yAudit

January 18, 2026

Prepared for
Centrifuge

Audited by
HHK
Watermelon

Centrifuge v3.1 Fix Review

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

Centrifuge v3.1 implements a decentralized protocol for on-chain asset management. It provides foundations for permissionless deployment and management of highly customizable tokenization solutions.

1.2 Audit Scope

This audit covers 20 pull requests across 4 engineering days. The client's pull requests were reviewed in two separate batches: the first 16 pull requests were reviewed between December 3rd, 2025 and December 5th, 2025. The remaining 4 pull requests were submitted at a later date by the client and reviewed by the yAudit team on January 16th, 2026.

1.3 Risk Assessment Framework

1.3.1 Severity Classification

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
Undetermined	Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Increased transaction costs
Informational	Code quality and best practice recommendations	Reduced maintainability and readability

Table 1: severity classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	5

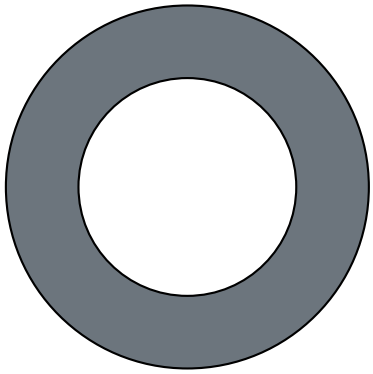

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

Centrifgue v3.1 represents a mature codebase with strong focus on its security stance. The fixes implemented to address findings identified in Centrifuge v3.1's audit competition were found to be correctly implemented, with researchers identifying only minor gas optimizations and styling recommendations to apply a final layer of polish to the codebase.

2 Audit Overview

2.1 Project Information

Protocol Name: Centrifuge
Repository: https://github.com/centrifuge/protocol-internal/
Commit Hashes:

- 34ff1f82806fb22680ef694eadca62fb86b62679
- 6b9d36eabee48728486f377ea2766a5cd233c555

Commit URLs:

- commit 34ff1f82
- PR 10
- PR 14
- PR 21
- PR 31
- PR 33
- PR 36

- PR 42
- PR 48
- PR 49
- PR 50
- PR 56
- PR 58
- PR 59
- PR 62
- PR 64
- PR 65
- commit 6b9d36ea
- PR 78
- PR 98
- PR 100
- PR 102

2.2 Audit Team

HHK, Watermelon

2.3 Audit Timeline

The audit was conducted from December 03, 2025 to January 18, 2026.

2.4 Audit Resources

Code repositories and documentation

2.5 Critical Findings

None.

2.6 High Findings

None.

2.7 Medium Findings

None.

2.8 Low Findings

None.

2.9 Gas Savings Findings

2.9.1 Cache `reservedBy` mapping to avoid duplicate storage reads

Technical Details

The `reserve()` and `unreserve()` functions load the `reservedBy` mapping twice when they could load it once by declaring `newReservedAmount` earlier in the function.

Impact

Gas savings.

Recommendation

Declare `newReservedAmount` earlier and use it instead of incrementing.

```
function reserve(...) external auth {
+   uint128 newReservedAmount = reservedBy[scId][caller][reason][asset][tokenId] + value;
-   reservedBy[scId][caller][reason][asset][tokenId] += value;
+   reservedBy[scId][caller][reason][asset][tokenId] = newReservedAmount;
    holding_.reserved += value;

-   uint128 newReservedAmount = reservedBy[scId][caller][reason][asset][tokenId];
    emit IncreaseReserve(asset, tokenId, poolId, scId, caller, reason, value, newReservedAmount);
}
```

```
function unreserve(...) external auth {
+   uint128 newReservedAmount = reservedBy[scId][caller][reason][asset][tokenId] - value;
-   reservedBy[scId][caller][reason][asset][tokenId] -= value;
+   reservedBy[scId][caller][reason][asset][tokenId] = newReservedAmount;
    holding_.reserved -= value;

-   uint128 newReservedAmount = reservedBy[scId][caller][reason][asset][tokenId];
    emit DecreaseReserve(asset, tokenId, poolId, scId, caller, reason, value, newReservedAmount);
}
```

Developer Response

Fixed by 85efcfc.

2.9.2 Salt reuse check may be executed earlier to save gas in revert case

Technical Details

`ShareClassManager.addShareClass` now requires the `salt` parameter's first 8 bytes to match the provided `poolId` value, in order to prevent `salt` values previously used for a pool from being used for a different pool.
The method also ensures that a given `salt` is used only once, by storing used values in its `ShareClassManager.salts` mapping and checking a new `salt` against such set: if the `salt` has already been used, a `AlreadyUsedSalt` error is returned.
Both checks may be moved to the beginning of the method, in order to marginally reduce the gas consumed by the method in both unhappy paths.

Impact

Gas optimization.

Recommendation

Move the code related to the mentioned checks to the beginning of the method:

```
@@ -38,6 +38,10 @@ contract ShareClassManager is Auth, IShareClassManager {
          auth
          returns (ShareClassId scId_)
      {
+         PoolId prefixedPoolId = PoolId.wrap(uint64(bytes8(salt)));
+         require(poolId == prefixedPoolId, InvalidSalt());
+         require(!salts[salt], AlreadyUsedSalt());
+
          scId_ = previewNextShareClassId(poolId);

          uint32 index = ++shareClassCount[poolId];
@@ -45,9 +49,6 @@ contract ShareClassManager is Auth, IShareClassManager {

          ShareClassMetadata storage meta = _updateMetadata(poolId, scId_, name, symbol);

-         PoolId prefixedPoolId = PoolId.wrap(uint64(bytes8(salt)));
-         require(poolId == prefixedPoolId, InvalidSalt());
-         require(!salts[salt], AlreadyUsedSalt());
          salts[salt] = true;
          meta.salt = salt;
```

Developer Response

Fixed by 9eb2a2d.

2.10 Informational Findings

2.10.1 Unused named return parameter in `AsyncRequestManager.maxMint`

Technical Details

`AsyncRequestManager.maxMint` defines the `shares` named return parameter but fails to assign a value to it.

Impact

Informational.

Recommendation

Maintain consistency with the rest of the `max*` methods and assign the return value to `shares`:

```
@@ -554,7 +554,7 @@ contract AsyncRequestManager is Auth, IAsyncRequestManager,
 ITrustedContractUpda
         if (!_canTransfer(
                 vault_, address(balanceSheet.escrow(vault_.poolId())), user,
uint256(investments[vault_][user].maxMint)
             )) return 0;
-        return uint256(investments[vault_][user].maxMint);
+        shares = uint256(investments[vault_][user].maxMint);
     }

     /// @inheritdoc IRedeemManager
```

Developer Response

Fixed by 2706805.

2.10.2 Outdated README references global escrow

Technical Details

The vault README.md still mentions global escrow even though it has been replaced with pool-specific escrow, which can be confusing for future auditors and integrators:

- ”The vault enforces controller/owner validation for all operations and integrates with the global escrow for asset custody”
- ”coordinating with BalanceSheet for share issuance/burning and the global escrow for asset custody.”
- Also still present in some diagrams

Impact

Informational.

Recommendation

Update the README and diagrams to reference pool-specific escrow changes made.

Developer Response

Fixed by bd06c52.

2.10.3 Missing event inside `SubsidyManager`

Technical Details

The `SubsidyManager` contract emits a `WithdrawSubsidy` event inside `withdraw()` but not inside `withdrawAll()`.
Since both functions have similar behavior, with the main difference being that `withdrawAll()` withdraws the entire available balance, it seems expected that it would also emit the `WithdrawSubsidy` event.

Impact

Informational.

Recommendation

Emit `WithdrawSubsidy` event inside `withdrawAll()`.

```
function withdrawAll(PoolId poolId, address to) external auth returns (address,
 uint256) {
...
    refund.withdrawFunds(to, amount);
+   emit WithdrawSubsidy(poolId, to, amount);

    return (address(refund), amount);
}
```

Developer Response

Fixed by 7934e82.

2.10.4 Missing nastpec documentation for `extraGasLimit` param in `ISpoke.request`

Technical Details

`ISpoke.request` now accepts an `extraGasLimit` parameter in order for callers to provide an additional gas stipend for cross-chain execution.
The highlighted method's natspec documentation is missing a description for the `extraGasLimit` parameter.

Impact

Informational.

Recommendation

Add a description for the `extraGasLimit` to the highlighted method's documentation.

Developer Response

Fixed by 740efd3.

2.10.5 Missing event emission in `BatchRequestManager.setEpochIds`

Technical Details

`BatchRequestManager.setEpochIds` implements functionality for a `BatchRequestManager` ward to override values in `epochId` mapping during a migration from V3's `ShareClassManager` to V3.1's `BatchRequestManager`.
The method could benefit from an event emission in order for off-chain components to be able to track calls to the highlighted method.

Impact

Informational.

Recommendation

Define an `EpochIdModified(PoolId pool)` event and log it within the highlighted method:

```
@@ -94,9 +94,12 @@ contract BatchRequestManager is Auth, BatchedMulticall,
IBatchRequestManager {
         _;
     }

+    event EpochIdModified(PoolId poolId, ShareClassId scId, AssetId assetId,
EpochId epochIdData);
+
     /// @dev used only for migrations
     function setEpochIds(PoolId poolId, ShareClassId scId, AssetId assetId,
EpochId memory epochIdData) external auth {
         epochId[poolId][scId][assetId] = epochIdData;
+        emit EpochIdModified(poolId, scId, assetId, epochIdData);
     }
```

Developer Response

Fixed by 0eb5d40

2.11 Final Remarks

The reviewed pull requests were found to correctly address the related issues identified within the latest audit competition held by the Centrifuge team, uncovering a small set of issues related with gas consumption optimization and code readability.