BURRA SEC

Centrifuge Security Review

Reviewed by: Goran Vladika, Pranav Garg

April 9th - April 15th, 2025

Centrifuge Security Review Report

Burra Security

April 25, 2025

Introduction

A time-boxed security review of the **Centrifuge** 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 Burra Security

Burra Security offers security auditing and advisory services with a special focus on cross-chain and interoperability protocols and their integrations.

About Centrifuge

Centrifuge V3 is an open, decentralized protocol for onchain asset management. Built on immutable smart contracts, it enables permissionless deployment of customizable tokenization products.

Using protocol-level chain abstraction, tokenization issuers access liquidity across any network, all managed from one Hub chain of their choice.

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* - af76d5a74c9c575572c66e4a1c10ba1b182032f5**

Scope

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

- Gateway.sol
- WormholeAdapter.sol
- AxelarAdapter.sol
- GasService.sol

Findings Summary

ID	Title	Severity	Status
H-01	Gateway is used as the refund receiver, but lacks ability to receive ETH	High	Resolved
M-01	User is not refunded the difference between msg.value and the actual cost	Medium	Resolved
L-01	Executing failed messages should be disabled when system is paused	Low	Resolved
I-01	Not resetting the transient storage vars can cause issues	Info	Resolved
I-02	Ensure the correctness of refundWormholeId	Info	Resolved
I-03	Mismatch between SendProof and ProcessProof events	Info	Resolved

Detailed Findings

[H-01] Gateway is used as the refund receiver, but lacks ability to receive ETH

Target

- Gateway.sol

Severity

- Impact: Medium
- Likelihood: High

Description

When gateway calls adapter to send the cross-chain message, it sets itself as a refund receiver. Both Wormhole and Axelar have similar refund mechanisms - user pays on the source chain for a certain amount of gas to be spent on the destination chain. If actual gas consumption on the destination chain turns out to be lower, the payment for unspent gas will be refunded by relayer to the refund receiver. That's where the issue is, `Gateway` contract is the refund receiver, but since it does not implement `receive()` function it can't receive the refunded ETH.

And the likelihood of refund being triggered for every message is quite high due to the current design of `GasService`. It uses immutable gasLimit for every message type, no matter how much gas is required to process the specific message. The `gasLimit` value has to be set such that it covers the execution of the most gas-intensive message. So for the most messages refund mechanism will be triggered, and the funds to be refunded will be effectively lost for the protocol.

Recommendation

Add `receive` function to the Gateway and track senders to be able to identify refunds.

```
    receive() external payable {
        emit Received(msg.sender, msg.value);
    }
```

BurraSec

Fix in PR#253 looks good, gateway can now receive direct transfers

[M-01] User is not refunded the difference between msg.value and the actual cost

Target

- Gateway.sol

Severity

- Impact: Medium
- Likelihood: Medium

Description

When doing the cross-chain calls, user provides ETH along with the call to pay for the message or the whole batch. `Estimate(...)` functions can be used off-chain to figure out the amount of fees to provide. However, gas price can fluctuate and change between estimation and the time of the actual TX execution. In case the fees turn out to be overpaid, the difference between msg.value and the actual

cost is not refunded back to the user. Overpaid fees stay in the `Gateway` contract where they can only be recovered by the admin actions.

Recommendation

Send back the remaining `fuel` to the user after the message or the batch are sent out.

BurraSec

Fix in PR#253 looks good, remaining fuel is sent to TX payer

[L-01] Executing failed messages should be disabled when system is paused

Target

- Gateway.sol

Severity

- Impact: Medium
- Likelihood: Low

Description

In case of an emergency, system can be paused which will in turn disable sending and receiving of cross-chain messages. However, executing the previously failed messages via retry function won't be disabled. That fact can be possibly exploited by an attacker. Attacker could pre-plant a failing message, then wait for an emergency situation where message could be re-tried and cause harm to the protocol.

Recommendation

Make `retry` function pausable.

Client

Fixed in https://github.com/centrifuge/protocol-v3/pull/254/files

BurraSec

Fix looks good

[I-01] Not resetting the transient storage vars can cause issues

Target

- Gateway.sol

Severity

- Impact: Low
- Likelihood: Low

Description

Gateway uses transient storage to handle the batch message processing. At the end of the batch, some transient vars are reset to default values, but `paymentMethod` is not. That's not an issue in a typical case where ie. `VaultRouter`'s `multicall` is the top level call of the transaction. But that's not necessary the case. Let's assume Alice and Bob are Centrifuge users who use the account abstraction wallets. Following bad outcomes can happen: 1. Alice wants to send batch, Bob wants to send subsidised msg. Alice's and Bob's operations get bundled in the same TX. Bob's operation will fail (no fuel) because `PaymentMethod.Transaction` will be left set after Alice's operation, even though Bob wanted to use the default subsidised mode. 2. Alice wants to send batch, overpays for the fees. If Alice's and Bob's operation end up bundled together Bob can piggyback off of Alice's leftover fuel and use it to send his message.

Recommendation

Reset all the transient storage vars to their original values.

BurraSec

Fix in PR#253 looks good, all transient vars are reset after msg/batch is processed.

[I-02] Ensure the correctness of refundWormholeId

Target

- WormholeAdapter.sol

Severity

Info

Description

`WormholeAdapter` sets the `refundWormholeId` in the constructor. That should always be the wormhole ID of the current chain, because refund address is the Centrifuge gateway of that chain. In order to avoid misinterpreting the `refundWormholeId` it could be automatically fetched on-chain: `relayer` -> `getDefaultDeliveryProvider()` -> `chainId()`. Alternatively, you could add docs/comments clarifying that `refundWormholeId` should always be the current chain.

BurraSec

Fix in PR#253 looks good

[I-03] Mismatch between SendProof and ProcessProof events

Target

- Gateway.sol

Severity

INFO

Description

Events `SendProof` and `ProcessProof` are used to track and match messages between source and destination chain. But there is a mismatch - sending will emit serialized proof, while receiving will emit deserialized proof (hash without the prefix). Same format should be used for consistency and to avoid issues in off-chain processing.

Client

Fixed by #260

BurraSec

Fix looks good