Centrifuge: Protocol v3

security review

reviewed by:

xmxanuel
cantina.xyz/u/xmxanuel

4th April 2025

1. Executive Summary

The Centrifuge team has asked xmxanuel to review their Solidity codebase for `protocol v3` as security reviewer to give feedback on overall design and to identify security vulnerability.

The review began with version 2b657c3de80b2f1153ce64403209f2c8cb8527ac. However, during the review, a new version was provided, which included additional features. The second version has the following commitHash `adea6c40869e529210005361d54add16507b108f`

The following report only includes findings related to security and not the overall design feedback. Since the protocol has still been under the development some additional `high` findings turned out to be known issues or under development which could be verified by additional provided documentation and are not included in the report.

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
First version (`v1`)	2b657c3de80b2f1153ce64403209f2c8cb8527ac	24th March 2025	First Version
Second review (`v2`)	adea6c40869e529210005361d54add16507b108f	1st April 2025	Second 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
HIGH	H1	Missing `auth` modifier in `Gateway.endBatch` enables re-entrance for 3rd party-adapter contracts to resend the same batch multiple times	Fixed
MEDIUM	M1	`PoolRouter.updateHoldingValue` is not callable from the `MessageProcessor` due to access control restrictions	Fixed
LOW	L1	Calling `Gateway.topUp` twice in one transaction would result in the loss of `fuel` and `ETH`	Fixed
LOW	L2	Calling `send` with an empty message in `isBatching` mode will result in a separate empty bridge transaction instead of bundling it into a single one	Fixed
LOW	L3	Publicly callable `Gateway.startBatch` allows griefing attack in the same transaction	Fixed
INFORMATIONAL	I1	Incorrect assumptions in the comments about leftover token `DUST` in the `MultiShareClass.claimDepositUntilEpoch`	Fixed

HIGH Findings

H1 Missing `auth` modifier in `Gateway.endBatch` enables re-entrance for 3rd party-adapter contracts to resend the same batch multiple times

Gateway.sol#L351

The Gateway allows bundling individual messages into a batch to send a transaction only once to multiple bridges. If batching is activated by calling `startBatch`, instead of sending a transaction to the bridge adapters, the `send` function stores them in a `batch`.

Only after a final `endBatch` call is the batch transaction sent. The `endBatch` function has no parameters and deletes the `batch` storage afterward, which is intended to be transient storage in later versions.

Therefore, the `endBatch` function is not using an `auth` modifier and can be called by anyone.

This enables a reentrance attack by third-party adapter contracts. A malicious adapter, called as part of the `_send` process in `endBatch`, could re-enter and call `endBatch` again.

```
function endBatch() external {
    require(isBatching, NoBatched());

    for (uint256 i; i < chainIds.length; i++) {
        uint16 chainId = chainIds[i];
        _send(chainId, batch[chainId]);
        delete batch[chainId];
    }

    delete chainIds;
    pendingBatch = false;
    isBatching = false;
}
```

The `batch` is only deleted after the `_send` call, and the `isBatching` flag is set to `false` also after the call. The reentrance of the `endBatch` would allow `_send` to process any batch multiple times. This is a critical vulnerability since the transactions represent all the `actions` of the Centrifuge protocol. For example, a pool manager might allow redeeming x amount for a user, but sending the transaction multiple times to the vault would allow redeeming more funds, etc.

Recommendation: Add an `auth` modifier to `startBatch` and `endBatch`. The `endBatch` function should set `isBatching` and `pendingBatch` to false again before calling `_send`.

Centrifuge: Fixed in centrifuge/protocol-v3#194

MEDIUM Findings

M1. `PoolRouter.updateHoldingValue` is not callable from the `MessageProcessor` due to access control restrictions

PoolRouter.sol#L505

Description: The `PoolRouter` contains functions that can only be called by the `poolAdmin`. The `poolAdmin` must first call `execute` to unlock the pool, which is only possible if the caller of `execute` is the `poolAdmin`.

```
function execute(PoolId poolId, bytes[] calldata data) external payable {
    require(unlockedPoolId.isNull(), IPoolRouter.PoolAlreadyUnlocked());
```

```
        require(poolRegistry.isAdmin(poolId, msg.sender), IPoolRouter.NotAuthorizedAdmin());

        accounting.unlock(poolId);
        unlockedPoolId = poolId;

        multicall(data);

        accounting.lock();
        unlockedPoolId = PoolId.wrap(0);
    }
```

Other functions have the `_protectedAndUnlocked` check, meaning they must be called via `execute`.

```
    /// @dev Ensure the method is protected (see `_protected()`) and the pool is unlocked,
    /// which means the method must be called through `execute()`
    function _protectedAndUnlocked() internal protected {
        require(!unlockedPoolId.isNull(), IPoolRouter.PoolLocked());
    }

    function updateHolding(ShareClassId scId, AssetId assetId) public payable {
        _protectedAndUnlocked();
        // ...
    }
```

The `updateHolding` function uses the `_protectedAndUnlocked` check. However, there is another function, `updateHoldingValue`, which uses the `auth` pattern and calls `updateHolding`.

```
    function updateHoldingValue(PoolId poolId, ShareClassId scId, AssetId assetId, D18 pricePerUnit)
    external auth {
        // ...
        updateHolding(scId, assetId);
        // ...
    }
```

The `updateHoldingValue` function is triggered by a vault bridge transaction, and the `MessageProcessor` contract, which is a `ward` on the `PoolRouter`, should call `updateHoldingValue`. However, the internal `updateHolding` call requires the `poolAdmin` execute flow, causing this function to always revert.

Recommendation: Use an internal `_updateHolding` function to reuse the logic of `updateHolding`. The `updateHoldingValue` function should call `_updateHolding` to bypass the access control check.

Centrifuge: Fixed in centrifuge/protocol-v3#244.

xmxanuel: Fixed.

LOW Findings

L1. Calling `Gateway.topUp` twice in one transaction would result in the loss of `fuel` and `ETH`

Gateway.sol#L336

Description: The `Gateway` needs to pay for the transactions to the bridges. Therefore, contracts using the gateway can call `topUp` to deposit `ETH` for an upcoming `send` call.

The transferred ETH is reflected in the transient `fuel` variable.

However, if the `topUp` is called twice, the previous `fuel` value will be overwritten.

This means the `send` call would have less `fuel` available, and the additional deposited ETH is lost.

```
if (!isBatching || pendingBatch) {
    require(msg.value != 0, "Gateway/cannot-topup-with-nothing");
    fuel = msg.value;
}
```

In the current implementation, all contracts using the `Gateway` call `topUp` and `send` only once, so it is not an issue.

Recommendation: Allow the `topUp` function to be called multiple times before calling `send` and add the `msg.value` to the existing `fuel`.

```
if (!isBatching || pendingBatch) {
    require(msg.value != 0, "Gateway/cannot-topup-with-nothing");
-    fuel = msg.value;
+    fuel += msg.value;
}
```

Centrifuge: Fixed in centrifuge/protocol-v3#194.

xmxanuel: Fixed.

L2. Calling `send` with an empty message in `isBatching` mode will result in a separate empty bridge transaction instead of bundling it into a single one

Gateway.sol#L266

Description: When the `Gateway` is in batching mode, calling the `send` function with an empty message will still add the `chainId` to the `chainIds` array. If the same `chainId` is called again with an actual message, the `chainId` will be pushed a second time to the `chainIds`.

This leads to a situation where the same `chainId` appears twice in the array.

```
if (isBatching) {
    pendingBatch = true;

    bytes storage previousMessage = batch[chainId];
    if (previousMessage.length == 0) {
        chainIds.push(chainId);
        batch[chainId] = message;
    } else {
        batch[chainId] = bytes.concat(previousMessage, message);
    }
}
```

Recommendation: Ensure that the `send` function is only called with a non-empty message when in batching mode.

During the execution of `endBatch`, after processing the first occurrence of a `chainId` with an actual message via `_send`, the `batch[chainId]` is deleted. Consequently, when the second occurrence of the same `chainId` is processed, `_send` is called with an empty message, resulting in an unnecessary empty bridge transaction.

Centrifuge: Fixed in centrifuge/protocol-v3#194.

xmxanuel: Fixed.

L3. Publicly callable `Gateway.startBatch` allows griefing attack in the same transaction

VaultRouter.sol#L51

Description: The `batching` of transactions can be exploited by a griefing attack within the same overall Ethereum transaction. The `Gateway.startBatch` function, which initiates the batching process, sets the transient storage flag `isBatching` to true.

The `multicall` function in both `VaultRouter` and `PoolRouter` only tops up the gateway for paying transactions if `isBatching()` has not been started. This logic is intended to allow `multicalls` within `multicalls` but can be exploited through a griefing attack.

```
function multicall(bytes[] calldata data) public payable override(Multicall, IMulticall) {
    bool wasBatching = gateway.isBatching();
    if (!wasBatching) {
        gateway.startBatch();
    }

    super.multicall(data);

    if (!wasBatching) {
        gateway.topUp{value: msg.value}();
        gateway.endBatch();
    }
}
```

The transient storage persists throughout the entire transaction. If untrusted third-party code is executed within the same transaction, such as by sending ETH to a contract address, an attacker could call `startBatch`. This would cause the actual multicall transaction to revert because the gateway did not receive an ETH top-up.

Recommendation: Implement access control to restrict who can call `startBatch`. Ensure that the `multicall` function handles the `isBatching` state securely to prevent unauthorized manipulation.

Centrifuge: Fixed in centrifuge/protocol-v3#194.

xmxanuel: Fixed.

Informational

I1. Incorrect assumptions in the comments about leftover token `DUST` in the `MultiShareClass.claimDepositUntilEpoch`

https://github.com/centrifuge/protocol-v3/blob/2b657c3de80b2f1153ce64403209f2c8cb8527ac/src/pools/MultiShareClass.sol#L422

Description: The comments in the claimDepositUntilEpoch function suggest that leftover token DUST is limited to a single atom. However, this assumption is incorrect. The discrepancy between the sum of claimable user amounts and the total issued share class tokens can be more than a single atom, as demonstrated in the following example.

Example Testcase

```
function testSumAllUsersLessThanIssuedShareAmount() public {
    uint navPerShare = 1.5 * 1e18;
    // Assume 3 users with deposits of 33, 33, and 34, totaling 100
    // All deposits are approved, and currency equals asset in value
    uint depositPool = 100 * 1e18;
    uint issuedShareAmount = depositPool * 1e18 / navPerShare;
    console.log("issuedShares", issuedShareAmount);

    // approveDeposits will be 100 * 1e18 and currency equals asset
    uint depositApproved = depositPool;

    // claimDepositUntilEpoch calculations for the 3 users
    uint claimableShareAmount_user1 = 33 * 1e18 * issuedShareAmount / depositApproved;
    uint claimableShareAmount_user2 = 34 * 1e18 * issuedShareAmount / depositApproved;
    uint claimableShareAmount_user3 = 33 * 1e18 * issuedShareAmount / depositApproved;

    uint sumAllUsers = claimableShareAmount_user1 + claimableShareAmount_user2 +
claimableShareAmount_user3;

    assertEq(sumAllUsers, issuedShareAmount, "issuedShareAmount not equal sumAllUsers");
}
```

Output

```
[FAIL: issuedShareAmount not equal sumAllUsers: 6666666666666666664 != 6666666666666666666]
testSumAllUsersLessThanIssuedShareAmount() (gas: 7316)
```

In this example, the sum of claimable shares for all users is less than the issued share amount by 2 atoms, not just 1. This indicates that the leftover DUST can be more than a single atom, contrary to the comment's implication.

Recommendation: Update the comments to reflect that the leftover `DUST` can be more than a single atom, especially in scenarios involving multiple investors. This will provide a more accurate understanding of potential discrepancies in share distribution.

Centrifuge: Fixed in 3e82ccb.

xmxanuel: Fixed.