Centrifuge Protocol v3

Security Review

Cantina Managed review by:

0xLeastwood, Lead Security Researcher
Drastic Watermelon, Associate Security Researcher

July 2, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Centrifuge empowers asset managers to tokenize, manage, and distribute their funds onchain, while providing investors access to a diversified group of tokenized assets.

From Apr 28th to May 19th the Cantina team conducted a review of centrifuge-protocol-v3 on commit hash 814ea57b. The team identified a total of **12** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	3	3	0
Low Risk	5	3	2
Gas Optimizations	0	0	0
Informational	4	3	1
Total	**12**	**9**	**3**

3 Findings

3.1 Medium Risk

3.1.1 `AxelarAdapter.execute` will fail to process any incoming message

Severity: Medium Risk

Context: AxelarAdapter.sol#L60-L75, CastLib.sol#L15-L18

Summary: `AxelarAdapter.execute` will fail to process any incoming message because `CastLib.toAddress(string)` will always revert for any input provided by Axelar's infrastructure.

Finding Description: `CastLib.toAddress(string)` incorrectly expects the input string's length to be equal to 20:

```
function toAddress(string calldata addr) internal pure returns (address) {
    require(bytes(addr).length == 20, "Input should be 20 bytes");
    return address(bytes20(bytes(addr)));
}
```

When relaying a cross chain transaction to the destination chain, Axelar provides the cross chain transaction's sender's address as a string, including a leading "0x": this may be verified by inspecting the `sourceAddress` calldata parameter of the destination chain's side of an Axelar cross chain transaction. See for example transaction 0x780b5469e5fd9289a40b21478bd380098a744751a527d715e848a42fe6b2237a.

Because all of the above, the `require` statement's condition will always evaluate to `false`:

```
Welcome to Chisel! Type `!help` to show available commands.
 string memory str = "0xe6B3949F9bBF168f4E3EFc82bc8FD849868CC6d8"
Traces:
  [211] 0xBd770416a3345F91E4B34576cb804a576fa48EB1::run()
     ← [Stop]

 bytes(str).length == 20
Type: bool
 Value: false

 bytes(str).length == 42
Type: bool
 Value: true
```

Impact Explanation: `AxelarAdapter.execute` failing for any incoming message, implies the inability for it to reach quorum within the Gateway contract. This in turn implies that no cross chain message will ever be processed by the system.

Likelihood Explanation: The issue will manifest for any cross chain message relayed with Axelar.

Recommendation: `CastLib.toAddress(string)` should verify the condition `bytes(addr).length == 42`.

Centrifuge: Fixed in commit 7eae90a0.

Cantina Managed: Fix verified.

3.1.2 `AxelarAdapter.send` sets incorrect `destinationAddress` for every cross chain transaction

Severity: Medium Risk

Context: AxelarAdapter.sol#L82-L99, CastLib.sol#L15-L18

Description: Axelar cross chain transactions may be initiated by using `IGateway.callContract`. In this call, senders must provide the contract intended to receive a cross chain contract call in the `contractAddress` parameter.

`AxelarAdapter.send` uses `CastLib.toString` to encode an address as a string, providing this method's output as the recipient of Axelar's cross chain contract call:

```
function toString(address addr) internal pure returns (string memory) {
    return string(abi.encodePacked(addr));
}
```

Because the `CastLib.toString` method formats an address as a string by returning `string(abi.encodePacked(addr))` it returns the UTF-8 encoded string represented by `addr`'s raw hex bytes:

```
Welcome to Chisel! Type `!help` to show available commands.
 address addy = 0xe6B3949F9bBF168f4E3EFc82bc8FD849868CC6d8;
Traces:
  [133] 0xBd770416a3345F91E4B34576cb804a576fa48EB1::run()
    ← [Stop]

 string(abi.encodePacked(addy))
Type: string
 UTF-8: N>I
 Hex (Memory):
 Length ([0x00:0x20]): 0x000000000000000000000000000000000000000000000000000000000000002e
 Contents ([0x20:..]): 0xe6b394efbfbdefbfbdefbfbd16efbfbd4e3eefbfbdefbfbdefbfbdefbfbdefbfbd49efbfbdefbfbdefbfb ⌋
↪  defbfbd000000000000000000000000000000000000
 Hex (Tuple Encoded):
 Pointer ([0x00:0x20]): 0x0000000000000000000000000000000000000000000000000000000000000020
 Length ([0x20:0x40]): 0x000000000000000000000000000000000000000000000000000000000000002e
 Contents ([0x40:..]): 0xe6b394efbfbdefbfbdefbfbd16efbfbd4e3eefbfbdefbfbdefbfbdefbfbdefbfbd49efbfbdefbfbdefbfb ⌋
↪  defbfbd000000000000000000000000000000000000
```

Impact Explanation: Axelar cross chain contract calls will be sent to incorrect recipients.

Likelihood Explanation: The highlighted issue will manifest for any cross chain contract call sent via AxelarAdapter.

Recommendation: To correcty encode an address as a string according to how Axelar expects, `CastLib.toString` should use or emulate OpenZeppelin's Strings library.

Centrifuge: Fixed in commit 7eae90a0.

Cantina Managed: Fix verified.

3.1.3 Edge case in request handling can cause deposits/redemptions to be broken

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Due to the asynchronous nature of Centrifuge V3, pool request cancellations following the initial request to deposit/redeem may be handled in-order on the destination chain but out-of-order on the source chain. This can cause permanent locking of future deposit/redemption requests, leading to a liveness issue for users.

The exact scenario for when deposits are potentially locked for an unknown amount of time is outlined below:

1. A deposit is made from a non-pool chain.
2. Before the deposit request has been relayed to the pool chain and updated `userOrder.pending`, the same user request to cancel the same deposit request.
3. The cancel deposit request is handled first by calling `Hub.cancelDepositRequest()` and because `cancelledAssetAmount = 0`, no message is relayed back to the non-pool chain.
4. Subsequently, the deposit request is handled on the pool chain and correctly fulfilled as expected.
5. But due to the rounding mentioned above, `userOrder.pending` is not fully zeroed out and may never be fully handled as long as there are other requests from users in a given epoch.
6. Consequently, future deposits will not function as expected because `AsyncRequestManager.requestDeposit()` expects `state.pendingCancelDepositRequest != true` which will never be possible because of point 5.

It is important to note that the more severe case for this issue is in redemptions, hence the outlined scenario mentioned above would also apply here.

Recommendation: Some considerations need to be made here to adjust any rounded balances in the `ShareClassManager` contract when claiming deposits/redemptions to ensure leftover amounts can always be cancelled to unlock new requests on the source chain.

Centrifuge: Fixed in commit ee717dd2.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 `BytesLib.sliceZeroPadded` adds non-zero bytes in padding

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: `BytesLib.sliceZeroPadded` can add non-zero bytes of padding.

The method used has been forked from solidity-bytes-utils/BytesLib.sol, relaxing the condition at BytesLib.sol#L238. While the intention for this change is to allow users of the library's method to request a `bytes` slice which ends after the initial data's end, the original implementation was not designed for this use case.

Given that the highlighted method is not used in any crucial section of the codebase, no real impact is implied.

Proof of Concept:

```
// SPDX-License-Identifier: BUSL-1.1
pragma solidity 0.8.28;

import "forge-std/Test.sol";
import {BytesLib} from "src/misc/libraries/BytesLib.sol";

contract BytesLibTest is Test {
    function testSliceZeroPadded(bytes memory data, bytes memory randomStart, uint8 randomLength) public pure {
        bytes memory value = bytes.concat(randomStart, abi.encodePacked(data));
        bytes memory expected;

        if (randomLength > data.length) {
            // Padding case
            bytes memory padding = new bytes(randomLength - data.length);
            expected = abi.encodePacked(data, padding);
        } else {
            // Base case, equivalent to original code's -- not really relevant for this test
            expected = BytesLib.slice(value, randomStart.length, randomLength);
        }

        bytes memory got = BytesLib.sliceZeroPadded(value, randomStart.length, randomLength);

        assertEq(got.length, expected.length, "!length");
        assertEq(
            got,
            expected,
            "!bytes"
        );
    }

    function testSliceZeroPadded_counterexample() public pure {
        bytes memory data = hex"deadbeef";
        bytes memory start = hex"0000000000000000000000004e59b44847b379578588920ca78fbf26c0b4956c";
        uint8 length = 38;

        testSliceZeroPadded(data, start, length); // fails with "!bytes"
    }
}
```

Recommendation: The method ought to be rewritten in Solidity to improve its clarity and readability. For example, the method could leverage the original implementation to obtain the initial data's slice and then add padding:

```
function sliceZeroPadded_solidity(bytes memory _bytes, uint256 _start, uint256 _length) external pure returns
↪  (bytes memory) {
    bool needsPad = _bytes.length < _start + _length;
    if (!needsPad) return slice(_bytes, _start, _length);

    bytes memory temp = slice(_bytes, _start, _bytes.length - _start);
    return abi.encodePacked(temp, new bytes(_length + _start - _bytes.length));
}
```

Centrifuge: Fixed in commit 3f632296.

Cantina Managed: Fix verified.

3.2.2 `ERC20.burn` requires wards to have been granted an allowance to burn tokens

Severity: Low Risk

Context: ERC20.sol#L151-L173

Description: `ERC20.burn` is decorated with the `auth` modifier, implying the method is only callable by authorized wards. At the same time, if method's caller is different from the account whose tokens are being burnt, the method will attempt to consume the allowance granted by the token holder to the caller. As a consequence, wards can be denied the ability of burning user tokens in case no allowance is ever set.

Recommendation: Either remove the `auth` modifier or avoid consuming the allowance.

Centrifuge: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 `VaultRouter.enableLockDepositRequest` can revert when wrapping underlying tokens

Severity: Low Risk

Context: VaultRouter.sol#L130-L144, VaultRouter.sol#L156-L157, VaultRouter.sol#L272-L282

Description: `VaultRouter.enableLockDepositRequest`, when `vaultDetails.isWrapper && assetBalance < amount` holds, attempts to wrap tokens on the user's behalf before incrementing a user's locked request. When wrapping tokens via `VaultRouter.wrap`, the contract will wrap the minimum between the request `amount` and the user's balance of `underlying` tokens:

```
function wrap(address wrapper, uint256 amount, address receiver, address owner) public payable protected {
    require(owner == msg.sender || owner == address(this), InvalidOwner());
    address underlying = IERC20Wrapper(wrapper).underlying();

    amount = MathLib.min(amount, IERC20(underlying).balanceOf(owner));
    require(amount != 0, ZeroBalance());
    SafeTransferLib.safeTransferFrom(underlying, owner, address(this), amount);

    _approveMax(underlying, wrapper);
    require(IERC20Wrapper(wrapper).depositFor(receiver, amount), WrapFailed());
}
```

In case that `IERC20(underlying).balanceOf(owner) < amount` holds and less tokens than `amount` are wrapped, `VaultRouter.enableLockDepositRequest` will still use `amount` when invoking `VaultRouter.lockDepositRequest`, eventually leading it to attempt to pull more funds than held by `owner`, making the transaction revert.

Recommendation: `VaultRouter.wrap` should return the amount of wrapped tokens obtained, which should be passed to `VaultRouter.lockDepositRequest`.

Centrifuge: Fixed in commit 162b66c8.

Cantina Managed: Fix verified.

3.2.4 Holding amounts may be updated erroneously when queueing asset amounts

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `BalanceSheet` contract facilitates deposits and withdrawals by pushing/pulling assets to and from the escrow. Share token burns and mints are being handled here too, each of which have direct influence on the `totalIssuance` being tracked and the asset holding balances.

When the `AsyncRequestManager` approves deposits, it notes the asset amount in the `BalanceSheet`. If the queue is enabled, the `sender.sendUpdateHoldingAmount()` call is delayed until the manager calls `BalanceSheet.submitQueuedAssets()`.

It becomes problematic when the pool manager calls `PoolManager.updatePricePoolPerShare()` while there are existing assets queued. Consequently, the later call to `BalanceSheet.submitQueuedAssets()` will update holding amounts on the latest/current `pricePoolPerAsset` which is not entirely accurate.

Ultimately, the `Holdings` contract is exposed to fluctuations in the asset ↔ pool currency exchange rate that happens while asset amounts are being queued.

Recommendation: It does seem that the Hub manager can make adjustments to the holding amounts by calling `Hub.updateHoldingValue()` so it might not be worth making a direct fix to this.

Centrifuge: Acknowledged.

Cantina Managed: Acknowledged.

3.2.5 Non-batched `Hub` transactions may fail to relay payload data to adapters

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: There are *two* instances where a non-batched transaction will partially fail at the `Gateway` because the transaction is refunded after the first `Gateway.send()` call:

- `Hub.notifyDeposit()`.
- `Hub.notifyRedeem()`.

These aforementioned functions have an internal `_pay()` function which attempts to subsidize payload sending costs to the adapters but will subsequently refund the leftover amount after the first send to the `Gateway` contract, causing the second payload send to be underpaid. This leads to lacklustre user experience as users are expected to repay the underpaid batch.

Recommendation: Modify affected functions in the `Hub` contract to handle this correctly.

Centrifuge: Fixed in commit 36d96a90.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 `BytesLib.sliceZeroPadded` reverts with panic instead of custom error

Severity: Informational

Context: BytesLib.sol#L17

Description: `BytesLib.sliceZeroPadded` executes an overflow check by ensuring that `_length + 31 >= _length` holds. Because the check is executed in checked arithmetic, when `_length + 31` does overflow, execution will revert with a panic error instead of the custom `SliceOverflow` error.

Recommendation: Wrap the `require` check in an `unchecked` block.

Centrifuge: Fixed in commit 3f632296.

Cantina Managed: Fix verified.

3.3.2 System assumes ERC6909 tokens cannot have `tokenId = 0`

Severity: Informational

Context: PricingLib.sol#L246-L247, TokenRecoverer.sol#L19-L34, Recoverable.sol#L21-L27, BalanceSheet.sol#L246-L253, Escrow.sol#L19-L33, AsyncVaultFactory.sol#L36, SyncDepositVaultFactory.sol#L44, PoolManager.sol#L153-L161, PoolManager.sol#L559-L563

Description: Different contracts implement a unified interface to interact with ERC-20 and ERC-6909 tokens: when `tokenId == 0` is provided, the system assumes it must interact with an ERC-20 token. Because the ERC-6909 spec doesn't specify any restriction for the `tokenId` field, the system may face compatibility issues with future ERC-6909 implementations being used.

Note that Uniswap's ERC-6909 implementation is unaffected, as `tokenIds` are assigned as an ERC-20's address cast to `uint160`.

Recommendation: Refactor the system to also support using ERC-6909 tokens with `tokenId = 0`.

Centrifuge: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 `Gateway` contract deployment fails when `deployer != msg.sender`

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `Gateway` contract's constructor makes an internal call to `setRefundAddress(GLOBAL_POT, IRecoverable(address(this)))` which has an `auth` modifier. The initial authentication on deployment is only ever granted to the `deployer` parameter in the constructor and hence this internal call will fail if `deployer != msg.sender`.

Recommendation: Consider documenting whether or not this is intended and making adjustments where necessary to accommodate more permissive behaviour.

Centrifuge: Fixed in commit 9c45f4ec.

Cantina Managed: Fix verified.

3.3.4 `Gateway` repay and retry actions do not adhere to "Checks-Effects-Interactions" guidelines

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Typical implemented behaviour for a function which makes any external call is to update all storage if possible before the call is made to avoid complex forms of reentrancy. It's unclear if there is any attack vector here because the `underpaid` and `failedMessages` counters are eventually updated and to circumvent this, an attacker would need to takeover the `Gateway` contract.

Recommendation: Consider updating the `repay()` and `retry()` functions to be CEI compatible.

Centrifuge: Fixed in commit bf6b8d3d.

Cantina Managed: Fix verified.