Centrifuge v3
Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher
Devtooligan, Security Researcher

August 12, 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 Jul 30th to Aug 4th the Cantina team conducted a review of protocol-v3 on commit hash 7ca58197. The team identified a total of **28** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	8	4	4
Gas Optimizations	3	2	1
Informational	17	15	2
Total	**28**	**21**	**7**

3 Findings

3.1 Low Risk

3.1.1 `BalanceSheet::multicall()` function can lose ETH in edge case scenario

Severity: Low Risk

Context: Hub.sol#L91-L104, BalanceSheet.sol#L80-L91, VaultRouter.sol#L67-L80

Description: The `multicall()` function implements batching functionality but lacks proper transaction payment handling that should accompany the batching operations. The function is marked as `payable`, indicating it can receive ETH, but it only handles batching start/end operations without corresponding payment operations.

The vulnerability manifests in the edge case where:

1. A user calls `multicall()` with `msg.value > 0` and an empty `data` array.
2. The function starts batching, calls `super.multicall(data)` with no operations, and ends batching.
3. No payment handling occurs, leaving the sent ETH permanently locked in the contract.

While most individual functions in the contract are not payable and would revert if called with ETH, the empty data array scenario bypasses this protection since no function calls are attempted.

Note: `Hub::multicall()` and `VaultRouter::multicall()` do have functionallity to process `msg.value`.

Recommendation: Condider checking `msg.value == 0` or checking `data.length >0` in the `BalanceSheet::multicall()` function.

Centrifuge: Fixed in commit 4816473f.

Cantina Managed: Fix verified.

3.1.2 Missing vault validation in link and unlink operations

Severity: Low Risk

Context: Spoke.sol#L381

Description: The `linkVault()` and `unlinkVault()` functions in the Spoke contract access vault details directly without validating that the vault was properly deployed through the system. This inconsistency creates a potential avenue for linking unregistered vaults. In the `updateVault()` function, there is an explicit check to ensure only validated vaults are processed:

```
// Needed as safeguard against non-validated vaults
// I.e. we only accept vaults that have been deployed by the pool manager
require(_vaultDetails[vault].asset != address(0), UnknownVault());
```

However, the `linkVault()` and `unlinkVault()` functions directly access `_vaultDetails[vault]` without this validation:

```
function linkVault(PoolId poolId, ShareClassId scId, AssetId assetId, IVault vault) public auth {
    // ... other checks ...
    VaultDetails storage vaultDetails_ = _vaultDetails[vault]; // Missing validation
    require(!vaultDetails_.isLinked, AlreadyLinkedVault());
    // ...
}
```

This could allow administrators to link vaults that haven't been properly deployed through the `deployVault()` function, potentially leading to inconsistent system state.

Recommendation: Add vault validation checks to both `linkVault()` and `unlinkVault()` functions to ensure consistency with the existing validation pattern. Alternatively, consider using the existing `vaultDetails()` function which includes the validation check, though this would require adjusting the code to work with the returned memory struct instead of storage references.

Centrifuge: Fixed in commit 421b15c1.

Cantina Managed: Fix verified.

3.1.3 Reentrancy protection mechanism in `_protected()` is ineffective

Severity: Low Risk

Context: Hub.sol#L65-L69, Hub.sol#L122-L127, Hub.sol#L146-L151, Hub.sol#L731-L732, Hub.sol#L740-L750

Description: The Hub contract's reentrancy protection mechanism using the `_protected()` internal function does not provide actual protection against reentrancy attacks. This could leave functions vulnerable to reentrancy, potentially allowing attackers to manipulate contract state or drain funds. The Hub contract implements a `_protected()` internal function intended to prevent reentrancy:

```
/// @dev Protect against reentrancy
function _protected() internal protected {}
```

This function is used in critical operations like `notifyDeposit()` and `notifyRedeem()` with the expectation that it will prevent reentrancy attacks. However, this implementation is ineffective because:

- The `protected` modifier's logic is not properly applied when used in this manner, because it is only on one location in the function. For nonreentrancy protection, these should be something at the beginning and the end of the function.

Functions relying on this protection include:

- `notifyDeposit()` - handles deposit notifications and callbacks.
- `notifyRedeem()` - handles redemption notifications and callbacks.

We don't see a real reentrancy possibility in these function, thus the risk is currently limited.

Note: a `nonReentrancy` mechanism is also important in relation to modifier `payTransaction`, because otherwise the `transactionRefund` can be overwritten.

The function `_isManager()` uses a similar pattern as `_protected()` and thus also doesn't protect against reentrancy. This is less important because its authorisation mechanism is effective.

Proof of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;
import "hardhat/console.sol";
contract testReentrancy {
    error UnauthorizedSender();
    address private transient _initiator;
    modifier protected() {
        if (_initiator == address(0)) {
            _initiator = msg.sender;
            _;
            _initiator = address(0);
        } else {
            require(msg.sender == _initiator, UnauthorizedSender());
            _;
        }
    }
    function _protected() internal protected {}
    function notifyDeposit(uint n) public {
        console.log(n);
        _protected();
        if (n > 0) this.notifyDeposit(n-1);
    }
    function test() public {
        notifyDeposit(5);
    }
}
```

Recommendation: Use the `protected` modifier instead of the `_protected()` function and remove the `_protected()` function. For `_isManager()`: doublecheck if the intention of the `protected` modifier is to provide reentrancy protection. If so, move it to the calling functions. Otherwise consider removing the `protected` modifier from `_isManager()` because it doesn't add any value.

Centrifuge: Acknowledged. Will be solved in a future release.

Cantina Managed: Acknowledged.

3.1.4 Deployment address verification mismatch for contract addresses

Severity: Low Risk

Context: FullDeployer.s.sol#L95-L98

Description: The `_verifyMainnetAddresses()` function in `FullDeployer.s.sol` contains hardcoded address verification that does not match the actual deployed contract addresses documented in the official Centrifuge protocol deployments. Specifically, three contracts have mismatched addresses:

- `asyncRequestManager`: Script expects `0x58d57896EBbF000c293327ADf33689D0a7Fd3d9A` but documentation shows `0xF06f89a1b6C601235729A689595571B7455dD433`.
- `asyncVaultFactory`: Script expects `0xE01Ce2e604CCe985A06FA4F4bCD17f1F08417BF3` but documentation shows `0xED9D489BB79c7cB58C522f36fC6944eaA95ce385`.
- `syncDepositVaultFactory`: Script expects `0x3568184784E8ACCaacF51A7F710a3DE0144E4f29` but documentation shows `0x21bf2544b5a0B03C8566a16592Ba1B3b192b50Bc`.

This discrepancy stems from version differences between `v3.0.0` and `v3.0.1` deployments, where the salt generation method changed from using hashed versions (`keccak256(abi.encodePacked("3"))`) to unhashed versions (`bytes32(bytes("3"))`).

Recommendation: Update the hardcoded addresses in `_verifyMainnetAddresses()` to match the correct v3.0.1 deployment addresses:

```
  function _verifyMainnetAddresses() internal view {
      // ... other addresses remain the same ...
-     require(address(asyncRequestManager) == 0x58d57896EBbF000c293327ADf33689D0a7Fd3d9A);
+     require(address(asyncRequestManager) == 0xF06f89a1b6C601235729A689595571B7455dD433);
      require(address(syncManager) == 0x0D82d9fa76CFCd6F4cc59F053b2458665C6CE773);
-     require(address(asyncVaultFactory) == 0xE01Ce2e604CCe985A06FA4F4bCD17f1F08417BF3);
+     require(address(asyncVaultFactory) == 0xED9D489BB79c7cB58C522f36fC6944eaA95ce385);
-     require(address(syncDepositVaultFactory) == 0x3568184784E8ACCaacF51A7F710a3DE0144E4f29);
+     require(address(syncDepositVaultFactory) == 0x21bf2544b5a0B03C8566a16592Ba1B3b192b50Bc);
      // ... rest of addresses remain the same ...
  }
```

Also update the deployment documentation.

Centrifuge: Fixed in commit 4b619114.

Cantina Managed: Fix verified.

3.1.5 `refund` address not always checked to be non zero

Severity: Low Risk

Context: Gateway.sol#L152, Gateway.sol#L180, Gateway.sol#L216-L218, Gateway.sol#L243-L244

Description: In several locations a check is done that `address(subsidy[poolId].refund!=address(0)`. However in function `_send()` this isn't explicitly checked.

Recommendation: Consider using a general address if the refund address is zero. For example the `GLOBAL_POT`; although the requires additional administration.

Centrifuge: Fixed in commit cf42ddca.

Cantina Managed: Fix verified.

3.1.6 `refund` address and `rely()`

Severity: Low Risk

Context: CommonDeployer.s.sol#L72, CommonDeployer.s.sol#L76, Gateway.sol#L216-L229, Gateway.sol#L237-L240

Description: Function `_requestPoolFunding()` can retrieve funds from any contract that `rely()`s on the `Gateway` contract.

This can be done if that address is added via `setRefundAddress()`.

The following contracts `rely()`s on the `Gateway` contract:

- `MultiAdapter`.
- `MessageProcessor`.

The risk is limited because `setRefundAddress()` is authorized, and `multiAdapter` and `messageProcessor` don't inherit `Recoverable`. A special case is the situation where `refund == address(this)`, then `subsidy[GLOBAL_POT].value` would be set to 0. This currently can't happen because `Gateway` doesn't `rely()` on itself.

Recommendation: Consider checking none of the system addresses are used in `setRefundAddress()`, especially `address(this)`.

Centrifuge: Acknowledged. This is prevented by the authorization mechanism.

Cantina Managed: Acknowledged.

3.1.7 `poolId` shouldn't be 0

Severity: Low Risk

Context: PoolEscrowFactory.sol#L31-L32, Gateway.sol#L32, Guardian.sol#L64-L66, PoolId.sol#L18-L20, Hub.sol#L107-L115

Description: `poolId` shouldn't be 0, because that is a special value that is used for `GLOBAL_POT`.

There is no explicit checked in:

- `guardian::createPool()`.
- `hub::createPool()`.
- `hubRegistry.registerPool()`.

Note: there is an implicit check in `hub::createPool()` where it is checked with `localCentrifugeId()`.

Note: the risk is limited because the function setting the `poolId` are authorized.

Recommendation: Consider explicitly checking `poolId != 0`.

Centrifuge: Acknowledged. `poolId` contains a non-zero `centrifugeId`, see finding "`CentrifugeId` shouldn't be 0".

Cantina Managed: Acknowledged.

3.1.8 `CentrifugeId` shouldn't be 0

Severity: Low Risk

Context: FullDeployer.s.sol#L124, MultiAdapter.sol#L34-L37, MessageDispatcher.sol#L44-L51, MessageProcessor.sol#L64, MessageProcessor.sol#L68

Description: `CentrifugeId` should not be 0 because this is a special value. It is also not allowed in `MessageProcessor::handle()`. There is no explicit check that the `CentrifugeId` isn't `0`, neither in the deployment script nor in the constructors of `MultiAdapter` or `MessageDispatcher`.

Recommendation: Consider explicitly checking `CentrifugeId != 0` to prevent configuration mistakes.

Centrifuge: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Gas Optimization

3.2.1 Gas savings by conditionally updating `shareQueue`

Severity: Gas Optimization

Context: BalanceSheet.sol#L171

Description: Small gas savings available by conditionally updating `shareQueue`.

Recommendation: Consider implementing something like:

```
- shareQueue.isPositive = shareQueue.delta != 0;
+ if (shareQueue.delta == 0 && shareQueue.isPositive) shareQueue.isPositive = false;
```

Centrifuge: Acknowledged. We prefer to keep the current version that reduces ifs.

Cantina Managed: Acknowledged.

3.2.2 `underpaid_[]` entries stay forever

Severity: Gas Optimization

Context: Gateway.sol#L202-L214

Description: Once `repay()` succeeds, then the `underpaid_[]` isn't necessary any more. Keeping them increases the contract state.

Recommendation: Consider deleting `underpaid[centrifugeId][batchHash]` when `underpaid_.counter` reaches zero and the message is sucessfully send.

Centrifuge: Fixed in commit c7371ee1.

Cantina Managed: Fix verified.

3.2.3 Redundant check in `newManager()`

Severity: Gas Optimization

Context: OnOfframpManager.sol#L131-L132, Spoke.sol#L430-L434

Description: Function `newManager()` checks the token isn't `address(0)`. However `Spoke::shareToken()` already checks the `token !=0`, so this check is redundant.

Recommendation: Consider removing the redundant check from `newManager()`.

Centrifuge: Fixed in commit a5f23ee2.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Naming inconsistencies in `OnOfframpManager`

Severity: Informational

Context: OnOfframpManager.sol#L38-L41, OnOfframpManager.sol#L53

Description: The `OnOfframpManager` constructor contains inconsistent variable naming that could lead to confusion for developers and auditors. The constructor parameter is named `spoke_` but is assigned to the `contractUpdater` state variable. This naming mismatch is inconsistent with the `OnOfframpManagerFactory`, which correctly uses `contractUpdater_` as the parameter name.

Additionally, the error message `NotSpoke()` in the `update` function is outdated and should use a more generic authorization error to match the naming convention used in similar contracts like `MerkleProof-Manager`.

Recommendation: Update the constructor parameter name and error message for consistency.

Centrifuge: Fixed in commit 91355d5b.

Cantina Managed: Fix verified.

3.3.2 Inconsistent License Identifier

Severity: Informational

Context: IPoolEscrowFactory.sol#L1, ITokenFactory.sol#L1, IVaultFactory.sol#L1

Description: Most interface files are changed to `GPL-2.0-or-later` in PR 477. However the following are not:

- `ITokenFactory.sol`.
- `IVaultFactory.sol`.
- `IPoolEscrowFactory.sol`.

Recommendation: Review and update the license info.

Centrifuge: Fixed in commit 402bd3bd.

Cantina Managed: Fix verified.

3.3.3 Simplify conditional structure

Severity: Informational

Context: GasService.sol#L85-L87

Description: The `messageGasLimit()` function in the GasService contract uses an unnecessarily complex chain of `else if` statements. Since each condition branch contains a `return` statement, the `else` keywords are redundant and can be removed to improve code readability and maintainability. The current implementation uses a long chain of `else if` statements:

```
if (kind == MessageType.ScheduleUpgrade) {
    return scheduleUpgrade;
} else if (kind == MessageType.CancelUpgrade) {
    return cancelUpgrade;
} else if (kind == MessageType.RecoverTokens) {
    return recoverTokens;
}
// ... continues for all message types
```

Recommendation: Consider simplifying the conditional structure by removing the `else` keywords since each branch returns early:

```
if (kind == MessageType.ScheduleUpgrade) return scheduleUpgrade;
if (kind == MessageType.CancelUpgrade)   return cancelUpgrade;
if (kind == MessageType.RecoverTokens)   return recoverTokens;
if (kind == MessageType.RegisterAsset)   return registerAsset;
// ... apply same pattern to all remaining conditions
```

Centrifuge: Fixed in commit f06a2336.

Cantina Managed: Fix verified.

3.3.4 Missing `TokenRecoverer` contract registration

Severity: Informational

Context: CommonDeployer.s.sol#L161-L166

Description: The `TokenRecoverer` contract is deployed in the `_preDeployCommon()` function but is not included in the contract registration block. While the contract is properly deployed and integrated into the system architecture, it is missing from the registration process that records deployed contract addresses as well as the deployment documentation.

Recommendation: Add the `TokenRecoverer` contract to the registration block to ensure consistent documentation and tracking:

```
+ register("tokenRecoverer", address(tokenRecoverer));
```

Also add it to the deployment documentation.

Centrifuge: Fixed in PR 559.

Cantina Managed: Fix verified.

3.3.5 Inconsistent setting of `isIncrease` when `delta` is zero

Severity: Informational

Context: BalanceSheet.sol#L218, BalanceSheet.sol#L241, BalanceSheet.sol#L246

Description: When net `deposits == 0` then `isIncrease` is set to True. This is inconsistent with `submitQueuedShares` which uses `shareQueue.isPositive` which currently is `false` when the delta is 0.

Recommendation: Stay consistent by setting `isIncrease` to true when `deposits > 0`.

Centrifuge: Fixed in commit cb9083ed.

Cantina Managed: Fix verified.

3.3.6 `_requestPoolFunding()` is suboptimal for shared `refund` addresses

Severity: Informational

Context: Gateway.sol#L152-L166, Gateway.sol#L216-L229

Description: If a `refund` address would be shared among different pools, then the approach of `_requestPoolFunding()` is suboptimal, because it gets all funds from `refund` and uses it to subsidize `poolId`. This would prevent other pools from using it, even though perhaps less funds are required.

Recommendation: Consider documenting at `setRefundAddress()` that `refund` addresses should not be shared between pools. If sharing `refund` addresses would be relevant then consider retrieving only the required amount (e.g. `cost - subsidy[poolId].value`).

Centrifuge: Fixed in commit ff6565f1.

Cantina Managed: Fix verified.

3.3.7 `uint96(...)` truncates

Severity: Informational

Context: Gateway.sol#L169, Gateway.sol#L226, Gateway.sol#L249

Description: `uint96()` truncates the parameter, without error. This is very unlikely to cause issues though because such large amounts of native tokens won't occur.

Recommendation: Consider using SafeCast.

Centrifuge: Fixed in commit d4bc0192.

Cantina Managed: Fix verified.

3.3.8 Across chains different share tokens can have the same vault address

Severity: Informational

Context: AsyncVaultFactory.sol#L28-L37, SyncDepositVaultFactory.sol#L37-L47

Description: As found by the project: right now the vault deployments are not using `CREATE2`, but the factories themselves are deployed deterministically. This means the vault addresses are based on the nonce, which means that across chains, different share tokens can have the same vault address. This can lead to user confusion.

Recommendation: Consider using `CREATE2` and a `salt` to deploy the vaults.

Centrifuge: Fixed in PR 560.

Cantina Managed: Fix verified.

3.3.9 Not all errors are custom errors

Severity: Informational

Context: PricingLib.sol#L183, PricingLib.sol#L209, CastLib.sol#L7, CastLib.sol#L16, CastLib.sol#L31, D18.sol#L42

Description: Most errors use custom errors however, some errors use strings. This is inconsistent.

Recommendation: Consider using custom errors everywhere.

Centrifuge: Fixed in commit 2597aa80.

Cantina Managed: Fix verified.

3.3.10 Different patterns for `file()`

Severity: Informational

Context: AxelarAdapter.sol#L43-L57, WormholeAdapter.sol#L43-L48, Guardian.sol#L126-L143

Description: Two different patterns are used for parameters of `file()`:

- `wireWormholeAdapter` uses twice `file(..., centrifugeId, wormholeId, ...)`.
- `wireAxelarAdapter` uses `file(..., axelarId, centrifugeId, ...)` + `file(.., centrifugeId, axelarId, ...)`.

The first pattern is easier because it only needs one `file()` function.

Recommendation: Consider using the first pattern also in `wireAxelarAdapter`.

Centrifuge: Fixed in PR 566.

Cantina Managed: Fix verified.

3.3.11 Could use `abi.encodeCall()`

Severity: Informational

Context: Spoke.sol#L539-L546

Description: Function `_safeGetAssetDecimals()` used `abi.encodeWithSignature()`. However `abi.encodeCall()` could also be used, which has additional checks.

Recommendation: Consider using `abi.encodeCall()`.

Centrifuge: Fixed in commit 4ddb6b08.

Cantina Managed: Fix verified.

3.3.12 Typos in comments

Severity: Informational

Context: GasService.sol#L48

Description: There are some typos present.

Recommendation: Consider making the following changes:

- `GasService`: take → taken.

Centrifuge: Fixed in commit 3f5c9de2.

Cantina Managed: Fix verified.

3.3.13 `_verifyAdmin()` check isn't foolproof

Severity: Informational

Context: FullDeployer.s.sol#L50-L59

Description: The `_verifyAdmin()` check isn't foolproof. A vault contract could comply with these checks, while these signers would not be able to control the safe. This could be done in the following ways:

- Have a 10 of 20 multisig where an attacker controls the other 12 accounts.
- Have a guard that prevents just signing by these addresses.
- Have module that allows alternative ways to allow transactions.
- Or a fake contract that always return true on a call to `isOwner()`.

See Staying Safe with Safe for more info.

Recommendation: Only use this check as a sanity check.

Centrifuge: Acknowledged. This is a check for us to double-check at deployment stage that the correct safe account is the configured one.

Cantina Managed: Acknowledged.

3.3.14 `timestampedPath` contains `block.chainid` twice

Severity: Informational

Context: JsonRegistry.s.sol#L41-L51

Description: `timestampedPath` contains `block.chainid` twice. According the rest of the string, the second instance should be `block.number`.

Recommendation: Consider replacing the second instance of `block.chainid` with `block.number`.

Centrifuge: Fixed in commit 57d0f8d2.

Cantina Managed: Fix verified.

3.3.15 Extra safeguard for `rely()` and `endorse()`

Severity: Informational

Context: HubDeployer.s.sol#L37-L65, Root.sol#L43-L46, Auth.sol#L25, Auth.sol#L25-L28

Description: In the deployment scripts, a large number of `rely()` statements are done. If accidentally an address is used for a contract that isn't deployed yet, then this isn't detected.

Note: the same issue is present with `root::endorse()`.

Recommendation: Consider having a wrapper function in the deployment scripts that check `user != address(0)` for `rely()` and `endorse()`. Alternatively this check can also be added in the functions themselves.

Centrifuge: Acknowledged. This is already checked in tests.

Cantina Managed: Acknowledged.

3.3.16 `adminSafe` not registered

Severity: Informational

Context: CommonDeployer.s.sol#L228, load_config.py#L55

Description: `CommonDeployer` has a comment that `register("adminSafe"...)` isn't necessary. However `load_vars.sh` has been refactored into `load_config.py`, which doesn't do the `register()`.

Recommendation: Consider adding `register("adminSafe"...)`.

Centrifuge: Fixed in commit 54783924. We only register in `register()` contracts that we deploy to later be able to read them from the json. The `adminSafe` is already in the json (in another section). I've removed the comment which seems to no longer be correct and creates more confusion.

Cantina Managed: Fix verified.

3.3.17 Explicit type conversion

Severity: Informational

Context: AsyncRequestManager.sol#L514-L515, AsyncRequestManager.sol#L519, AsyncRequestManager.sol#L538-L539, AsyncRequestManager.sol#L543

Description: Function `maxDeposit()` does an explicit type conversion from `uint128` to `uint256`, because `_maxDeposit()` returns an `uint128`. However the similar function `maxRedeem()` doesn't do this.

Recommendation: Consider also adding an explicit type conversion in function `maxRedeem()`.

Centrifuge: Fixed in commit f4234dd3.

Cantina Managed: Fix verified.