Centrifuge Security Review

Auditors

Gerard Persoon, Lead Security Researcher

0xLeastwood, Lead Security Researcher

Devtooligan, Security Researcher

Jonatas Martins, Associate Security Researcher

Report prepared by: Lucas Goiriz

August 6, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Centrifuge is the infrastructure that facilitates the decentralized financing of real-world assets natively on-chain, creating a fully transparent market which allows borrowers and lenders to transact without unnecessary intermediaries. The protocol aims to lower the cost of borrowing for businesses around the world, while providing DeFi users with a stable source of collateralized yield that is uncorrelated to the volatile crypto markets.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of liquidity-pools according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 11 days in total, Centrifuge engaged with Spearbit to review the liquidity-pools-internal protocol. In this period of time a total of **97** issues were found.

Summary

Project Name	Centrifuge
Repository	liquidity-pools
Commit	be582c7c
Type of Project	DeFi
Audit Timeline	Jul 15th to Jul 26th
Two days fix period	Jul 26 - Jul 28

The Spearbit team reviewed Centrifuge's liquidity-pools changes holistically on commit hash 7b367a604bcf3bf7de5afb1bcfd956f922669779 and determined that all issues were resolved and no new issues were identified.

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	1	1	0
Medium Risk	2	2	0
Low Risk	25	16	9
Gas Optimizations	8	7	1
Informational	60	36	24
Total	**97**	**63**	**34**

5 Findings

5.1 Critical Risk

5.1.1 Centrifuge router can perform untrusted actions on behalf of open vaults

Severity: Critical Risk

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

Description: The Centrifuge router enables several features such as multicall actions, permissionlessly claiming, ERC20 wrapping and locking requests. In order to allow the router contract to perform certain actions to the vault, it must first be added as an endorsed operator. `open()` must be called to enable vault interactions and left as such to allow for permissionlessly claiming.

```
function open(address vault) public protected {
    IERC7540Vault(vault).setEndorsedOperator(_initiator(), true);
}
```

However, there are some serious concerns with this because if a user has an approval for the vault, anyone can `requestDeposit()` on behalf of this `owner` and pass a `controller` parameter for which they control. There is no validation on the vault side because `isOperator[owner][msg.sender]` holds true as the caller is the router itself. The same applies to `requestRedeem()`, allowing tranch tokens to be redeemed unwillingly in which the `controller` is any arbitrary account.

```
function requestDeposit(address vault, uint256 amount, address controller, address owner, uint256
↪  topUpAmount)
    external
    payable
    protected
{
    (address asset,) = poolManager.getVaultAsset(vault);
    if (owner == address(this)) {
        _approveMax(asset, vault);
    }

    _pay(topUpAmount);
    IERC7540Vault(vault).requestDeposit(amount, controller, owner);
}
```

Recommendation: To prevent attackers from spoofing deposits/redemptions through the router contract, `require(owner == _initiator() || owner == address(this), "CentrifugeRouter/invalid-owner");` will ensure that only the asset owner has access to these actions.

It's worth documenting the expected uses of `open()` enabling interactions with the vault so users only use it where necessary.

Centrifuge: Fixed in commit 3b21650d.

Spearbit: Fixed.

5.2 High Risk

5.2.1 Assets can get stuck in `TransferProxy`

Severity: High Risk

Context: TransferProxyFactory.sol#L18

Description: The function `transferAssets()` uses the following code to get the tokens from the `TransferProxy`:

However there is no allowance set in the `TransferProxy` so this will always fail.

Recommendation: Set an allowance for the `poolManager` in `TransferProxy`.

Centrifuge: Fixed in commit 5e11282a.

Spearbit: Fixed.

5.3 Medium Risk

5.3.1 Inconsistency in message library between rust and solidity implementations

Severity: Medium Risk

Context: BytesLib.sol#L68, PoolManager.sol#L158-L163

Description: Message data is passed between Ethereum and Centrifuge through the gateway contract. Incoming messages are dispatched when quorum has been reached. The first byte indicates the intended action to be executed on a target manager contract. Each manager contract implements a `handle()` function which decodes this data according to the message ID. There are some inconsistencies in the formatting of data that is passed from Centrifuge chain to Ethereum.

The inconsistencies apply to the following messages (as per latest commit):

- There is an extra 32 bytes that is expected from Centrifuge but not used in:
  - `TransferAssets`.
  - `TransferTrancheTokens`.
- `UpdateTrancheHook` is missing a 16 byte `tranchId` parameter.
- `UpdateCentrifugeGasPrice` should pass two parameters, a `uint128` and `uint64` when only a `uint64` is being provided.
- `DisputeMessageRecovery` is missing the adapter address.
- `RecoverTokens` should decode the `amount` parameter to a `uint128` instead.
- Two types of addresses are used: a 20 byte address and a 32 bytes address, while the Solidity `toAddress()` uses a 32 byte address.

Recommendation: Maintain consistency between the message library implementations on Centrifuge chain and Ethereum. Doublecheck the used address formats.

The `PoolManager` contract needs to be updated to enable tranch/asset token transfers and comply with inbound messages from Centrifuge.

```
} else if (call == MessagesLib.Call.TransferAssets) {
    handleTransfer(message.toUint128(1), message.toAddress(17), message.toUint128(49));
} else if (call == MessagesLib.Call.TransferTrancheTokens) {
    handleTransferTrancheTokens(
        message.toUint64(1), message.toBytes16(9), message.toAddress(34), message.toUint128(66)
    );
```

Centrifuge: Fixed in commit 223a0f36.

Spearbit: Fixed.

5.3.2 Frozen users may transfer tranche tokens

Severity: Medium Risk

Context: PoolManager.sol#L96-L117

Description: A user whose account has been frozen may freely transfer tranche tokens by calling `transferTrancheTokens()` on the `PoolManager`.

If a frozen user were to call `transferTrancheTokens()` on the `CentrifugeRouter`, the transaction would revert.

```
/// @inheritdoc ICentrifugeRouter
function transferTrancheTokens(
    address vault,
    Domain domain,
    uint64 chainId,
    bytes32 recipient,
    uint128 amount,
    uint256 topUpAmount
) public payable protected {
    SafeTransferLib.safeTransferFrom(IERC7540Vault(vault).share(), _initiator(), address(this), amount);
    _approveMax(IERC7540Vault(vault).share(), address(poolManager));
    _pay(topUpAmount);
    IPoolManager(poolManager).transferTrancheTokens(
        IERC7540Vault(vault).poolId(), IERC7540Vault(vault).trancheId(), domain, chainId, recipient,
↪   amount
    );
}
```

The frozen user is prevented from calling this function because the restriction manager is checked in the call to the hook during the transfer of the tranche tokens from the user to the router.

However, when a user calls the same function on the `PoolManager` the tokens are immediately burned and there is no call to the hook or restriction manager.

Another problem stemming from this missing check is that a transfer may be initiated to a recipient that is either frozen or not a member. This process would ultimately fail in the last step when the tokens were attempted to be transferred to the recipient, however the sender's shares are already burned and it was a waste of gas and time to process the round-trip message to the Centrifuge chain.

Proof of concept:

```
function testPOC_frozenAccountTransfer(uint128 amount) public {
    uint64 validUntil = uint64(block.timestamp + 7 days);
    address destinationAddress = makeAddr("destinationAddress");
    vm.assume(amount > 0);

    address vault_ = deploySimpleVault();
    ERC7540Vault vault = ERC7540Vault(vault_);
    ITranche tranche = ITranche(address(ERC7540Vault(vault_).share()));

    centrifugeChain.updateMember(vault.poolId(), vault.trancheId(), destinationAddress, validUntil);
    centrifugeChain.updateMember(vault.poolId(), vault.trancheId(), address(this), validUntil);
    assertTrue(tranche.checkTransferRestriction(address(0), address(this), 0));
    assertTrue(tranche.checkTransferRestriction(address(0), destinationAddress, 0));

    // Fund this address with samount
    centrifugeChain.incomingTransferTrancheTokens(
        vault.poolId(), vault.trancheId(), uint64(block.chainid), address(this), amount
    );
    assertEq(tranche.balanceOf(address(this)), amount);

    // fails for invalid tranche token
    uint64 poolId = vault.poolId();
    bytes16 trancheId = vault.trancheId();

    centrifugeChain.freeze(poolId, trancheId, address(this));
    assertFalse(tranche.checkTransferRestriction(address(this), destinationAddress, 0));

    // Approve and transfer amount from this address to destinationAddress
    tranche.approve(address(poolManager), amount);
    poolManager.transferTrancheTokens(
        vault.poolId(), vault.trancheId(), Domain.EVM, uint64(block.chainid),
↪   destinationAddress.toBytes32(), amount
    );
    assertEq(tranche.balanceOf(address(this)), 0);
}
```

Recommendation: Call `tranche.checkTransferRestriction` which will revert if the sender or recipient are frozen or if the recipient is not a member.

```
  /// @inheritdoc IPoolManager
  function transferTrancheTokens(
      uint64 poolId,
      bytes16 trancheId,
      Domain destinationDomain,
      uint64 destinationId,
      bytes32 recipient,
      uint128 amount
  ) external {
      ITranche tranche = ITranche(getTranche(poolId, trancheId));
      require(address(tranche) != address(0), "PoolManager/unknown-token");

+     require(tranche.checkTransferRestriction(msg.sender, address(uint160(uint256(recipient))),
↪   amount));
      tranche.burn(msg.sender, amount);
```

Centrifuge: We have already implemented a fix for that. Also please note: in case a user is frozen on evm, their address on Centrifuge chain is also frozen and not a member anymore. So even if they would try to exploit the transfer, they wouldn't be able to use their own accounts on centrifuge chain and would need to find another account that passed KYC to participate.

Fixed in commit bc1c02e2.

Spearbit: Fixed adding the `_onTransfer()` function in `Tranche.burn()`.

5.4 Low Risk

5.4.1 Users can delay claims to avoid being frozen

Severity: Low Risk

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

Description: Users can avoid issues related to membership expiry and being explicitly frozen by delaying their call to `claimCancelRedeemRequest()` for as long as possible. This allows frozen holders time to figure out another path as they control the recipient of tranche tokens.

The same applies to `claimCancelDepositRequest()` and frozen assets. It's a path that exploiters may use to "store" stolen funds for an arbitrary amount of time.

Recommendation: It may be worth making this permissionless and setting the recipient of the claim upon the initial request.

Centrifuge: This would not work for smart contract integrations, since the contracts would directly receive tokens and would not be able to update their bookkeeping. Acknowledged.

Spearbit: Acknowledged.

5.4.2 `validateController` check is vulnerable if router is ever an operator of itself

Severity: Low Risk

Context: ERC7540Vault.sol#L227, ERC7540Vault.sol#L234, ERC7540Vault.sol#L248

Description: The `IERC7540Vault(vault).isOperator(controller, address(this))` check added to the router's claim functions becomes redundant if the router becomes an operator of itself. As of a result, permissionless claims can be made for any `controller` account, regardless if they called `open()` to enable vault interactions through the router contract.

Recommendation: Disallow an address to be an operator of itself by checking for `isOperator[x][x]` within `setEndorsedOperator()`/`setOperator()`/`authorizeOperator()` functions.

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.4.3 `messageHandlers()` can potentially send system messages

Severity: Low Risk

Context: Gateway.sol#L111-L112

Description: Via function `file()` the `messageHandlers()` can be set for all message types, including the types defined in `enum Call`.

In `_dispatch()` they will only be used if not matched with any of the detected types. However this includes `id == 0`.

In `send()` it will allow adding `messageHandlers()` to send system messages, which could compromise the integrity of the system and circumvent checks.

> **Note:** this will require several errors/mistakes to be abused: malicious code in a `MessageHandler`, malicious id set in the call to `file()` and updates executed by an admin.

Recommendation: In `file()` make sure no handlers can be added for the ids in `enum Call`.

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.4.4 Redundant line in `Deployer`

Severity: Low Risk

Context: Deployer.sol#L77, GasService.sol#L14

Description: There doesn't seem to be an `auth` function in `gasService` that is called by `poolManager`.

Recommendation: Doublecheck the conclusion and remove the `gasService.rely(address(poolManager))` if not necessary.

Centrifuge: Fixed in commit 2aebdd6b.

Spearbit: Fixed.

5.4.5 `emit ExecuteMessage()` can emit the proof

Severity: Low Risk

Context: Gateway.sol#L178-L192

Description: The `emit ExecuteMessage(payload,...)` emits `payload` which contains either the message or the proof (prefixed with header 1), depending on the order in which messages are received.

It seems most logical to always `emit` the message.

Recommendation: Consider changing the code to something like:

```
  // Handle message
  if (isMessageProof) {
      _dispatch(state.pendingMessage, false);
+     emit ExecuteMessage(state.pendingMessage, msg.sender);
  } else {
      _dispatch(payload, false);
+     emit ExecuteMessage(payload, msg.sender);
  }
  if (state.votes.isEmpty()) {
      delete state.pendingMessage;  // do this after the emit
  }
- emit ExecuteMessage(payload, msg.sender);
```

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.4.6 Update of `poolManager()` will cripple `TransferProxy`s

Severity: Low Risk

Context: Gateway.sol#L104, InvestmentManager.sol#L49, TransferProxyFactory.sol#L8

Description: Other contracts have a way to update the `poolManager()`. If they would do that then the old `poolManager()` won't be able to continue functioning normally. So `transferAssets()` can't be used anymore and the assets will stay stuckin the `TransferProxy`.

If this is detected quickly this can most likely be fixed by reverting the change and then the impact will be limited.

Recommendation: Create a way to recover the `asset`s or create a way to update the `poolManager` in the `TransferProxy`s & `TransferProxyFactory`.

> **Note:** with `newTransferProxy()` being permissionless, there might be a lot of `TransferProxy`s to update.

Centrifuge: Fixed in commit 5e11282a by adding `recoverTokens()`.

Spearbit: Fixed.

5.4.7 `updateMember()` doesn't check for `address(0)`

Severity: Low Risk

Context: RestrictionManager.sol#L120-L128

Description: `updateMember()` doesn't check for `address(0)` like `freeze()` does. So the membership of `address(0)` could be limited by accident. Currently this is not a problem because:

- The membership is not checked with `from` (e.g. with `mint`) in `checkERC20Transfer()`.
- The membership is not checked with `to` (e.g. with `burn`) because `burn()` doesn't call `checkERC20Transfer()`.

However if the logic changes, for example due to the issue "Frozen users may transfer tranche tokens" this might become relevant.

Recommendation: Consider adding a check for address(0) in `updateMember()`. Alternatively `root.endorse(address(0));` could be done.

Centrifuge: Fixed in commit c53a9c05 by changing the logic of `checkERC20Transfer()`.

Spearbit: Fixed.

5.4.8 Gas estimate is calculated incorrectly in `Gateway`

Severity: Low Risk

Context: Deployer.sol#L46-L47, Gateway.sol#L341-L342

Description: The gas estimate calculation through the `Gateway.estimate()` function is incorrect, the variables `proofCost` and `messageCost` are swapped. However, there's no current impact on the code because both values are set to be equal in the `deployment` scripts.

Recommendation: It's recommended to fix the code using the following:

```
  function estimate(bytes calldata payload) external view returns (uint256[] memory tranches, uint256
↪  total) {
      bytes memory proof = abi.encodePacked(uint8(MessagesLib.Call.MessageProof), keccak256(payload));
-     uint256 proofCost = gasService.estimate(payload);
-     uint256 messageCost = gasService.estimate(proof);
+     uint256 proofCost = gasService.estimate(proof);
+     uint256 messageCost = gasService.estimate(payload);
      tranches = new uint256[](adapters.length);
      // ...
  }
```

Centrifuge: Fixed in commit 7509aa58.

Spearbit: Fixed.

5.4.9 `veto()` doesn't undo all actions of `endorse()`

Severity: Low Risk

Context: Root.sol#L62-L65

Description: An update of `endorsements[]` has immediate effect on most operations, however not for the effects of `setEndorsedOperator()`.

Recommendation: Develop an approach to retrieve all the vaults for which `setEndorsedOperator()` has been done and execute `setOperator(user,false)` on these vaults.

Centrifuge: Any `veto()` call would be executed through a `spell` (migration contract), that would then also include the required `setOperator(user,false)` calls.

Spearbit: Acknowledged.

5.4.10 `Root` contract trusts the `Gateway` contract

Severity: Low Risk

Context: Deployer.sol#L100, Root.sol#L112-L126

Description: The `Root` contract trusts the `Gateway` contract, which is used to allow `handle()`. The `Gateway` contract can also be replaced so this might lead to unexpected attack vectors.

Recommendation: Consider using a dedicated authorization method for the `handle()` related functions in `Root`:

- `handle`
- `scheduleRely`
- `cancelRely`
- `recoverTokens`

And then remove `gateway.rely(address(root));` from the `deployer`.

Centrifuge: This is using the `auth` pattern to simplify extensions of the contracts in the future. The contracts are `immutable` and any future changes to the `Gateway` contract would be audited and go through a governance vote before being executed through the timelock.

Spearbit: Acknowledged.

5.4.11 `transferTrancheTokens()` allows sending to non existing chains

Severity: Low Risk

Context: PoolManager.sol#L96-L102

Description: `transferTrancheTokens()` doesn't check the `destinationId` (`chainId`). So the tokens might end up at a non-existing chain and be lost. Several other bridge based protocols check for allowed destinations.

Recommendation: Consider checking the `destinationId` to prevent loss of tokens or (in case it is checked at the Centrifuge chain) loss of gas.

Centrifuge: Messages always go through Centrifuge Chain. Centrifuge Chain knows which chain IDs are valid, so can trigger a return transfer if the chain id target is invalid.

However, in the end it is the same as specifying the destination of a normal ERC20 transfer: it is up to the user to ensure the destination is valid.

Spearbit: Acknowledged.

5.4.12 Updating set of active adapters does not always clear votes

Severity: Low Risk

Context: Gateway.sol#L169

Description: Centrifuge governance has the ability to configure new sets of adapters, where quorum is always the length of this new set. When new adapters are enabled, the old ones are replaced but as long as the number of adapters does not decrease, pre-existing message votes will continue to persist.

As old adapters are changed for a reason, their old votes might not always be trusted. If several adapters are compromised, then the set might be replaced with one of the same length. However, the adapters which inherit the same ID are shown to have voted for potentially malicious messages because votes are not cleared. This might lead to a situation where enough votes are collected to process a malicious message.

Recommendation: Consider clearing votes whenever the set of active adapters is updated.

```
if (adapter.activeSessionId != state.sessionId) {
    // Clear votes from previous session
    delete state.votes;
    state.sessionId = adapter.activeSessionId;
}
```

Centrifuge: Fixed in commit a375028a.

Spearbit: Fixed.

5.4.13 Spam of cancel requests might impact project

Severity: Low Risk

Context: InvestmentManager.sol#L141

Description: To request deposits and withdrawals, users must be validated by the tranche hook. Considering the `Restriction Manager` as the hook, users need to be unfrozen and members. The redeem cancellation also checks if the user is valid, but this check is missing in the `cancelDepositRequest()` function in the `Investment-Manager` contract. Additionally, there is no validation to check if the user has any deposit requests, allowing a user without a pending request call `cancelDepositRequest()`.

This opens an opportunity for a spam attack, where users spam multiple cancel deposit requests through different accounts, wasting gas from the `Gas Service` and potentially harming the Centrifuge chain that will need to deal with amount of invalid request.

Another impact depends on how the Centrifuge chain deals with the zero-amount cancellation, leading to a user in a stale state where they can't make any deposits.

This attack might not be possible on the Ethereum mainnet because of the gas fees but can be used on other L2 chains where the gas fees are lower.

> **Note:** spam transactions can also be done via `transferAssets()` and `transferTrancheTokens()`.

Recommendation: The recommendation is to validate if the user has `pendingDepositRequest` and if they are allowed.

Centrifuge: The restriction check is missing for `cancelDepositRequest()`, since assets are not restricted tokens. To avoid spamming, we intentionally introduced the shouldRefuel function in the gasService and might add more strict conditions for cross-chain requests that the gas service is paying for (example: `depositValue > threshold`). In case users are spamming through the CentrifugeRouter, they would still need to pay for their own gas.

We think that introducing a check whether users have outstanding requests before calling cancellation would still improve the UX.

Fixed in commit 40498fb3.

Spearbit: Fixed.

5.4.14 Slot name could lead to collisions

Severity: Low Risk

Context: CentrifugeRouter.sol#L27, Gateway.sol#L27

Description: Slotnames are used for `transient` storage slots. There is a small chance that other codebases use the same name and a small chance that transactions are combined for example via ERC4337.

Recommendation: Consider using a more specific name for the slot, to prevent accidental collisions, for example:

- `centrifuge-initiater`
- `centrifuge-quota`

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.4.15 `recoveries[][]` could be left when an adapter is removed

Severity: Low Risk

Context: Gateway.sol#L132-L134, Gateway.sol#L258

Description: After an adapter is removed, there still could be a `recoveries[][]` entry linked to this adapter.

If an adapter is installed again later, these `recoveries[][]` entries can be used again, which might be unwanted.

> **Note:** `executeMessageRecovery()` can't be used on the `recoveries[][]` entries of removed adapters because that is blocked by `_handle()`.
>
> **Note:** `_disputeMessageRecovery()` does allow removing `recoveries[][]` entries of removed adapaters.

Recommendation: Consider removing `recoveries[][]` entries when removing an adapter. If that is done consider `_disputeMessageRecovery()` to only work with valid adapters.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.4.16 `getTranchePrice()` can return a price of 0

Severity: Low Risk

Context: InvestmentManager.sol#L371-L382, PoolManager.sol#L432

Description: Function `getTranchePrice()` can return a price of `0`, if it is queried before the first `updateTranchePrice()`is done. This would give incorrect results in `convertToShares()` and `convertToAssets()`.

Recommendation: Consider checking `computedAt` in function `getTranchePrice()` and revert if it still `0`.

Centrifuge: Fixed in commit 398a0623.

Spearbit: Fixed.

5.4.17 Tranche tokens of a frozen account may be burned

Severity: Low Risk

Context: Deployer.sol#L72, InvestmentManager.sol#L288, PoolManager.sol#L107, RestrictionManager.sol#L67

Description: A user whose account has been frozen can still have their tranche tokens burned. This is inconsistent with the mint function, which prohibits frozen users from minting. As burning can be considered a transfer to the zero address, a frozen account should not be able to perform this action.

This behavior may violate regulations in some jurisdictions. If an account is frozen for regulatory reasons, burning tokens might be prohibited, similar to how some stablecoins (e.g., USDC) disallow burning of tokens by users on their deny list.

Recommendation: Call the `_onTransfer` hook in the `Tranche.burn()` function similar to how it is called in `mint()`. This way, if the restriction manager were in place for the tranche, a transaction originating from a frozen user would revert.

```
  function burn(address from, uint256 value) public override(ERC20, ITranche) {
      super.burn(from, value);
+     _onTransfer(from, address(0), value);
  }
```

The function `fulfillRedeemRequest()` will keep working because it burns from `escrow` and `escrow` is `endorsed`, so it will not be blocked by `checkERC20Transfer()`.

Centrifuge: Fixed in commit bc1c02e2.

Spearbit: Fixed.

5.4.18 `Vault` address is not verified to be valid in InvestmentManager

Severity: Low Risk

Context: InvestmentManager.sol#L239-L247, InvestmentManager.sol#L265-L273, InvestmentManager.sol#L294-L302, InvestmentManager.sol#L317-L321, InvestmentManager.sol#L334-L339, PoolManager.sol#L436-L443

Description: The `vault` address is not verified to be valid in the following functions: `fulfillDepositRequest()`, `fulfillRedeemRequest`, `fulfillCancelDepositRequest()`, `fulfillCancelRedeemRequest()`, `triggerRedeemRequest()`. The `vault` could have been removed or recreated.

> **Note:** `getVault()` doesn't check the validity of the `vault`.

Recommendation: It's recommended to validate the `vault` address in each one of the functions mentioned. This could be done by checking the validity in `getVault()`.

Centrifuge: Fixed in commit 09a2eb04.

Spearbit: Fixed.

5.4.19 Wrap / Unwrap revert on zero amount

Severity: Low Risk

Context: CentrifugeRouter.sol#L107-L119, CentrifugeRouter.sol#L309

Description: The `wrap()` and `unwrap()` functions revert if the amount is zero. Normally users would not call these functions with a zero amount, and the same goes for `openLockDepositRequest()` and `claimRedeem()`.

However in an automated flow (for example in combination with `multicall()`), this could happen. And then there is no reason to `revert` on a zero amount.

For reference:

- Morpho ERC20WrapperBundler does revert on zero amounts.
- OZ ERC20Wrapper does not revert on zero amounts.

Recommendation: Consider allowing zero amounts for both `wrap()` and `unwrap()`.

Centrifuge: Fixed in commit 1c3f57d8 by only wrapping when necessary.

Spearbit: Fixed.

5.4.20 `_handleRecovery()` can be done repeatedly by one adapter

Severity: Low Risk

Context: Gateway.sol#L142

Description: A single adapter can trigger `_handleRecovery()` due to the lack of quorum checks. If this adapter is malicious, it could repeatedly call to `DisputeMessageRecovery()`.

Recommendation: It's recommended to validate if the message reaches quorum before calling `_handleRecovery()`. This would prevent repeated calls to `DisputeMessageRecovery()`.

Centrifuge: Acknowledged. I don't think the suggestion works. The dispute recovery mechanism itself intentionally doesn't go through the quorum, any single adapter can dispute any mechanism. Indeed they can repeatedly do this but there is no (simple) solution for that.

Spearbit: Acknowledged.

5.4.21 Unsafe cast of `addresses.length`

Severity: Low Risk

Context: Gateway.sol#L68

Description: The code truncates the `address.length` to `uint8`. If addresses.length were 257, this would result in 1. However, the entire addresses array is assigned to adapters later (see Gateway.sol#L92). Afterward, in functions like send(), the full adapters.length is used.

Recommendation: It's recommended to have `quorum_` as a `uint256`, or use SafeCast.

Centrifuge: Fixed in commit bb9adce5.

Spearbit: Fixed.

5.4.22 `Gateway` could call `handle()` in its own contract

Severity: Low Risk

Context: Gateway.sol#L235

Description: The `Gateway` contract itself contains a `handle()` function that could potentially be called, allowing for an additional layer of recursion. This scenario is only possible if the `Gateway` address is added to the `messageHandlers[]` array through the `file()` function.

Recommendation: It's recommended to prevent these errors: either check in `file()` or right before calling `handle()`.

Centrifuge: In theory -- yes, it can call itself but only if some admin intentionally add `gateway` as a handler for a specific message. Highly unlikely to happen because if that's the case, the message can be handled within the `gateway.handle()` itself. It's possible but I don't see any security or any other implications.

Spearbit: Acknowledged.

5.4.23 Some functions of `CentrifugeRouter` don't check `vault` validity

Severity: Low Risk

Context: CentrifugeRouter.sol#L157, CentrifugeRouter.sol#L168, CentrifugeRouter.sol#L174, CentrifugeRouter.sol#L189, CentrifugeRouter.sol#L218, CentrifugeRouter.sol#L224, CentrifugeRouter.sol#L256, CentrifugeRouter.sol#L273

Description: Several functions of CentrifugeRouter don't explicitly check that the vault is valid, so they could accidentally interact with an old or invalid vault.

Other functions use `poolManager.getVaultAsset(vault)`, which checks the vault is valid. The impact seems to be limited.

Recommendation: Consider checking the validity of the vault.

Centrifuge: If `_pay()` is called that value of quota in `Gateway` will be zeroed after the transaction is completed. And here we are talking in situations where the address of the `vault` leads to some contract with no-op implementation of the functions calls.

In case where you `removeVault` and `deployVault` then, there are permission which won't be presented hence transaction calls using that address will fail.

Spearbit: Acknowledged.

5.4.24 In transit funds inaccessible after removing a vault or contract updates

Severity: Low Risk

Context: CentrifugeRouter.sol#L122-L128, CentrifugeRouter.sol#L136-L145, InvestmentManager.sol#L35, PoolManager.sol#L399-L415

Description: When calling `removeVault()`, there could still be funds in transit, while being stored in one of the `escrows`. The administration is kept in `lockedRequests[][]` and `investments[][]`. `unlockDepositRequest()` and `executeLockedDepositRequest()` will fail at `getVaultAsset()` when trying to access these funds. The old vault can no longer access functions from the `investmentManager`.

A similar issue occurs if `poolManager` or `Gateway` are upgraded. Then `CentrifugeRouter` will not function anymore and has to be replaced. Any funds still in `transit` are difficult to access.

The funds can be recovered from `escrow` by Centrifuge governance through a `spell` that could get approval to transfer locked funds from the `escrow` contract and distribute them back to the user.

Although function `removeVault()` and contract upgrades are authorised there are likely always funds in transit trough the protocol.

Recommendation: Consider having a way to allow the owner to retrieve the assets of removed `vaults` and upgrade contracts and clean up the administration.

For `CentrifugeRouter` double check if it is useful to allow upgrades of `poolManager` or `Gateway`.

Centrifuge: This is an admin only method, there is no way to call this besides by a migration contract (`spell`) becoming a `ward` through the `timelock` in the `Root`, and then executing this. It would only be used if there are no locked requests. The same applies to investments pending in the `InvestmentManager` (which are stored by the vault addresses).

In case user funds get stuck in the `escrow`, they can also be recovered by Centrifuge governance through a `spell` that could get approval to transfer locked funds from the `escrow` contract and distribute them back to the user.

Spearbit: Acknowledged.

5.4.25 Very low number of minimal decimals

Severity: Low Risk

Context: PoolManager.sol#L38

Description: The value for `MIN_DECIMALS` is set to `1`, but the lowest real-life example has 2 decimals, as we can see in weird-erc20. Having a lower `MIN_DECIMALS` typically makes rounding errors more severe.

Recommendation: It's recommended to set the `MIN_DECIMALS` to at least `2`.

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.5 Gas Optimization

5.5.1 Function `checkERC20Transfer()` can be optimized

Severity: Gas Optimization

Context: RestrictionManager.sol#L62-L81

Description: The function `checkERC20Transfer()` can be optimized to save some gas.

Recommendation: Consider changing the code to something like the following:

```
function checkERC20Transfer(/*...*/) /*...*/ {
    // ...
    if (root.endorsed(to))
         return true;
    if (uint128(hookData.to).getBit(FREEZE_BIT) == true)
        return false;
    if (abi.encodePacked(hookData.to).toUint64(0) < block.timestamp )
        return false;
    return true;
}
```

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.5.2 `transfer()` of `TransferProxy` doesn't need to specify `amount`

Severity: Gas Optimization

Context: TransferProxyFactory.sol#L17-L18

Description: The function `transfer()` of `TransferProxy` allows specifying an amount that is larger than the available assets. This will result in a revert in `poolManager.transferAssets()` and thus wastes gas. There doesn't seem to be a need to be able to specify the `amount`.

Recommendation: Consider querying the asset `balance` and use that to call `poolManager.transferAssets()`. If the `balance == 0` the function could return directly.

Alternatively you could use the `min()` of the `amount` and the `balance`.

Centrifuge: Fixed in commit 5e11282a.

Spearbit: Fixed.

5.5.3 `getLSBits()` and `getMSBits()` can be simplified

Severity: Gas Optimization

Context: BitmapLib.sol#L20, BitmapLib.sol#L25

Description: The functions `getLSBits()` and `getMSBits()` are only used with 128 bits so these functions can be simplified to use less gas. Additonally `getLSBits()` and `getMSBits()` can then return an `uint128` which saves a typecast.

Recommendation: Consider simplifying these functions.

Centrifuge: Removed in commit 7a034b96.

Spearbit: Fixed.

5.5.4 `shares.toUint128()` done multiple times

Severity: Gas Optimization

Context: InvestmentManager.sol#L471-L472, InvestmentManager.sol#L472

Description: The `shares.toUint128()` conversion is performed multiple times. A small amount of gas might be saved by storing the result in a variable.

Recommendation: It's recommended to cache the `shares.toUint128()`.

Centrifuge: Fixed in commit 40498fb3.

Spearbit: Fixed.

5.5.5 Redundant check in `_handleRecovery()`

Severity: Gas Optimization

Context: Gateway.sol#L128-L134, Gateway.sol#L243

Description: The `msg.sender` check in `_handleRecovery()` is redundant with the check on the first line of `_handle()` which is the only place `_handleRecovery()` is called from.

Recommendation: Remove the redundant check.

```
  function _handleRecovery(bytes memory payload) internal {
      // ...
      if (MessagesLib.messageType(payload) == MessagesLib.Call.InitiateMessageRecovery) {
-         require(activeAdapters[msg.sender].id != 0, "Gateway/invalid-sender");
          require(activeAdapters[adapter].id != 0, "Gateway/invalid-adapter");
          // ...
        } else if (/*...*/) {
          // ...
      }
  }
```

Centrifuge: Fixed in commit 9610f206.

Spearbit: Fixed.

5.5.6 For loops could be shorter in `ArrayLib`

Severity: Gas Optimization

Context: ArrayLib.sol#L7

Description: The functions in `ArrayLib` only have to loop through `adapters.length` elements, because that it the number of elements that is in use.

Recommendation: Consider limiting the the loop of `countNonZeroValues()`, `decreaseFirstNValues()` and `isEmpty()`. For example in the following way.

```
//Gateway.sol
uint256 n = adapters.length; // extra storage read if only done for this purpose
// ...
if (state.votes.isEmpty(n)) // ...
// ...

//ArrayLib.sol
function isEmpty(uint16[8] memory arr, uint256 n) internal pure returns (bool) {
    require (n <= 8, ....); // to be sure, although this is also checked in `file()`
    for (uint256 i; i < n; i++) {
        if (arr[i] != 0) return false;
    }
    return true;
}
```

However, because that array is very short anyway, perhaps it doesn't save any gas in the end. The code also might not be intuitive because you work on one array but pass the boundries from another array.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.5.7 `msg.sender == _initiator()` in functions with `modifier protected()`

Severity: Gas Optimization

Context: CentrifugeRouter.sol#L57

Description: The `modifier protected()` enforces that `msg.sender == _initiator()`. So within `functions` that have this `modifier`, `msg.sender` could be used instead of `_initiator()`. That would save some gas.

Recommendation: Consider directly using `msg.sender` in functions with the `modifier protected()`.

Centrifuge: The inspiration is from morpho-blue. Morpho's bundler code needed to use `_initiator()` to support callbacks, which we don't have. Using `msg.sender` directly makes sense at this point. Fixed in commit 8ff21270.

Spearbit: Fixed.

5.5.8 Array lenghts in `for` loops can be cached

Severity: Gas Optimization

Context: ArrayLib.sol#L13, ArrayLib.sol#L26, ArrayLib.sol#L7, CentrifugeRouter.sol#L320, Gateway.sol#L345, TrancheFactory.sol#L49

Description: Caching array lengths: `....length` could save some gas, especially in `for` loops.

Recommendation: It's recommened to cache array lengths.

Centrifuge: Fixed in commit c3a0959d.

Spearbit: Fixed.

5.6 Informational

5.6.1 `TransferProxy` could be deployed with `create2`

Severity: Informational

Context: TransferProxyFactory.sol#L41

Description: TransferProxy could be deployed with `CREATE2`. The advantage is that the address can be determined before deployment and that the address will be the same on all chains.

Recommendation: Consider using `CREATE2`. Consider having a function to calculate the deployement address based on the `destination`

Centrifuge: Fixed in PR 54.

Spearbit: Fixed.

5.6.2 `disallowAsset()` doesn't fully block the use of an asset

Severity: Informational

Context: InvestmentManager.sol#L493-L516, InvestmentManager.sol#L518, InvestmentManager.sol#L61-L72, InvestmentManager.sol#L99-L109, PoolManager.sol#L192

Description: Function `disallowAsset()` doesn't disable the `vaults` that use this asset. The functions `requestDeposit()` and `requestRedeem()` do check `isAllowedAsset()`. However in further stages this check isn't done anymore. This could lead to using assets that are not longer supported.

This could potentially violate strict invariant checks.

Recommendation: Be aware of this situation and make sure sufficient liquidity is available.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.3 The check for `supportsInterface()` can allow non compliant hooks

Severity: Informational

Context: PoolManager.sol#L224

Description: The check for `supportsInterface()` can return `true` even it the hook doesn't support ERC165.

For example if it has a fallback function that returns `true` on every function call.

Recommendation: Consider using the OZ ERC165Checker.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.4 `abi.encodeWithSelector()` can be replaced with `abi.encodeCall()`

Severity: Informational

Context: ICentrifugeRouter.sol#L237-L241, SafeTransferLib.sol#L17, SafeTransferLib.sol#L27, SafeTransferLib.sol#L37

Description: In several places `abi.encodeWithSelector()` is used. However `abi.encodeCall()` would be safer because it also checks for the correct types. See the Solidity manual.

Recommendation: Consider replacing `abi.encodeWithSelector()` with `abi.encodeCall()`.

Centrifuge: Fixed in commits c53a9c05 and 5b4c17f3.

Spearbit: Fixed.

5.6.5 Typo in `ICentrifugeRouter`

Severity: Informational

Context: ICentrifugeRouter.sol#L245

Description: A comment for `multicall()` contains a typo.

Recommendation: Consider changing the comment to:

```
- ///         router.multical{value: msgValue}(calls);
+ ///         router.multicall{value: msgValue}(calls);
```

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.6.6 Missing function interfaces

Severity: Informational

Context: CentrifugeRouter.sol#L27-L31, Gateway.sol#L27-L43, Guardian.sol#L13-L15, IPoolManager.sol#L53, InvestmentManager.sol#L29-L35, PoolManager.sol#L41-L49, PoolManager.sol#L53, Root.sol#L20-L23

Description: For several public variables there is no function interface defined in the interface files.

Recommendation: Doublecheck the usefulness of these variables being public. Change them to `internal` if not useful. Consider adding function interfaces in the interface files.

Centrifuge: Fixed in several places in PR 15.

Spearbit: Fixed.

5.6.7 Parameters of `fulfillCancelDepositRequest()` not clear

Severity: Informational

Context: InvestmentManager.sol#L294-L300

Description: It is not clear what the difference is between `assets` and `fulfillment` in `fulfillCancelDepositRequest()`.

The project explained:

> It is mainly for precision loss. There can be multiple steps in between on Centrifuge Chain. So basically assets is the rounded down option, that the user actually gets, and fulfillment is the rounded up version, that we decrease pendingDepositRequest by (which we need to ensure goes to 0 at the end).

Recommendation: To make it clear consider changing the code to:

```
  function fulfillCancelDepositRequest(
      // ...
-     uint128 assets,
+     uint128 assetsDown,
-     uint128 fulfillment
+     uint128 assetsUp
  )
```

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.8 Use the internal function `_maxDeposit()` instead of the external function `maxDeposit()`

Severity: Informational

Context: InvestmentManager.sol#L455

Description: The `maxDeposit()` function is used instead of `_maxDeposit()` function.

Recommendation: Consider using `maxDeposit()` is instead of `_maxDeposit()`.

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.6.9 It's unnecessary to use `abi.encodePacked`

Severity: Informational

Context: RestrictionManager.sol#L76

Description: There's no need to use encodePacked . Instead, you could simply use `uint128(hookData.to) >> 64`.

Recommendation: Consider using `uint128(hookData.to) >> 64`.

Centrifuge: Fixed in commit c53a9c0d.

Spearbit: Fixed.

5.6.10 Add balance check in `lockDepositRequest` function

Severity: Informational

Context: CentrifugeRouter.sol#L101

Description: The functions `requestDeposit()` and `requestRedeem()` in `ERC7540Vault` verify the available balance before transferring tokens. However, `lockDepositRequest()` lacks this balance check.

Recommendation: Consider adding a check for user balance in `lockDepositRequest()`.

```
  function lockDepositRequest(address vault, uint256 amount, address controller, address owner)
      public
      payable
      protected
  {
      address initiator = _initiator();
      require(owner == initiator || owner == address(this), "CentrifugeRouter/invalid-owner");
+     require(IERC20(asset).balanceOf(owner) >= amount, "CentrifugeRouter/insufficient-balance");

      lockedRequests[controller][vault] += amount;
      (address asset,) = poolManager.getVaultAsset(vault);
      SafeTransferLib.safeTransferFrom(asset, owner, address(escrow), amount);

      emit LockDepositRequest(vault, controller, owner, initiator, amount);
  }
```

Centrifuge: Acknowledged. The balance checks in `requestDeposit` and `requestRedeem` are not technically necessary, they are only added for better error handling for users. This is not really an issue in `lockDepositRequest`.

Spearbit: Acknowledged.

5.6.11 Revert when signer is zero address in `isValidSignature` function

Severity: Informational

Context: SignatureLib.sol#L24

Description: In case the `signer` is zero address, all invalid signatures are allowed. While `permit()` and `authorizeOperator()` -- the functions that call `isValidSignature()` -- check for this, it's safer to implement the check here as well. This precaution ensures security if the function is used differently in the future.

Recommendation: Consider adding a check that reverts when `signer` is the zero address.

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.6.12 Unused imports

Severity: Informational

Context: Axelar.s.sol#L5, Axelar.s.sol#L7, Deployer.sol#L16

Description: There are unused imports in:

- `Axelar.s.sol` script: `ERC20` and `AxelarForwarder`.
- `Deployer.sol` script: `MockSafe`.

Recommendation: Consider removing the imports

Centrifuge: Fixed in commits c53a9c05 and 46722417.

Spearbit: Fixed.

5.6.13 Replace hardcoded values for id with constants in MessagesLib

Severity: Informational

Context: Gateway.sol#L220-L226, Gateway.sol#L228

Description: The `id` values in `_dispatch()` are equivalent to the values of `enum Call` in `MessagesLib`. Using these enum values instead of hardcoded numbers would prevent errors when new values are added and improve code readability.

Recommendation: Consider replacing the hardcoded numbers for values from `MessagesLib`.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.14 Inconsistent event parameters in `_handle()` function

Severity: Informational

Context: Gateway.sol#L149, Gateway.sol#L158, Gateway.sol#L162, Gateway.sol#L192

Description: All other event emissions in this function use `adapter_`, but this one uses `msg.sender`. Since `_handle()` is called via `handle()`, where `adapter_ == msg.sender`, the result is the same. However, this inconsistency in variable usage may cause confusion.

Recommendation: Consider replacing `msg.sender` with `adapter_`.

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.6.15 Send unused gas from `QUOTA_SLOT` back to the user

Severity: Informational

Context: Gateway.sol#L302

Description: All refunds end up in the gateway, both from the pay/refund parameters and unused gas registered in QUOTA_SLOT. The gas from `QUOTA_SLOT` may be sent back to the original `msg.sender`

Recommendation: Consider sending these refunds back to the original `msg.sender`.

Centrifuge: This is an intentional design decision, to keep complexity low + subsidize some transactions (through `shouldRefuel`).

Spearbit: Acknowledged.

5.6.16 Functions `onERC20Transfer()` and `onERC20AuthTransfer()` calculate the `selector`

Severity: Informational

Context: IHook.sol#L19, IHook.sol#L26, RestrictionManager.sol#L46, RestrictionManager.sol#L57

Description: The functions `onERC20Transfer()` and `onERC20AuthTransfer()` calculate the `selector`, which is error prone. The `selector`can also be retrieved via Solidity.

The comment for `onERC20AuthTransfer()` in `IHook` is most likely incorrect.

Recommendation: Consider changing the code to:

```
  function onERC20Transfer(/*...*/) /*...*/ {
      // ...
-     return bytes4(keccak256("onERC20Transfer(/*...*/)"));
+     return RestrictionManager.onERC20Transfer.selector;
  }
  function onERC20AuthTransfer(/*...*/) /*...*/ {
-     return bytes4(keccak256("onERC20AuthTransfer(/*...*/)"));
+     return RestrictionManager.onERC20AuthTransfer.selector;
  }
```

Double check the comments in `IHook` and also adapt them accordingly.

Centrifuge: Fixed in commit c53a9c05 and PR 61.

The `IHook` documentation stays in the `keccak256()` format.

Spearbit: Fixed.

5.6.17 Initializing with `0` isn't necessary if the variable is also defined in the for `loop`

Severity: Informational

Context: CastLib.sol#L26, ERC7540VaultFactory.sol#L49, Gateway.sol#L212, Gateway.sol#L80, TrancheFactory.sol#L49

Description: Some `for` loops define and then initialize the loop variable with `0`, while others don't, which is inconsistent.

Initializing with `0` isn't necessary if the variable is also defined in the for `loop`.

Recommendation: Consider removing the initialization with `0`.

Centrifuge: Fixed in commit ff06937a.

Spearbit: Fixed.

5.6.18 `shouldRefuel()` case in `send()` can be optimized

Severity: Informational

Context: Gateway.sol#L318

Description: The `currentAdapter` returns any excess gas to `address(this)`. If this amount would be larger than `0`, then the next `adapter` could use this.

Recommendation: Consider changing the code to:

```
- uint256 tank = address(this).balance;
  for (uint256 i; i < numAdapters; i++) {
      // ...
      uint256 consumed = currentAdapter.estimate(/*...*/);
-     if (consumed <= tank) {
+         if (consumed <= address(this).balance)
-         tank -= consumed;
          currentAdapter.pay{value: consumed}(payload, address(this));
      }
      currentAdapter.send(payload);
  }
```

Centrifuge: Fixed in commit 6cdb54af.

Spearbit: Fixed.

5.6.19 Different order for `TYPEHASH` parameters

Severity: Informational

Context: ERC20.sol#L33-L34, ERC7540Vault.sol#L30-L31

Description: The `AUTHORIZE_OPERATOR_TYPEHASH` for `authorizeOperator()` has `deadline` first and then `nonce`. The `PERMIT_TYPEHASH` for ERC20 `permit()` has `nonce` first and then `deadline`.

The standard eip-712 doesn't specify an order so both orders should be valid.

Recommendation: For consistency consider putting `nonce` before `deadline`.

Centrifuge: Fixed in commit ff06937a.

Spearbit: Fixed.

5.6.20 Variable name `manager` is confusing

Severity: Informational

Context: ERC7540Vault.sol#L53

Description: When reading the code of `ERC7540Vault` it is not obvious that `manager` refers to `InvestmentManager` and not to `PoolManager`.

Recommendation: Suggestion: rename `manager` to something like `investmentManager`.

```
- IInvestmentManager public manager;
+ IInvestmentManager public investmentManager;
```

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.21 Comment about `non-transferable` is not clear

Severity: Informational

Context: ERC7540Vault.sol#L27

Description: The comment about `non-transferable` is not clear. More information about the background can be found in EIP-7540.

Recommendation: Consider changing the comment to something like:

```
- /// @dev Requests for Centrifuge pool are non-transferable and all have ID = 0
+ /// @dev Requests for Centrifuge pool all have ID = 0 because no support for transferable requests
↪  will be implemented.
```

And perhaps add a link to the relevant sections of `EIP-7540`.

Centrifuge: Fixed in commit 0c4d52dc.

Spearbit: Fixed.

5.6.22 `BytesLib` can be replaced with pure Solidity

Severity: Informational

Context: BytesLib.sol#L9

Description: The library `BytesLib` is used, which uses a lot of assembly. Most of the functionality can be replaced with pure Solidity which improves readability and maintainability.

If the `bytes` array is `calldata`, then the Solidity slices `[a..b]` can be used, see array-slices.

Most of the use of these functions indeed use `calldata`.

Additionally the EVM now has `mcopy`, so maybe a better implementation can be made for `slice`, see EIP-5656.

Proof of concept: This proof of concept, that can be run in Remix, shows the approach:

```
// SPDX-License-Identifier: none
pragma solidity 0.8.25;
import "hardhat/console.sol";

contract convert {
    function handle(bytes calldata message) public pure {
        console.log(uint8(bytes1(message[0:1])));
        console.log(uint16(bytes2(message[0:2])));
        console.log(uint64(bytes8(message[0:8])));
        console.log(uint128(bytes16(message[0:16])));
        console.log(uint256(bytes32(message[0:32])));
        console.logAddress(address(bytes20(message[0:20])));
        console.log(string(message[0:128])); // fixed lenght string
        console.log(string(message[128:]));   // remaining data used as string
    }
    function test() public view {
        bytes memory m = "12345678901234567890123456789012345678901234567890123456789012345678901234567
↪  890123456789012345678901234567890123456789012345678901234567890";
        convert(this).handle(m); // external call to get calldata
    }
}
```

Recommendation: Consider replacing the library `BytesLib` with pure Solidity.

Centrifuge: Acknowledged. This requires changing the functions in `BytesLib` to external, which makes it a linked library, which various tools don't support well.

Spearbit: Acknowledged.

5.6.23 Deadlines

Severity: Informational

Context: Global scope

Description: Currently there is no expiration on deposit orders which increases the centralization risk and may lead to long wait times for users. The complex nature of the system involves many cross-chain and offchain components, when any of these fail it could also affect the time user funds are held in escrow and in some extreme cases may result in loss of funds.

Implementing an expiration system would not only reduce the centralization risk but also may increase adoption. Additionally, by giving a user the ability to claim tokens related to expired requests, it reduces overall gas and transaction costs incurred by the protocol.

Recommendation: Consider implementing an expiration system.

Centrifuge: There is no deadline mechanism for pools on Centrifuge chain. A deposit / redemption request is submitted and stays open until fulfilled, or cancelled. Having a deadline implementation on the liquidity pool end

would not add any benefit, since a cross-chain message still needs to be sent to cancel the request after the deadline expires. It can even lead to more unnecessary cross-chain messages, as requests might already have been fulfilled & the state has simply not been updated yet on evm side.

Also please note that it is part of the asynchronous tokenized vault implementation standard that the share price is purely informational - it's simply a price snapshot that comes with a timestamp. Due to the asynchronous character of the system, there can not be any price guarantee for deposits / redeem requests. In addition, for Centrifuge pools each user has their own custom share price depending on the epoch their order has been fulfilled. The price can also be the result based on order execution from multiple epochs.

The `maxMint/maxDeposit/maxWithdraw/maxRedeem` values always show the exact share/asset amounts that users will receive based on their own exact share price.

Spearbit: Acknowledged.

5.6.24 `triggerRedeemRequest` risks

Severity: Informational

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

Description: `triggerRedeemRequest` is only intended to be called in emergencies when directed to do so by jurisdictional authorities.

It increases the centralization risk and accidental use of this function by an admin could create user dissatisifaction.

Furthermore, it is not clear if a failed DAO vote would prevent this function from being called which may violate the directions of jurisdictional authorities and put the protocol at risk.

Recommendation: Consider carefully the mechanism used to call this function and document associated risks clearly.

Centrifuge: This is an admin only method for issuers of pools, it does not involve DAO governance.

Spearbit: Acknowledged.

5.6.25 Slippage guard

Severity: Informational

Context: Global scope

Description: Due to the asynchronous nature of the Centrifuge system, it may be worth considering the addition of a slippage guard mechanism to the workflow. There could be a significant change in the price of the RWA during the processing of the off-chain components which would result in a price much different than a user was expecting. Even when Centrifuge was being perfectly fair, having a slippage guard in place can help remove the perception of unfair dealing.

Slippage protection reduces the centralization risk of the protocol and may help increase adoption.

Recommendation: In future development, consider the value of a slippage guard or some sort of similar mechanism that would protect users in the case of a price volatility. As this type of protection is universally enjoyed throughout tradfi and defi, it may be worth considering for future versions of the ERC-7540 standard.

Centrifuge: Pools on Centrifuge chain don't support min expected assets or any kind of slippage protection. It wouldn't work with the nature of the system, since order requests can be fulfilled across multiple epochs.

Also the slippage should not be an issue for our use-case. In our case users deposit funds and receive the according amount of shares once their order gets fulfilled based on the NAV (net asset value) of the pool during the epoch execution. Users with order fulfillments during the same epoch all get the same price and can not impact each other.

Our users are aware that the price gets determined during epoch execution.

Spearbit: Acknowledged.

5.6.26 Gateway design

Severity: Informational

Context: Global scope

Description: Several design decisions around the Gateway contract were made in consideration of future changes to the system configuration. For example, initially there will only be one adapter but the system is built for the possibility of using multiple adapters. The current implementation includes logic for processing multiple adapters in addition to "*shortcuts*" added in for the intended initial rollout of one single adapter.

```
if (adapter.quorum == 1 && !isMessageProof) {
```

Storage for an array of adapters has been added and there is also an entire quorum feature built out in an attempt to support the eventuality of multiple adapters. However, due to the plans to initially require 100% quorum on all adapters, the quorum logic is disabled through hardcoded logic. The current code includes references to system features (such as quorum):

```
if (state.votes.countNonZeroValues() >= adapter.quorum) {
```

But this type of logic is technically not feasible due to a hardcoded contraint which ensures the quorum will always be 100%:

```
uint8 quorum_ = uint8(addresses.length);
```

When possible over-engineering should be avoided as it can result in unneeded complexity and security vulnerabilities such as the issue "Updating set of active adapters does not always clear votes". There also may be additional vulnerabilities that were not uncovered during this review that stem from the implementation of features not currently enabled due as a result of hardcoded logic.

It may be worth considering using a simpler implementation now that addresses the project's needs. In the future, if the plan features are decided to be enabled, the contract will have to be updated to remove hardcoded logic anyways and can implement the additional complexity required at that time.

Recommendation: Reconsider the use of concepts such as quorum, multiple adapters, sessionId, message proof and others that are not currently used by the system and implement a simple Gateway contract that addresses the current needs.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.27 `setOperator()` incomplete documentation

Severity: Informational

Context: IERC7540.sol#L17-L23

Description: `IERC7540Operator` doesn't document the return variable of `setOperator()`.

Recommendation: Consider expanding the documentation.

Centrifuge: Fixed in commit 6d8d7038.

Spearbit: Fixed.

5.6.28 Multicall could use `_initiator()`

Severity: Informational

Context: CentrifugeRouter.sol#L317, CentrifugeRouter.sol#L46

Description: The function `Multicall()` directly uses `INITIATOR_SLOT.tloadAddress()`, while other functions use `_initiator()` (for example `modifier protected()`).

Concentrating all access via the same functions improves consistency, readability and maintainability.

Recommendation: Consider also using `_initiator()` in `Multicall()`. Also consider wrapping `INITIATOR_SLOT.tstore(/*...*/)` in a function.

Centrifuge: Fixed in commit 8ff2127 by removing `_initiator()`.

Spearbit: Fixed.

5.6.29 Unnecessary typecasts in concat

Severity: Informational

Context: BitmapLib.sol#L35-L36

Description: The variables `left` and `right` are already `uint128` so a typecast to `uint128` isn't necessary. Readability can be improved by removing these.

Recommendation: Consider changing the code to:

```
  function concat(uint128 left, uint128 right) internal pure returns (uint256) {
-     return uint256(uint128(left)) << 128 | uint128(right);
+     return uint256(left) << 128 | right;
  }
```

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.30 Some Solidity files have a difference licence

Severity: Informational

Context: Root.sol#L1

Description: Most files have the licence `AGPL-3.0-only` with the exception of:

```
interfaces\token\IHook.sol 1:// SPDX-License-Identifier: MIT
interfaces\IERC20.sol      1:// SPDX-License-Identifier: MIT
```

Recommendation: Doublecheck the licenses.

Centrifuge: Fixed in commit 547939e9.

Spearbit: Fixed.

5.6.31 Some functions use `revert()` without error message

Severity: Informational

Context: ERC7540Vault.sol#L374-L375, ERC7540Vault.sol#L379-L380, ERC7540Vault.sol#L384-L385, ERC7540Vault.sol#L389-L390

Description: Some functions use `revert()` without an error message. This might make troubleshooing errors more difficult.

Recommendation: Consider adding a (custom) error message.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.32 String based errors used

Severity: Informational

Context: Root.sol#L33

Description: The use of custom errors can save gas, allow for custom parameters and makes it easier for integrating projects to detect the errors.

Recommendation: Consider custom errors.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.33 Hardcoded values used in `Auth.sol` and `Root.sol`

Severity: Informational

Context: Auth.sol#L14, Auth.sol#L20, Auth.sol#L26, Root.sol#L57, Root.sol#L63, Root.sol#L69

Description: `Auth.sol` and `Root.sol` use values `0` and `1`, which are not obvious.

Recommendation: Consider usings constants for the values `0` and `1`.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.34 Events are emitted when state hasn't changed

Severity: Informational

Context: Auth.sol#L21

Description: Events are emitted when state has not been changed by the following functions:

- `rely`
- `deny`
- `endorse`
- `veto`
- `pause`
- `setEndorsedOperator`
- `setOperator`
- `setOperator`
- `allowAsset`

- `disallowAsset`

Recommendation: Restrict events or revert when state is not changed.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.35 Use of variable name `tranches` in `Gateway` is confusing

Severity: Informational

Context: Gateway.sol#L349

Description: The use of the variable name `tranches` in `Gateway` is confusing. These `tranches` are unrelated to `Tranche.sol/tranchetoken` which could lead to confusion.

Recommendation: Consider using a different name.

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.36 `updateRestriction()` could try to call non existing hook

Severity: Informational

Context: PoolManager.sol#L270-L274

Description: Function `updateRestriction()` doesn't check `hook != address(0)`. If the hook would be `0` then the call would revert without a clear error message.

Recommendation: Consider detecting this situation and reverting with a clear error message.

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.37 Rigorous reentrancy protection in an adapter could block `triggerRedeemRequest`

Severity: Informational

Context: InvestmentManager.sol#L354

Description: If an adapter has a rigorous reentrancy protection, it might not be possible to send a message from an `handle()`, because it has to reenter in the adapter.

Recommendation: Double check the adapters you want to integrate. If necessary create a workaround.

Centrifuge: Acknowledged. All adapters are intended to be developed by Centrifuge, specifically for Liquidity Pools.

Spearbit: Acknowledged.

5.6.38 No vault level emit for `triggerRedeemRequest()`

Severity: Informational

Context: ERC7540Vault.sol#L394-L408, InvestmentManager.sol#L366

Description: There is no `emit` on the `vault` level for `triggerRedeemRequest()`.

A dapp reading the event might not be able to interpret the `poolId` / `trancheId` from:

```
emit TriggerRedeemRequest(poolId, trancheId, user, poolManager.idToAsset(assetId), shares);
```

Recommendation: Consider emitting an event that references the `vault`. This might requires a callback to ERC7540Vault, comperable to the `on...()` callbacks.

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.39 Lacking function parameter documentation in `IInvestmentManager`

Severity: Informational

Context: IInvestmentManager.sol#L124-L137, InvestmentManager.sol#L300

Description: `IInvestmentManager` doesn't fully document the parameters of all the functions. This is especially important for `fulfillCancelDepositRequest()` because it has different parameters than the other functions.

Recommendation: Consider documenting the parameters of all the functions.

Centrifuge: Fixed in commit 40498fb3.

Spearbit: Fixed.

5.6.40 Could prevent transfer of 0 assets

Severity: Informational

Context: InvestmentManager.sol#L540

Description: The functions `claimCancelDepositRequest()` and `claimCancelRedeemRequest()` don't check if the amount is zero before transferring the tokens.

Recommendation: It's recommended to check if `assets == 0` before transferring.

Centrifuge: Fixed in commit ed0cae6e.

Spearbit: Fixed.

5.6.41 `transferFrom` vs. `safeTransferFrom`

Severity: Informational

Context: ERC7540Vault.sol#L154, InvestmentManager.sol#L486, InvestmentManager.sol#L553

Description: It's a best practice to use `safeTransferFrom` instead of `transferFrom` . The functions `ERC7540Vault.requestRedeem()` , `InvestimentManager._processDeposit()`, `InvestimentManager.claimCancelRedeemRequest()`.

Recommendation: It's recommended to use `safeTransferFrom` .

Centrifuge: Since these are all only interactions with tranche tokens that are always our own ERC20 implementation, there is no need for safe transfers.

Spearbit: The `transferFrom` functions are used only with tranche tokens, so there is no impact. Acknowledged.

5.6.42 Add sanity checks in `PoolManager`

Severity: Informational

Context: PoolManager.sol#L172, PoolManager.sol#L255, PoolManager.sol#L285-L307, PoolManager.sol#L285, PoolManager.sol#L366

Description: Sanity checks are inconsistently applied throughout the various functions called by `handle()` in the PoolManager contract.

It is difficult to predict all of the potential side-effects for all of the edge cases the checks could prevent. For example, if the value of a key were set to the zero address, this may result in the ability to set the zero address as a legitimate value in another mapping which may create a security vulnerability.

The fact that the calls to handle are a result of multiple messages between chains which trigger additional off-chain components, the chance of these type of issues is increased.

Recommendation: Consider adding checks for zero address and zero id in cases where storage is being updated or other security risks may arise.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.43 Comment in `fulfillDepositRequest()` is incorrect

Severity: Informational

Context: InvestmentManager.sol#L239, InvestmentManager.sol#L257

Description: A comment in function`fulfillDepositRequest()` is incorrect.

Recommendation: Consider changing the comment to:

```
- // Mint to escrow. Recipient can claim by calling withdraw / redeem
+ // Mint to escrow. Recipient can claim by calling deposit / mint
```

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.44 `Deposit()` can't be done with the exact same amount as `DepositRequest()`

Severity: Informational

Context: InvestmentManager.sol#L251

Description: After `requestDeposit()` you can't use the exact amount of assets to call `deposit()`. This is due to rounding errors in both `ERC7540Vault` and the Centrifuge chain.

Also `deposit(maxDeposit(controller), receiver)` leaves some shares behind due to rounding errors.

Recommendation: Document that users should use the following approach to get the maximum number of shares. Preferably use:

```
mint(maxMint(controller), receiver);
```

Alternatively use:

```
deposit(maxDeposit(controller), receiver);
```

Centrifuge: Fixed in commit c53a9c05.

Spearbit: Fixed.

5.6.45 Important that `MAX_DECIMALS <= PRICE_DECIMALS`

Severity: Informational

Context: InvestmentManager.sol#L27

Description: It is important that `MAX_DECIMALS` is always lower than or equal to `PRICE_DECIMALS`.

Recommendation: Document this with a code comment.

Centrifuge: Acknowledged.

Spearbit: Acknowledged.

5.6.46 Usage of hardcoded value `8`

Severity: Informational

Context: ArrayLib.sol#L12, ArrayLib.sol#L25, ArrayLib.sol#L6, Gateway.sol#L23, Gateway.sol#L367, IGateway.sol#L129, IGateway.sol#L22

Description: The `ArrayLib` library only supports a `MAX_ADAPTER_COUNT` of exactly `8`. If this constant changes, the library will break. To prevent this, and increase maintainability the hardcoded number `8` in the functions should be replaced with `MAX_ADAPTER_COUNT`.

Recommendation: It's recommended to change the hardcoded number to `MAX_ADAPTER_COUNT` in all the linked cases.

Centrifuge: ArrayLib is a generic library, that implements this method for fixed length arrays. Partly fixed in commit 547939e9.

Spearbit: Fixed, except for `ArrayLib`.

5.6.47 Could use `safeTransferETH()`

Severity: Informational

Context: Gateway.sol#L120

Description: Is a best practice to use `safeTransferETH()` instead of using `transfer()`.

Recommendation: It's recommended to use the `safeTransferETH()`.

Centrifuge: Fixed in commit 304789aa.

Spearbit: Fixed.

5.6.48 `MAX_ADAPTER_COUNT` defined twice

Severity: Informational

Context: Gateway.sol#L29, IGateway.sol#L4

Description: The `MAX_ADAPTER_COUNT` constant is defined in both `Gateway` and in `IGateway`.

Recommendation: It's recommended to remove one of the duplicate declarations.

Centrifuge: Acknowledged. We keep it like this because we want to expose this as a `public` variable in the contract (hence it needs to be defined in the `Gateway` contract), and we don't want to import from a src file in the interface file.

Spearbit: Acknowledged.

5.6.49 No sanity checks on message `length`

Severity: Informational

Context: Gateway.sol#L209

Description: The code of `_dispatch()` to handle batch messages doesn't check if the length value, obtained from `message.toUint16(offset)`, matches the actual length of the message. A check can be added as a safety precaution. Without it, the code might encounter an out-of-bounds error when accessing `message[]`.

Recommendation: It's recommended to check that `length <= message.length-2`.

Centrifuge: Fixed in commit ba79622f.

Spearbit: Fixed.

5.6.50 Unclear when to use `lockDepositRequest()`

Severity: Informational

Context: CentrifugeRouter.sol#L91

Description: The function `lockDepositRequest()` is meant for users that would like to interact with the protocol but don't have permissions yet. However there is not an easy way to figure out the need to call this function or directly use `requestDeposit()`.

Recommendation: Consider implementing a `view` function to determine if `lockDepositRequest()` is necessary based on the restriction manager.

Centrifuge: Fixed in commit 75827538.

Spearbit: Fixed.

5.6.51 `deployVault()` could be called immediately after `removeVault()`

Severity: Informational

Context: PoolManager.sol#L399

Description: The `deployVault()` is a permissionless function and could be called immediately after `removeVault()`. This will make it impossible to completely remove a vault.

Recommendation: It's recommended to remove the vault and block new deployments to the same asset.

Centrifuge: This method is primarily meant to be used for migrating vault contracts. In that case,`removeVault()` would be called in a spell (migration contract), that would do something like:

```
poolManager.removeVault(/*...*/);
poolManager.file("vaultFactory", newVaultFactory);
poolManager.deployVault(/*...*/);
```

Spearbit: Acknowledged.

5.6.52 It is not obvious that there are two `escrow`s

Severity: Informational

Context: CentrifugeRouter.sol#L29, Deployer.sol#L51-L52, ERC7540Vault.sol#L50, InvestmentManager.sol#L30, PoolManager.sol#L41

Description: The variable `escrow` is used by most contracts. It is not obvious the `CentrifugeRouter` uses one instance and all the other contracts use another instance. It can only be seen in the `deployer` script.

Recommendation: Consider changing `escrow` to `routerEscrow` in `CentrifugeRouter`.

Centrifuge: We feel it is better to keep it `escrow`. From the perspective of the `CentrifugeRouter` contract it is just an `escrow`.

Spearbit: Acknowledged.

5.6.53 `RequestId` value of `0` is not obvious

Severity: Informational

Context: CentrifugeRouter.sol#L152, CentrifugeRouter.sol#L170, CentrifugeRouter.sol#L184, CentrifugeRouter.sol#L195, CentrifugeRouter.sol#L220, CentrifugeRouter.sol#L230, CentrifugeRouter.sol#L87, ERC7540Vault.sol#L28

Description: Several functions are called with a value of `0`, where it is not obvious what this value means.

> **Note:** ERC7540Vault does have a constant for this value:

```
uint256 private constant REQUEST_ID = 0;
```

Recommendation: Consider using a constant instead of `0` for the `requestId`.

Centrifuge: Fixed in commit 0c4d52dc.

Spearbit: Fixed.

5.6.54 Value of `wards[]` is set directly

Severity: Informational

Context: Adapter.sol#L54, CentrifugeRouter.sol#L41, ERC20.sol#L41, ERC7540Vault.sol#L92, ERC7540VaultFactory.sol#L33, GasService.sol#L40, Gateway.sol#L51, InvestmentManager.sol#L41, PoolManager.sol#L60

Description: The values for `wards` is set directly in the code, which is more error phrone and makes maintenance of the code more difficult.

Recommendation: It's recommended to use the `rely(msg.sender)` instead of `wards[msg.sender] = 1`.

> **Note:** `rely` itself requires authorization, so it might be necessary to add an internal version of `rely()` to be able to do this.

Centrifuge: Fixed in commits 1af93750 and 0ca68a01.

Spearbit: Fixed.

5.6.55 `denyVault()` doesn't undo all actions of `newVault()`

Severity: Informational

Context: ERC7540VaultFactory.sol#L58

Description: The `denyVault()` function doesn't undo the `vault.rely(wards_[i])`.

Recommendation: It's recommended to `vault.deny()` all wards added in `newVault()` function.

Centrifuge: Acknowledged. This is an admin only method and any calls to `denyVault()` will be through a `spell` (migration contract) that will also remove any other `wards`.

Spearbit: Acknowledged.

5.6.56 `deployVault()` has fewer checks than `removeVault()`

Severity: Informational

Context: PoolManager.sol#L366

Description: The `removeVault()` does have an additional check:

```
require(pools[poolId].createdAt != 0, "PoolManager/pool-does-not-exist");
```

The extra check is not necessary because the check for `tranche.token != address(0)` implicitly also checks this, but it is not consistent

Recommendation: It's recommended to remove the additional check.

Centrifuge: Fixed in commit ed0cae6e.

Spearbit: Fixed.

5.6.57 Any tokens left in `CentrifugeRouter` can be used by anyone

Severity: Informational

Context: CentrifugeRouter.sol#L303

Description: Any tokens left in `CentrifugeRouter` can be used by anyone, for example via the `_approveMax()` where you can approve for any random token.

According to the project: This is a design decision: a user needs to ensure no tokens are remaining in the Centrifuge Router at the end of the multicall.

This might not be obvious to users of the `CentrifugeRouter`.

Recommendation: Highlight this fact in the documentation.

Centrifuge: This is a design decision: a user needs to ensure no tokens are remaining in the Centrifuge Router at the end of the multicall. This is exactly how it also works for Morpho bundlers (which is the main inspiration): ERC20WrapperBundler.sol#L37.

Fixed in commit ff06937a.

Spearbit: Fixed.

5.6.58 Variable name `balances` doesn't cover all use cases

Severity: Informational

Context: ERC20.sol#L22, Tranche.sol#L74-L76

Description: The `balances` variable includes both balance and hook data, but its name doesn't accurately represent all the information it contains.

Recommendation: The recommendation is to rename `balances` to `balancesAndHookData` to make it obvious extra data is stored here. USDC does something similar by calling it balanceAndBlacklistStates, see FiatTokenV1.sol#L41-L44.

Centrifuge: Fixed in commit 7a034b96.

Spearbit: Fixed.

5.6.59 Unused definition for `newTransferProxy()` with two parameters

Severity: Informational

Context: TransferProxyFactory.sol#L23

Description: There is no implementation for `newTransferProxy()` with two parameters.

> **Note:** there is one with one parameter.

`TransferProxyFactoryLike` is also never inherited from.

Recommendation: Doublecheck the need for `newTransferProxy()` with two parameters and remove it if not needed. Consider inheriting from `TransferProxyFactoryLike`.

Centrifuge: Fixed in commit b69abb0f.

Spearbit: Fixed.

5.6.60 `Like` interfaces don't start with an `I`

Severity: Informational

Context: Adapter.sol#L21, Adapter.sol#L9, ERC7540VaultFactory.sol#L20, ERC7540VaultFactory.sol#L7, Guardian.sol#L8, TrancheFactory.sol#L7, TransferProxyFactory.sol#L22

Description: All the `...Like` interfaces are not prefixed with an `I`. Usually a `I` prefix is used to indicate interfaces.

Recommendation: Prefix all the `...Like` interfaces with an `I`.

Centrifuge: Fixed in commit b69abb0f.

Spearbit: Fixed.