

deBridgeGate => is defined as `private`

Recommendation: Consider reviewing the codebase to identify variables with incorrect visibility settings.

LiFi: Fixed with PR 285.

Spearbit: Verified.

5.5.20 Library `LibMappings` not used everywhere

Severity: Informational

Context: LibMappings.sol

Description: The library `LibMappings` is used in several facets. However, it is not used in the following facets

- ReentrancyGuard
- AxelarFacet
- HopFacet.sol
- MultichainFacet
- OptimismBridgeFacet
- OwnershipFacet

Recommendation: Consider using `LibMappings` where possible.

LiFi: Solved by removing LibMappings in all other facets, via PR 287.

Spearbit: Verified.

5.5.21 `transferERC20()` doesn't have a null address check for receiver

Severity: Informational

Context: LibAsset.sol#L79-L98

Description: `LibAsset.transferFromERC20()` has a null address check on the receiver, but `transferERC20()` does not.

Recommendation: Verify if this difference is intentional. Consider making these checks uniform across both functions depending on the requirement.

LiFi: Fixed in PR 284.

Spearbit: Verified.

5.5.22 `LibBytes` can be improved

Severity: Informational

Context: LibBytes.sol

Description: The following functions are not used

- concat()
- concatStorage()
- equal()
- equalStorage()
- toBytes32()
- toUint128()
- toUint16()
- toUint256()
- toUint32()
- toUint64()
- toUint8()
- toUint96()

The call to function `slice()` for `calldata` arguments (as done in `AxelarExecutor`) can be replaced with the in-built slicing provided by Solidity. Refer to its documentation.

Recommendation: Consider removing unused functions. If you want to depend on default Solidity primitives, you can remove the call to `slice()` with the default counterpart for `calldata` arrays.

Note : Also SquidFacet inserting `Strings` library. The function can be kept in the `LibBytes`.

LiFi: Fixed with PR 267.

Spearbit: Verified.

5.5.23 Keep generic errors in the `GenericErrors`

Severity: Informational

Context: RelayerCelerIM.sol#L30

Description: During the code review, It has been noticed that some of the contracts are re-defined errors. The generic errors like a `WithdrawFailed` can be kept in the GenericErrors.sol

Recommendation: It is recommended to review all errors and place them into the GenericErrors.sol.

LiFi: Fixed with PR 271.

Spearbit: Verified.

5.5.24 Attention points for making the Diamond immutable

Severity: Informational

Context: src

Description: There are additional attention points to decide upon when making the Diamond immutable:

After removing the Owner, the following functions won't work anymore:

- AccessManagerFacet.sol - `setCanExecute()`
- AxelarFacet.sol - `setChainName()`
- HopFacet.sol - `registerBridge()`
- MultichainFacet.sol - `updateAddressMappings()` & `registerRouters()`
- OptimismBridgeFacet.sol - `registerOptimismBridge()`
- PeripheryRegistryFacet.sol - `registerPeripheryContract()`
- StargateFacet.sol - `setStargatePoolId()` & `setLayerZeroChainId()`
- WormholeFacet.sol - `setWormholeChainId()` & `setWormholeChainIds()`

There is another authorization mechanism via `LibAccess`, which arranges access to the functions of

- DexManagerFacet.sol
- WithdrawFacet.sol

Several Periphery contracts also have an Owner:

- AxelarExecutor.sol
- ERC20Proxy.sol
- Executor.sol
- FeeCollector.sol
- Receiver.sol
- RelayerCelerIM.sol
- ServiceFeeCollector.sol

Additionally `ERC20Proxy` has an authorization mechanism via `authorizedCallers[]`

Recommendation: Consider doing the following

- create an alternative way to update the essentials functions that are currently authorized via the owner, perhaps with `LibAccess`;
- remove the authorizations of `DexManagerFacet.sol` and `WithdrawFacet.sol` or remove these facets;
- evaluate the Periphery contracts and determine if the Owners should be locked down;
- evaluate the authorization mechanism of ERC20Proxy and check what should be locked down.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.5.25 Check on the final asset in `_swapData`

Severity: Informational

Context: MakerTeleportFacet.sol#L92

Description: `MakerTeleportFacet` verifies that the final received asset in `_swapData` is DAI. This check is not present in majority of the facets (including `CircleBridgeFacet`). Ideally, every facet should have the check that the final `receivingAssetId` is equal to `sendingAssetId`.

Recommendation: Consider adding this check for all Facets. If you do, considering making a `modifier` for:

```
if (_swapData.length == 0) {
    revert NoSwapDataProvided();
}
if (_swapData[_swapData.length - 1].receivingAssetId != dai) {
    revert InvalidSendingToken();
}
```

Alternatively, consider removing it everywhere because the transaction will revert anyway.

LiFi: We decided to remove those checks from already existing facets such as `MakerTeleportFacet`. We make no token in the Diamond, so even if `receivingAssetId` is not equal to `sendingAssetId`, it will fail.

Checked removed in PR 276.

Spearbit: Verified.

5.5.26 Discrepancies in pragma versioning across faucet implementations

Severity: Informational

Context: AllBridgeFacet.sol#L2, ThorSwapFacet.sol#L2

Description: The use of different pragma versions in facet implementations can present several implications, with potential risks and compliance concerns that need to be addressed to maintain robust and compliant contracts.

Recommendation: Consider changing pragmas with *0.8.17*.

LiFi: Solved in PR 277.

Spearbit: Verified.

5.5.27 Inconsistent use of `validateDestinationCallFlag()`

Severity: Informational

Context: AmarokFacet.sol#L61-L65, SquidFacet.sol#L74-L79

Description: Highlighted code can be replaced with a call to `validateDestinationCallFlag()` function as done in other Facets.

Recommendation: Replace the highlighted code with a call to `validateDestinationCallFlag()` function.

LiFi: Fixed in PR 269.

Spearbit: Verified.

5.5.28 Inconsistent utilization of the `isNativeAsset` function

Severity: Informational

Context: AcrossFacet.sol#L106, LibAsset.sol#L97

Description: The `isNativeAsset` function is designed to distinguish native assets from other tokens within facet-based smart contract implementations. However, it has been observed that the usage of the isNativeAsset function is not consistent across various facets. Ensuring uniform application of this function is crucial for maintaining the accuracy and reliability of the asset identification and processing within the facets.

Recommendation: Consider utilizing `isNativeAsset` function.

LiFi: Fixed with PR 274.

Spearbit: Verified.

5.5.29 Unused events/errors

Severity: Informational

Context: Executor.sol#L23-L24, HopFacetOptimized.sol#L26-L27, ThorSwapFacet.sol#L10

Description: The contracts contain several events and error messages that are not used anywhere in the contract code. These unused events and errors add unnecessary code to the contract, increasing its size.

Recommendation: It is recommended to remove any unused events and errors from the contract code to make it more streamlined and easier to read. This can be done by performing a code review and identifying any events or errors that are not referenced in the contract code.

LiFi: Fixed with PR 261 and PR 266.

Spearbit: Verified.

5.5.30 Make bridge parameters dynamic by keeping them as a parameter

Severity: Informational

Context: HopFacetOptimized.sol#L147, MakerTeleportFacet.sol#L115

Description: The current implementation has some bridge parameters hardcoded within the smart contract. This approach limits the flexibility of the contract and may cause issues in the future when upgrades or changes to the bridge parameters are required. It would be better to keep the bridge parameters as a parameter to make them dynamic and easily changeable in the future.

`HopFacetOptimized.sol` => *Relayer & RelayerFee*

`MakerTeleportFacet.sol`'s => Operator person (or specified third party) responsible for initiating minting process on destination domain by providing (in the fast path) Oracle attestations.

Recommendation: Modify the smart contract code to include bridge parameters as a parameter rather than hardcoding them. This will allow for easier upgrades and changes to the bridge parameters in the future without the need for extensive modifications to the code.

LiFi: Acknowledged.

Spearbit: Acknowledged.

5.5.31 Incorrect comment

Severity: Informational

Context: CircleBridgeFacet.sol#L32

Description: The highlighted comment incorrectly refers USDC address as DAI address.

Recommendation:

```
-    /// @param _usdc The address of DAI on the source chain.
+   /// @param _usdc The address of USDC on the source chain.
```

LiFi: Fixed in PR 264.

Spearbit: Verified.

5.5.32 Redundant console log

Severity: Informational

Context: ThorSwapFacet.sol#L13

Description: The contract includes `Console.sol` from test file, which is only used for debugging purposes. Including it in the final version of the contract can increase the contract size and consume more gas, making it more expensive to deploy and execute.

Recommendation: Consider removing `console.sol`.

LiFi: Fixed with PR 261.

Spearbit: Verified.

5.5.33 `SquidFacet` doesn't revert for incorrect `routerType`

Severity: Informational

Context: SquidFacet.sol#L174

Description: If `_squidData.routeType` passed by the user doesn't match `BridgeCall`, `CallBridge`, or `CallBridgeCall`, `SquidFacet` just takes the funds from the user and returns without calling the bridge. This, when combined with the issue "*Max approval to any address is possible*", lets anyone steal those funds. Note: the Solidity `enum` checks should prevent this issue, but it is safer to do an extra check.

Recommendation: Add an `else` condition which just reverts.

LiFi: Fixed in PR 248.

Spearbit: Verified.