BURRA SEC

LI.FI (RelayDepositoryFacet v1.0) Security Review

Reviewed by: Goran Vladika

25 August, 2025

LI.FI (RelayDepositoryFacet v1.0) Security Review Report

Burra Security

August 25, 2025

Introduction

A time-boxed security review of the **LI.FI** protocol was done by **Burra Security** team, focusing on the security aspects of the smart contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About RelayDepositoryFacet v1.0

The RelayDepositoryFacet enables direct deposits of assets into Relay Protocol V2 Depositories. This facet supports both native tokens and ERC20 tokens, with optional swap functionality before depositing.

About Burra Security

Burra Sec offers security auditing and advisory services with a special focus on cross-chain and inter-operability protocols and their integrations.

Security review team

Goran Vladika is a security researcher and smart contract engineer with five years of experience in the blockchain industry. After beginning his Web3 career in the DeFi space, Goran joined Offchain Labs as a blockchain engineer, where he contributed to the core smart contract components of Arbitrum. His work included the design, implementation, and security of Arbitrum's native bridge, token bridge and rollup stack, critical infrastructure that secures billions of dollars in TVL. This bridging technology has since been adopted by dozens of applications and L2 and L3 chains built using the Arbitrum Orbit stack. Goran's experience building cross-chain systems at both the protocol and application layers has provided him with a strong foundation in blockchain security. As a security researcher, he has helped secure leading projects in the interoperability space including Centrifuge, LiFi, PancakeSwap, ZetaChain and DODO, as well as L1/L2 protocols such as Telcoin and Citrea.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

***review commit hash* - 4837b45b3ae705dee625015b1a084b3c69dee2a2**

Scope

The following smart contracts were in the scope of the audit:

- src/Facets/RelayDepositoryFacet.sol
- src/interfaces/IRelayDepository.sol

Findings Summary

ID	Title	Severity	Status
I-01	No need to inherit LiFiData	Info	Resolved
I-02	Considerations for the BridgeData's receiver and destinationChain	Info	Ack
I-03	User can end up overpaying transfer, but will be refunded on destination chain	Info	Ack

Detailed Findings

[I-01] No need to inherit LiFiData

Target

- RelayDepositoryFacet.sol#L23

Severity

INFO

Description

Facet contract `RelayDepositoryFacet` inherits `LiFiData`, even though it does not use any of the constants declared there. This unnecessary bloats the contract size.

Recommendation

Remove `LiFiData` from the inheritance list

LI.FI

fixed https://github.com/lifinance/contracts/commit/63a08b7a9e33eb2c081af45987981a06aa590445

BurraSec

Verified

[I-02] Considerations for the BridgeData's receiver and destinationChain

Target

- RelayDepositoryFacet.sol

Severity

INFO

Description

The main difference between `RelayDepositoryFacet` and other facets is in how provided `BridgeData` is used. Typically, `receiver` and `destinationChain` are forwarded to the integrated protocol call. In the case of `RelayDepositoryFacet` those bridging details are stored off-chain, and facet simply receives and forwards the `orderId` which is the identifier for the off-chain data. That means bridgeData's `receiver` and `destinationChain` are only used in `LiFiTransferStarted` event.

There are couple of considerations for your off-chain infra which consumes the `LiFiTransferStarted` and similar events.

In case you use bridgeData `receiver` and `destinationChain` to store the actual transfer info: - document that there is no guarantee that `receiver` and `destinationChain` emitted match the actual off-chain transfer data - user can bypass the API and provide fake data - consider if it is useful to emit `BridgeToNonEVMChain` in case when `RelayDepository` is used to bridge to Solana (determined by provided `destinationChain`) - this would require additional param `nonEvmReceiver` in `RelayDepositoryData` - there would still be no guarantee that emitted receiver matches the actual receiver, nor that Solana is the actual desination

On the other hand, if you will use bridgeData to always set `receiver = depositor` and `destinationChain = 0` then you could simplify the code: - remove custom `depositorAddress` param since `bridgeData.receiver` is sufficient and can be used in depository call - semantically it would mean `receiver` receives the relay deposit credits in the `RelayDepository`. It's up to user how deposit is used later - in this case `validateBridgeData` already validates that receiver

address is not zero, so no need for extra check - you can enforce `destinationChain == 0` (meaning “deposit destination is always source chain itself”)

Recommendation

Decide which `bridgeData` approach makes more sense considering the backend needs and implement any adjustments if needed.

LI.FI

acknowledged (and added some comments in contract and docs file)

https://github.com/lifinance/contracts/commit/9d86e488b6f651205605dd65cb0086caed8507c3

BurraSec

Acknowledged

[I-03] User can end up overpaying transfer, but will be refunded on destination chain

Target

- RelayDepositoryFacet.sol

Severity

INFO

Description

Let’s say LiFi’s backend generates a route for user which uses `Relay` protocol to bridge 5 arbUSDC for 4.9 baseUSDC. That means Relay’s API generated an order with those details and sent the `orderId` to the LiFi backend. This `orderId` will be provided in a call to the `RelayDepositoryFacet`. Additionally, let’s say LiFi backend generates preswap step to swap from ETH to USDC since user has only ETH.

Swap execution is configured to yield at least 5 arbUSDC. But it will likely yield more than the provided minimum, let's say it results with 6 arbUSDC due to positive slippage. That means 6 arbUSDC will be sent to the `RelayDepository`, even though off-chain order specifies 5 arbUSDC input. In other words, the transfer is overpaid. This would be a problem if the difference is not refunded, as user would effectively lose 1 arbUSDC.

But, even though Relay docs don't explicitly mention the overpayment case in refund scenarios, I tested this in the production environment and it turns out that Relay solver does notice when amount is overpaid and it delivers it to the destination chain nevertheless, along with output specified in the off-chain order. This is the case for both native asset and token bridging.

Recommendation

Monitor the `RelayDepository` transfers where provided `minAmount` is not equal to the original `minAmount` due to swap execution outcome. If Relay solver's implementation would change to not refund the overpaid difference, then it'd make sense to change the facet implementation to return the difference to the user directly.

LI.FI

acknowledged (added comments in contract and docs)

https://github.com/lifinance/contracts/commit/898c77f65632565d1de12013e08dd61681335b32

BurraSec

Acknowledged