Centrifuge:

Protocol v3.1.0 - December 2025

security review

reviewed by:

xmxanuel
cantina.xyz/u/xmxanuel

16th December 2025

1. Executive Summary

The Centrifuge team has asked xmxanuel to review only the fixes in their Solidity codebase for protocol v3.1.0 that relate to findings from the Sherlock competition concerning cross-chain pool attacks.

2. Disclaimer

This security review report is provided "as is" and for informational purposes only. The purpose of this report is to assist the client in identifying potential security vulnerabilities in the reviewed code, based on the scope and methodology agreed upon. It does not constitute legal, investment, financial, or any other form of professional advice.

No warranties or guarantees are made regarding the completeness, accuracy, or security of the reviewed code. The author and any affiliated parties make no representations or warranties, express or implied, and expressly disclaim any liability or responsibility for any loss, damage, or other consequence arising from or related to the use of, reliance on, or inability to use this report or the reviewed code. This includes, without limitation, direct, indirect, incidental, consequential, special, exemplary, or punitive damages, even if advised of the possibility of such damages.

This report reflects the results of a best-effort security review conducted within the scope and time constraints agreed upon with the client. Although an attempt was made to identify potential security issues, there is no assurance that all vulnerabilities have been discovered. A smart contract security review cannot be considered a guarantee or certification of security. Users and project teams remain solely responsible for the use, deployment, and management of any smart contracts.

This report does not constitute an endorsement or recommendation of the reviewed project. Third parties should not rely on this report and are strongly encouraged to perform their own independent assessments.

3. Repository

https://github.com/centrifuge/protocol-v3/

Version	Commit Hash	Date	Note
Centrifuge Review Commit Hash	49b93e129a9b8c2c1a127c7376bc2194afa6ee4c	4th December 2025	review version
Fix Version: v3.1.0	6b9d36eabee48728486f377ea2766a5cd233c555	13th January 2026	fix version

4. Findings

The following findings were identified during the security review of the codebase. Each finding is categorized by severity level based on its potential impact and includes a detailed description and recommended remediation.

Summary

Severity	ID	Title	Status
Low	L1	Malicious ERC20 token can steal entire pool escrow via BalanceSheet reentrancy	Fixed
Low	L2	Malicious snapshotHook enables reentrancy-based privilege escalation via `_sender` cache	Fixed
Informational	I1	BalanceSheet multicall opens gateway batch unnecessarily	Acknowledged
Informational	I2	ProtocolGuardian `setAdapters` bypasses timelock security model	Fixed
Informational	I3	BatchedMulticall + Gateway as ward would allow anyone to call auth functions	Acknowledged

Medium Findings

Low Findings

L1. Malicious ERC20 token can steal entire pool escrow via BalanceSheet reentrancy

BalanceSheet.sol#L113-L134

Description: The `BalanceSheet` contract inherits `BatchedMulticall`, enabling a `_sender` caching vulnerability. Any untrusted 3rd party contract triggered by the multicall could re-enter the contract under manager privileges.

Combined with the `withdraw()` function's external ERC20 transfer, this creates a fund theft vector through malicious or compromised tokens.

```
function withdraw(
    PoolId poolId,
    ShareClassId scId,
    address asset,
    uint256 tokenId,
    address receiver,
    uint128 amount,
    bool wasNoted
) external payable isManager(poolId) {
    IPoolEscrow escrow_ = escrow(poolId);

    if (wasNoted) {
        escrow_.withdraw(scId, asset, tokenId, receiver, amount);
        // ... accounting updates
    }

    escrow_.authTransferTo(asset, tokenId, receiver, amount);  // External call!
}
```

The `authTransferTo()` call triggers `SafeTransferLib.safeTransfer()`, which calls the token's `transfer()` function. A compromised ERC20 token implementation could re-enter.

Attack Scenario:

1. Pool holds 99% USDC + 1% upgradeable EUR stablecoin
2. The EUR stablecoin is compromised via malicious upgrade
3. Manager calls `BalanceSheet.multicall([withdraw(eurToken, ...)])`
4. `_sender = manager` is cached in transient storage
5. `withdraw()` calls `escrow.authTransferTo(eurToken, ...)` → `eurToken.transfer()`
6. Malicious token re-enters: `BalanceSheet.withdraw(USDC, attacker, usdcBalance, wasNoted=false)`
7. `isManager()` checks `msgSender()` → returns `_sender = manager` (still cached!)
8. `wasNoted=false` bypasses reserve accounting
9. `authTransferTo(USDC, attacker, ...)` → **99% of pool USDC stolen**

A single compromised or malicious ERC20 token can drain ALL other assets from a pool's escrow. The manager does not need to be malicious; they are simply withdrawing a token that turned out to be compromised.

This is particularly relevant for pools holding upgradeable stablecoins or tokens with proxy patterns.

Recommendation: Block re-entrancy from 3rd party contracts in the multicall pattern with `msgSender()` by only using the cached `_sender` if the caller is the `Gateway`.

```
function msgSender() internal view virtual returns (address) {
    return _sender != address(0) && msg.sender == address(gateway) ? _sender : msg.sender;
}
```

If another Centrifuge protocol contract needs to call specific Hub or BalanceSheet functions during the multicall flow, it should get extra permission via the manager role.

Centrifuge: Fixed. Added gateway check to `msgSender()`. See PR #78.

xmxanuel: Fixed.

L2. Malicious snapshotHook enables reentrancy-based privilege escalation via `_sender` cache

BatchedMulticall.sol#L42-L44

Description: The `BatchedMulticall` pattern caches the original caller in transient storage (`_sender`) during multicall execution. The `msgSender()` function returns this cached sender whenever `_sender` is set:

```
function msgSender() internal view virtual returns (address) {
    return _sender != address(0) ? _sender : msg.sender;
}
```

This design allows any external contract called during a multicall to inherit the original caller's privileges when calling back into Hub functions.

A malicious pool manager can exploit this by deploying a backdoored `snapshotHook` that, for example, re-adds them as manager after being removed.

Attack Flow:

1. Alice (pool manager) deploys a `MaliciousSnapshotHook` with a backdoor
2. Alice sets the hook: `Hub.setSnapshotHook(poolId, maliciousHook)`
3. Bob (another manager) removes Alice: `Hub.updateHubManager(poolId, Alice, false)`
4. Bob calls `Hub.multicall([updateHoldingValue(...)])`
    - `_sender = Bob` is cached in transient storage
    - `updateHoldingValue()` triggers `Holdings.callOnSyncSnapshot()`
    - `maliciousHook.onSync()` is called (note: NOT a view function)
    - Hook re-enters: `Hub.updateHubManager(poolId, Alice, true)`
    - `_isManager()` checks `msgSender()` which returns `_sender = Bob`
    - Bob is a manager, check passes → **Alice re-added as manager**

A malicious manager can plant a persistent backdoor that allows them to regain manager privileges whenever any other manager triggers a multicall operation that calls `callOnSyncSnapshot`. This undermines the trust model where removing a manager should revoke their access.

The `msgSender()` function does not verify that the caller is actually the `gateway` (the expected callback path); it returns `_sender` for any caller as long as `_sender` is set.

Recommendation: Add a check to ensure `msgSender()` only returns `_sender` when the call originates from the gateway callback:

```
function msgSender() internal view virtual returns (address) {
    return _sender != address(0) && msg.sender == address(gateway) ? _sender : msg.sender;
```

```
}
```

If another Centrifuge protocol contract needs to call specific Hub or BalanceSheet functions during the multicall flow, it should get extra permission via the manager role.

Centrifuge: Fixed. Added gateway check to `msgSender()`. See PR #78.

xmxanuel: Fixed.

Informational

I1. BalanceSheet multicall opens gateway batch unnecessarily

BalanceSheet.sol#L38

`BalanceSheet` inherits `BatchedMulticall`, which always opens a `Gateway.withBatch()` flow on every `multicall()` call. However, many BalanceSheet functions (e.g., `deposit`, `withdraw`, `issue`, `revoke`) don't send cross-chain messages.

This results in unnecessary gas overhead when batching local-only operations.

Recommendation: Consider whether BalanceSheet needs `BatchedMulticall` inheritance, or implement a lighter multicall variant for local-only operations.

Centrifuge: Acknowledged.

xmxanuel: Acknowledged.

I2. ProtocolGuardian `setAdapters` bypasses timelock security model

ProtocolGuardian.sol#L132-L137

The protocol uses a timelock for `Root.scheduleRely` actions, but `ProtocolGuardian.setAdapters` can set global pool adapters without any timelock:

```
function setAdapters(...) external onlySafe {
    multiAdapter.setAdapters(centrifugeId, GLOBAL_POOL, adapters, threshold, recoveryIndex);
}
```

If the Guardian Safe is compromised, an attacker gains full protocol access by setting malicious adapters - bypassing the timelock security model entirely.

Recommendation: Consider adding a timelock to `setAdapters` for consistency with other privileged operations.

Centrifuge: Fixed. Removed `setAdapters` from ProtocolGuardian (will require spells). See PR#79.

xmxanuel: Fixed.

I3. BatchedMulticall + Gateway as ward would allow anyone to call auth functions

Multicall.sol#L16

Theoretical vulnerability: If a contract inherits `BatchedMulticall` and has Gateway as a ward, anyone could call `auth`-protected functions.

Attack flow:

1. Attacker calls `Contract.multicall([authFunction()])`
2. `multicall` → `gateway.withBatch()` → Gateway calls back `executeMulticall`

3. `executeMulticall` → `super.multicall()` uses `delegatecall`
4. With delegatecall, `msg.sender` == Gateway
5. Auth check passes because Gateway is ward

No `BatchedMulticall` contracts (Hub, BalanceSheet, VaultRouter, BatchRequestManager) currently have Gateway as ward. This is theoretical but could be introduced accidentally.

Recommendation: Document this pattern as a security invariant: never add Gateway as ward to BatchedMulticall contracts.

Centrifuge: Acknowledged.

xmxanuel: Acknowledged.