
LI.FI Security Review

GasZipFacet(v2.0.0), GasZipPeriphery(v1.0.0) and IGasZip

Independent Review By:

Sujith Somraaj (somraajsujith@gmail.com)

November 7, 2024

Contents

1	About Researcher	2
2	Disclaimer	2
3	Scope	2
4	Risk classification	2
4.1	Impact	2
4.2	Likelihood	3
4.3	Action required for severity levels	3
5	Executive Summary	3
6	Findings	4
6.1	Medium Risk	4
6.1.1	Validate msg.value in startBridgeTokensViaGasZip function	4
6.1.2	Redundant receiverAddress checks instartBridgeTokensViaGasZip and swapAndStartBridgeTokensViaGasZip functions	4
6.2	Low Risk	4
6.2.1	receiverAddress validation in _startBridge is redundant	4
6.2.2	Assert balances post swap in depositToGasZipERC20 function	5
6.2.3	Validate if finalTokenId of _swapData is native in swapAndStartBridgeTokensViaGasZip function	5
6.3	Gas Optimization	6
6.3.1	getDestinationChainsValue function could be optimized	6
6.4	Informational	7
6.4.1	Remove unused NativeAssetTransferFailed error import	7
6.4.2	Revert if _swapData.sendingAssetId is NATIVE in depositToGasZipERC20 function	7
6.4.3	Validate _gasZipData.receiver in depositToGasZipNative function	7
6.4.4	Replace require with custom error inside getDestinationChainsValue function	7
6.4.5	Remove unused error OnlySwapsFromERC20ToNativeAllowed	8
7	Appendix	9

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/GasZipFacet.sol
- src/Periphery/GasZipPeriphery.sol
- src/Interfaces/IGasZip.sol

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 2.5 days 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 11 issues were found.

Project Summary	
Project Name	LI.FI
Repository	lifinance/contracts
Commit Hashes	6bc64464.....8e7b3ff5 , 2d8927aba.....9ebb89c0
Type of Project	
Audit Timeline	October 8, 2024 - October 9, 2024 November 5, 2024
Methods	Manual Review

Issues Found	
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	3
Gas Optimizations	1
Informational	5
Total Issues	11

6 Findings

6.1 Medium Risk

6.1.1 Validate msg.value in startBridgeTokensViaGasZip function

Context: [GasZipFacet.sol#L35](#)

Description: The startBridgeTokensViaGasZip of GasZipFacet bridges gas tokens using the gas.zip protocol. This function has two input parameters, a LI.FI specific _bridgeData and gas.zip specific _gasZipData.

The function should move the user's msg.value to the _gasZipData.receiver on the _gasZipData.destinationChains. But the GasZipFacet, instead of the msg.value, uses the _bridgeData.minAmount, which is not validated as equal to the msg.value. Hence, the user could use this function to drain the native tokens on the diamond by initiating a bridge without sending in msg.value.

Severity: MEDIUM (Impact: HIGH (loss of stuck funds in Diamond) + Likelihood: LOW (diamond will only hold locked funds))

Recommendation: Consider validating if msg.value equals bridgeData.minAmount in case of the startBridgeTokensViaGasZip function

LI.FI: Fixed in [eac03e1653fdec1b651e36b7b53ee56b1aa71062](#)

Researcher: Verified.

6.1.2 Redundant receiverAddress checks in startBridgeTokensViaGasZip and swapAndStartBridgeTokensViaGasZip functions

Context: [GasZipFacet.sol#L43](#)

Description: The functions startBridgeTokensViaGasZip and swapAndStartBridgeTokensViaGasZip bridge gas tokens using the gas.zip protocol. The function accepts two / three input parameters, including the ILiFi.BridgeData and the IGasZip.GasZipData. The ILiFi.BridgeData is validated using the validateBridgeData modifier inherited from the Validatable contract.

However, this validation is not sufficient as the validateBridgeData checks only if BridgeData.receiver is address(0) and there are no validations for the GasZipData.receiver which is the actual receiver address passed on to the gas.zip protocol, and can be address(0)

Severity: MEDIUM (Impact: High (false sense of security + permanent loss of funds) + Likelihood: Low (user error))

Recommendation: Consider making the following changes to the startBridgeTokensViaGasZip and swapAndStartBridgeTokensViaGasZip functions:

- Validate IGasZip.GasZipData.receiver to be not equal to bytes32(0).
- The validateBridgeData is not super important and can be optimized to only validate necessary parameters.

LI.FI: Fixed in [c000a46221ccabe51e4f6bd09140b362c6061262](#).

Researcher: Verified.

6.2 Low Risk

6.2.1 receiverAddress validation in _startBridge is redundant

Context: [GasZipFacet.sol#L107](#)

Description: The receiverAddress in the gas _gasZipData and _bridgeData are validated to be equal in case of evm-specific bridging. For non-evm targets, such a validation is avoided.

However, the nature of evm-specific bridging is identified from the `_bridgeData.receiver`, which is a user-specified address. Therefore, a user can specify `_bridgeData.receiver` to be `NON_EVM_RECEIVER_IDENTIFIER` and bypass this check.

Recommendation: Consider implementing an alternative approach to validate the receiver for both evm and non-evm paths.

LI.FI: `NON_EVM_RECEIVER_IDENTIFIER` is an identifier that tells the user/integrator that `_bridgeData.receiver` is not the real receiver, so they have to look somewhere else. `_gasZipData.receiverAddress` has to be checked by the user in this case, and if it contains an EVM address, that should be no problem.

Researcher: Consider mentioning this in the inline-documentation for improved transparency.

6.2.2 Assert balances post swap in `depositToGasZipERC20` function

Context: [GasZipPeriphery.sol#L53](#)

Description: The `depositToGasZipERC20` function calls the LI.FI diamond to process a swap before bridging them using the gas zip router.

This function operates under the following assumptions:

- liFiDEXAggregator consumes all the `_swapData.fromAmount` moved from the user to the `gasZipPeriphery` contract
- liFiDEXAggregator always swaps to a native token
- liFiDEXAggregator always returns an `uint256` that matches the native tokens deposited to the `gasZipPeriphery` post-swap
- The final recipient of the swap is `gasZipPeriphery`.

However, all these assumptions may / may not hold under all scenarios and can lead to unexpected behavior.

Recommendation: Consider adding the following balance checks to the `depositToGasZipERC20` function:

1. Assigning the `sendingAssetId` balance post-swap and refunding any remaining funds in the `gasZipPeriphery` to the user.
2. Asserting if the contract's native balance equals `swapOutputAmount` post-swap.

LI.FI: Partially fixed in [26ae13cfb1904403e85a3b1c39f3253720a9452a](#). Our assumptions are described right. At the same time, in case we messed up, we should be able to refund the user since we've some degree of control over the external contract (liFiDEXAggregator)

Researcher: The `WithdrawPeriphery` added will allow recovery of any locked funds if they're not consumed by another user yet. So in general, the issue still exists and additional caution should be exercised by the users while generating their transaction data.

6.2.3 Validate if `finalTokenId` of `_swapData` is native in `swapAndStartBridgeTokensViaGasZip` function

Context: [GasZipFacet.sol#L61](#)

Description: The `swapAndStartBridgeTokensViaGasZip` function swaps an external ERC20 token to native tokens before bridging them using the `gas.zip` protocol.

The swap output is overridden as the `_bridgeData.minAmount` and passed on to the `gas.zip` protocol. However, no validations are performed if the final token post-swaps is a native token, which can lead to unexpected behavior (Utilizing the diamond's native tokens by locking other ERC20 tokens).

Recommendation: Consider validating the final token post swaps before bringing the following:

```

function swapAndStartBridgeTokensViaGasZip(
  ILiFi.BridgeData memory _bridgeData,
  LibSwap.SwapData[] calldata _swapData,
  IGasZip.GasZipData calldata _gasZipData
) external payable
  nonReentrant
  refundExcessNative(payable(msg.sender))
  containsSourceSwaps(_bridgeData)
  doesNotContainDestinationCalls(_bridgeData)
  validateBridgeData(_bridgeData)
{
  ....
+   if(!LibAsset.isNativeAsset(_swapData[_swapData.length - 1].receivingAssetId))
+     revert OnlyNativeAllowed();
  ....
}

```

LI.FI: Fixed in [29106c532101d34216ca77376bcf526ae6c7a793](#).

Researcher: Verified.

6.3 Gas Optimization

6.3.1 `getDestinationChainsValue` function could be optimized

Context: [GasZipFacet.sol#L102](#), [GasZipPeriphery.sol#L95](#)

Description: The `getDestinationChainsValue` function is an external helper function that accepts an array of `_chainIds` and casts them into a single `uint256` variable.

This function could be further optimized by making the following changes:

- convert the input array from memory to calldata as the values are read-only
- cache the `chainIds` length
- change the visibility identifier from `public` to `external` (won't save gas)
- pre-increment the loop variable `i`

Recommendation: Consider optimizing the `getDestinationChainsValue` function as following:

```

function getDestinationChainsValue(
  uint8[] calldata _chainIds
) external pure returns (uint256 destinationChains) {
  uint256 length = _chainIds.length;
  require(length <= 32, "Too many chain IDs");

  for (uint256 i; i < length; ++i) {
    // Shift destinationChains left by 8 bits and add the next chainID
    destinationChains =
      (destinationChains << 8) |
      uint256(_chainIds[i]);
  }
}

```

LI.FI: Fixed in [193afc4e5dd2cc8d8b9284e9193cf6caeb458ff3](#)

Researcher: Verified.

6.4 Informational

6.4.1 Remove unused `NativeAssetTransferFailed` error import

Context: [GasZipPeriphery.sol#L14](#)

Description: The `GasZipPeriphery` file imports the `NativeAssetTransferFailed` error but is not used anywhere in the codebase as the native transfer failure handling is delegated to the `SafeTransferLib`.

Recommendation: Consider removing the unused error import

LI.FI: Fixed in [82bcecc36ca83728d914be8af31b023fb79ee7e2](#)

Researcher: Fixes verified.

6.4.2 Revert if `_swapData.sendingAssetId` is `NATIVE` in `depositToGasZipERC20` function

Context: [GasZipPeriphery.sol#L37](#)

Description: The `depositToGasZipERC20` is non-payable to avoid using native tokens to be swapped. However, if the `_swapData.amount` is passed in as zero, the function allows `NATIVE` tokens to be passed in as `_swapData.sendingAssetId`, which is not the intended behavior.

Recommendation: Consider adding an explicit revert, if the `_swapData.sendingAssetId` is `NATIVE`

LI.FI: The contract is designed not to hold any funds. If one tried to sneak in `swapData` that swaps from native, but the transaction had no message value, then the transaction would fail as it would run out of funds. If this case is possible at all, then it's a very low-probability edge case, and we acknowledge it and prefer to save gas.

Researcher: Acknowledged.

6.4.3 Validate `_gasZipData.receiver` in `depositToGasZipNative` function

Context: [GasZipPeriphery.sol#L77](#)

Description: The receiver is validated to be not equal to `address(0)` in the `GasZipFacet`. However, similar validations are not made in the `GasZipPeriphery` contract.

Recommendation: Consider validating if `_gasZipData.receiver` is not `address(0)` in `depositToGasZipNative` function.

LI.FI: Fixed in [bfa147fdd1b90a5dcd631d543491e149c17063a9](#)

Researcher: Verified.

6.4.4 Replace `require` with custom error inside `getDestinationChainsValue` function

Context: [GasZipFacet.sol#L105](#)

Description: The `require` check inside `getDestinationChainsValue` uses the literal string "Too many chain IDs." Literal strings for error messages are more gas-intensive than custom error definitions and must be consistent with the overall code base.

Recommendation: Consider replacing the `require` with a custom error like `TooManyChainIDs()` for better gas efficiency and code quality.

LI.FI: Fixed in [098847226ed5bae11ffd372df60035cc5ef56fd3](#)

Researcher: Verified.

6.4.5 Remove unused error OnlySwapsFromERC20ToNativeAllowed

Context: [GasZipFacet.sol#L20](#)

Description: The error OnlySwapsFromERC20ToNativeAllowed defined in GasZipFacet is unused and can be removed.

Recommendation: Remove OnlySwapsFromERC20ToNativeAllowed error from GasZipFacet.

LI.FI: Fixed in [883b3f4b2efcb9633044d12c21b82ce8eb1360dc](#)

Researcher: Verified

7 Appendix

The initial audit of the files GasZipFacet, GasZipPeriphery, and IGasZip was conducted from October 8th to October 9th, based on the commit hash 6bc644642db2d38d2a55a4e18ecd2c478e7b3ff5. Following the initial review, a re-audit was performed on these files using the updated commit hash 2d8927ababff6ace0c577d92407bcf289ebb89c0.

The main scope of the re-audit included the implementation of safeTransferETH for transferring native tokens, a variable name update within IGasZip, and the addition of a new constant, NON-EVM-RECEIVER-IDENTIFIER.