Electisec

Prepared for
Centrifuge

Audited by
adriro
watermelon

October 18, 2025

Centrifuge v3.1 Upgrade [DRAFT]

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

Centrifuge v3 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 the v3.1 upgrade, which comprises changes to previously reviewed contracts and entirely new ones, across 7 days of review.

```
src
├── core
│   ├── hub
│   │   ├── Hub.sol
│   │   ├── HubHandler.sol
│   │   ├── HubRegistry.sol
│   │   └── ShareClassManager.sol
│   └── spoke
│       ├── Spoke.sol
│       └── VaultRegistry.sol
├── hooks
│   ├── BaseTransferHook.sol
│   ├── FreelyTransferable.sol
│   ├── FreezeOnly.sol
│   ├── FullRestrictions.sol
│   ├── interfaces
│   │   ├── IFreezable.sol
│   │   └── IMemberlist.sol
│   ├── libraries
│   │   └── UpdateRestrictionMessageLib.sol
│   └── RedemptionRestrictions.sol
├── managers
│   ├── hub
│   │   ├── interfaces
│   │   │   ├── INAVManager.sol
│   │   │   └── ISimplePriceManager.sol
│   │   ├── NAVManager.sol
│   │   └── SimplePriceManager.sol
│   └── spoke
│       └── QueueManager.sol
├── valuations
│   ├── interfaces
│   │   └── IOracleValuation.sol
│   ├── IdentityValuation.sol
│   └── OracleValuation.sol
└── vaults
```

```
├── BatchRequestManager.sol
└── SyncManager.sol
```

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.	Reduced transaction costs
Informational	Code quality and best practice recommendations	Improved maintainability and readability

Table 1: tab:severity-classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	4
Informational	4

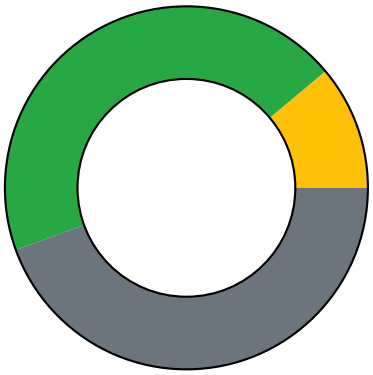

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

Centrifuge's v3.1 represents a mature, incrementally evolving protocol with a keen focus on its security stance. The protocol demonstrates strong foundations and a commitment to being secured through continuous auditing, conservative refactoring practices, and controlled feature deployment that prioritizes code quality and readability alongside functionality.

2 Audit Overview

2.1 Project Information

Protocol Name: Centrifuge
Repository: https://github.com/centrifuge/protocol
Commit Hash: 6b95a7aa94704e1c6c1cc03c0f614d599c735a71
Commit URLs:

- 6b95a7a
- PR 606
- PR 626
- PR 628
- PR 544
- PR 548
- PR 555
- PR 563
- PR 650
- PR 580

- PR 346
- PR 615
- PR 595
- PR 667
- PR 659
- PR 675
- PR 642

2.2 Audit Team

adriro, watermelon

2.3 Audit Resources

Code repositories and documentation

2.4 Critical Findings

None.

2.5 High Findings

None.

2.6 Medium Findings

2.6.1 Prevent overflow in NAV accounting

Technical Details

The calculation of the NAV is given by the `netAssetValue()` function.

```
213:         return equity + gain - loss - liability;
```

If liabilities exceed the adjusted equity, the calculation may overflow. This can cause a revert in the path submitting snapshots to the Hub, which are linked to the NAVManager.

Impact

Medium. The issue can block updates to the Hub for as long as the calculation overflows.

Recommendation

Consider clamping the calculation to zero to avoid the revert.
Note that this could also skew the final NAV after being aggregated in the SimplePriceManager contract.

Developer Response

Fixed by PR#708.

2.7 Low Findings

2.7.1 Check accounts are positive in `closeGainLoss()`

Technical Details

The implementation of `closeGainLoss()` ignores the `isPositive` return value when fetching the current state of the gain and loss accounts.

```
176:         (, uint128 gainValue) = accounting.accountValue(poolId, gainAccount_);
177:         (, uint128 lossValue) = accounting.accountValue(poolId, lossAccount_);
```

The function then proceeds to adjust the equity account based on these values. If there are gains, it debits the gain account and credits the equity. If there are losses, it credits the loss account (debit normal) and debits the equity.

Impact

Low. The accounting would be incorrect if the returned values are not positive.

Recommendation

Given the implementation assumptions, consider checking that the returned values are indeed positive.

Developer Response

Fixed in PR#728.

2.7.2 `SyncManager.maxDeposit()` uses incorrect denomination in `_canTransfer()` call

PR#675 introduces changes for `SyncManager.maxMint()` and `SyncManager.maxDeposit()` functions to take into account potential transfer restrictions imposed by a vault's share token's hooks.

Technical Details

The current `SyncManager.maxDeposit()` implementation passes the result of `SyncManager._maxDeposit`, which is an amount denominated in a vault's asset, as a parameter in a call to `SyncManager._canTransfer()`, which expects an amount of vault shares.

Impact

Low. While current `ITransferHook` implementations do not impose transfer restrictions based on the amount of share tokens being transferred, passing an asset amount instead of a share amount may hinder correct functionality in future implementations.

Recommendation

Calculate the amount of corresponding shares using `convertToShares(vault_, amount)` as is done within `SyncManager.maxMint()`.

Developer Response

Fixed by PR#723.

2.7.3 `maxRedeemClaims()` should depend on the last `redeem` epoch

Technical Details

The refactored `maxRedeemClaims()` function in BatchRequestManager uses the `revoke` field from the stored EpochId struct, but it should take the `redeem` value instead.

Impact

Low.

Recommendation

Change the `revoke` field for the `redeem` field.

Developer Response

Fixed by PR#720, which actually changes `maxDepositClaims` to use `epochId[..].issue` instead of `*.deposit` because otherwise `maxDepositClaims` returns one claimable epoch after `approveDeposits` while `notifyDeposit` fails with `IssuanceRequired` as long as the issuance is missing (i.e., `epochId.issue < epochId.deposit`). So before this fix, it incorrectly signals a possible deposit claim.

2.7.4 The `approveRedeems()` function can be non-payable

Technical Details

The `approveRedeems()` function doesn't need to be payable since it doesn't dispatch a message.

Impact

Low.

Recommendation

Remove the `payable` modifier.

Developer Response

Fixed in PR#721.

2.8 Gas Savings Findings

2.8.1 Gas savings in QueueManager

Technical Details

The expression to check if the delay has elapsed can be simplified to the third sub-expression.

```
74:         require(
75:             sc.lastSync == 0 || sc.minDelay == 0 || block.timestamp >= sc.lastSync +
 sc.minDelay, MinDelayNotElapsed()
76:         );
```

For the duplicate check in the `assetIds` array, the `scId` argument can be removed, as this is constant for all elements. The validation can be simplified by ensuring asset IDs are in ascending order instead of using temporal storage. The check can also be removed because calling `submitQueuedAssets()` would blank deposits and withdrawals, causing the same asset ID to be skipped if repeated.

```
80:         for (uint256 i = 0; i < assetIds.length; i++) {
81:             bytes32 key = keccak256(abi.encode(scId.raw(), assetIds[i].raw()));
82:             if (TransientStorageLib.tloadBool(key)) continue; // Skip duplicate
83:             TransientStorageLib.tstore(key, true);
84:
85:             // Check if valid
86:             (uint128 deposits, uint128 withdrawals) = balanceSheet.queuedAssets(
poolId, scId, assetIds[i]);
87:             if (deposits > 0 || withdrawals > 0) {
88:                 balanceSheet.submitQueuedAssets(poolId, scId, assetIds[i], sc.
extraGasLimit, address(0));
89:                 validCount++;
90:             }
91:         }
```

Additionally, the call to `queuedShares()` can be moved below the assets loop to fetch the updated value of `queuedAssetCounter` instead of tracking the `validCount` and then comparing it against the cached value of `queuedAssetCounter`.

Impact

Gas savings.

Recommendation

Consider implementing the recommended suggestions.

Developer Response

Fixed in PR#726.

2.9 Informational Findings

2.9.1 `HubRegistry.updateCurrency()` allows assigning non-registered currency to an existing pool

`HubRegistry.updateCurrency` may be used by an authorized address to update the `HubRegistry.currency` mapping.

Technical Details

The method fails to ensure that the new `currency_` being linked to a given `poolId_` has been previously registered via `HubRegistry.registerAsset`.

Impact

Informational.

Recommendation

Ensure `currency_` has been registered:

```
@@ -87,6 +89,7 @@ contract HubRegistry is Auth, IHubRegistry {
     function updateCurrency(PoolId poolId_, AssetId currency_) external auth {
         require(exists(poolId_), NonExistingPool(poolId_));
         require(!currency_.isNull(), EmptyCurrency());
+        require(isRegistered(currency_));

         currency[poolId_] = currency_;
```

Developer Response

Fixed as recommended in PR#724.

2.9.2 Incorrect Nastspec documentation in `OracleValuation`

OracleValuation.sol provides an implementation for trusted price oracles to update asset prices.

Technical Details

The comments at OracleValuation.sol#L20-L22 indicate that, to utilize the highlighted contract, developers must use `hub.updateFeeder()`. While Hub.sol doesn't implement such method, the correct way to integrate the contract is by using `Holdings.updateValuation`.

Impact

Informational.

Recommendation

Correct the documentation as shown.

Developer Response

Fixed in PR#725.

2.9.3 `BaseTransferHook.trustedCall()` ignores inner kind switch

Technical Details

The implementation of `trustedCall()` in BaseTransferHook doesn't switch on the `kind` field of the `UpdateContractUpdateAddress` struct, treating all messages as updates to the manager.

Impact

Informational.

Recommendation

Check if `kind` equals some constant and revert if not.

```
UpdateContractMessageLib.UpdateContractUpdateAddress memory m =
    UpdateContractMessageLib.deserializeUpdateContractUpdateAddress(payload);

if (m.kind == "manager") {
    address token = address(spoke.shareToken(poolId, scId));
    require(token != address(0), ShareTokenDoesNotExist());

    manager[token][m.what.toAddress()] = m.isEnabled;
} else {
    revert UnknownUpdateContractKind();
}
```

Developer Response

Fixed in PR#715.

2.9.4 `BaseTransferHook.isDepositRequestOrIssuance()` returns `true` for mints to `crosschainSource`

`BaseTransferHook.isDepositRequestOrIssuance()` may be used within an `ITransferHook` implementation to capture and execute arbitrary logic when vault share tokens are minted to an address different than `BaseTransferHook.depositTarget`.

Technical Details

During a cross-chain transfer's delivery on the target chain, share tokens are first minted into the `Spoke` contract and then transferred to the recipient: link.
Given that the `Spoke` contract is assigned to `BaseTransferHook.crosschainSource`, which is known to be different from `BaseTransferHook.depositTarget`, the mentioned predicate will return `true` during the share token mint.

Impact

Informational.

Recommendation

Modify the mentioned function to not return `true` when `to == crosschainSource`:

```
 @@ -109,23 +109,29 @@ abstract contract BaseTransferHook is Auth, IMemberlist,
 IFreezable, ITransferHo
     ) public view virtual returns (bool);

     function isDepositRequestOrIssuance(address from, address to) public view
returns (bool) {
-        return from == address(0) && to != depositTarget;
+        return from == address(0) && to != depositTarget && to !=
crosschainSource;
     }
```

Developer Response

Fixed in PR#725.

2.10 Final Remarks

The audit focused on both incremental updates to core and hook-related contracts from version 3.0.1, as well as new contracts introduced with version 3.1.
Updates to pre-existing contracts imply no significant change in the system's business logic; instead, they involve a refactor to improve the codebase's readability and simplicity significantly. Within the refactor, one minor issue was identified that caused an incorrect epoch to be used in the calculations for the maximum claimable redeem epoch for a given investor.
The contracts introduced in version 3.1 aim to fully implement pool net asset value calculations on-chain, using a new `IValuation` implementation that accepts asset price submissions from authorized accounts, along with two new `ISnapshotHook` implementations. Within these contracts, one medium-severity issue was identified: net asset value calculations could trigger an uncaught negative overflow, leading to failures in transactions that submit snapshot updates to the Hub.