CANTINA

OT Perimeter
Security Review

Cantina Managed review by:

Akshay Srivastav, Security Researcher
Ladboy233, Security Researcher

March 14, 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

OpenTrade powers safe, compliant, and scalable stablecoin yield products designed for the next generation of financial services & markets.

From Feb 21st to Mar 1st the Cantina team conducted a review of ot-perimeter on commit hash 6fcec5c6. The team identified a total of **46** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	7	7	0
Medium Risk	11	11	0
Low Risk	18	18	0
Gas Optimizations	1	1	0
Informational	9	7	2
Total	**46**	**44**	**2**

3 Findings

3.1 High Risk

3.1.1 Incorrect debit cap check in `addAdjustmentAmount` leads to DoS of debit adjustment

Severity: High Risk

Context: PoolDynamic.sol#L554-L556

Description: The debit cap check implemented in `addAdjustmentAmount` is incorrect.

```
if (debit + _accountings.totalCurrentDebits > assetBalanceOf(lender)) {
  revert DebitGreaterThanAssets();
}
```

As it can be seen, the function includes `_accountings.totalCurrentDebits` into individual user debit limit which is incorrect.

Scenario:

1. User has 100 pool tokens (== 100 asset tokens).
2. Outstanding `totalCurrentDebits` of pool is 1000.
3. Admin wants to increase the debit of user by 10.
4. `addAdjustmentAmount(user, 10, 0)` call reverts due to incorrect check.

Recommendation: Consider adding these changes:

```
- if (debit + _accountings.totalCurrentDebits > assetBalanceOf(lender)) {
+ if (debit > assetBalanceOf(lender)) {
      revert DebitGreaterThanAssets();
  }
```

OpenTrade: Fixed in commit 57f3e7ae. With the change to do a burn to add debit adjustment instead of tracking the debits this is no longer an issue:

```
if (debit > 0) {
    if (debit > assetBalanceOf(lender)) {
      revert DebitGreaterThanAssets();
    }
}
```

Cantina Managed: Fix verified.

3.1.2 Debit amounts can be bypassed by transferring pool tokens to another lender wallet

Severity: High Risk

Context: PoolDynamic.sol#L289, PoolDynamic.sol#L690

Description: The current implementation of `PoolDynamic` contract allows transferring of pool tokens even when a lender has outstanding debit. Due to this the debit amounts can be avoided by transferring pool tokens to another lender wallet.

Scenario:

1. Lender has 100 shares (== 100 assets) and 100 debits. His net effective balance is 0 so he is not allowed any redemption.
2. But lender can transfer his 100 shares (pool tokens) to another whitelisted wallet.
3. Since the new wallet will have no debits, the lender can request redemption of his 100 shares and get 100 USDC.

Due to this issue a lender can get away with more asset tokens that originally intended. Leading to loss of funds to the protocol.

Recommendation: When a pool token transfer occurs (`_beforeTokenTransfer`), consider validating that the lender is not transferring his `assetBalance - debitAmount` tokens.

OpenTrade: Fixed in commit 27f52ccf. This is no longer an issue because debits are implemented as a burn.

Cantina Managed: Fix verified.

3.1.3 Debit amounts can be charged to lender multiple times

Severity: High Risk

Context: PoolDynamic.sol#L291

Description: In the `requestRedeem` function, after adjusting the asset value with debits, the function does not reset the debit value of user. Due to this the same debit amount is subtracted from all subsequent redemptions of user.

Scenario:

1. User has 100 shares (==100 assets) and 10 debits.
2. User requests redemption of 50 shares. His asset withdrawal amount is calculated as `50 - 10 = 40`.
3. After a while user requests redemption of his remaining 50 shares. His asset withdrawal amount is again calculated as `50 - 10 = 40`.
4. User had 10 debits but he could only withdraw 80, leading to a loss of 10 asset tokens.

Recommendation: Consider resetting the `debitAmount` of user once it has been accounted in user's asset balance. Or, consider implementing a simpler debit/credit system in which the admin can directly mint and burn pool tokens of users.

OpenTrade: Fixed in commit 27f52ccf. This is no longer an issue because debits are implemented as a burn.

Cantina Managed: Fix verified.

3.1.4 `setfeeCollectorAddress` and `setBorrowerWalletAddr` in `PoolControllerDynamic.sol` lack access control

Severity: High Risk

Context: PoolControllerDynamic.sol#L131

Description: Access control are missing for function `setfeeCollectorAddress` and `setBorrowerWalletAddr`. If the borrower wallet and fee collector wallet are different wallet controlled by different entity, the user can swap these two addresses and lender's depost fund may redirected to wrong wallet.

Recommendation: Add the modifier `onlyPoolAdmin` to ensure these two function have access control.

OpenTrade: Fixed in commit 57f3e7ae.

Cantina Managed: Fix verified.

3.1.5 Accepted redemption amounts can be used to create new requests

Severity: High Risk

Context: PoolDynamic.sol#L317-L319, PoolDynamic.sol#L344-L348

Description: As per the `acceptRedemption` function, once a request is accepted, lender's `requestedShares` amount is reduced which increases his `maxRedeemRequest` value. Due to this the same shares can be used to request a new redemption again.

Scenario:

1. Lender has 100 shares.
2. He requests redemption of 100 shares. His `requestedShares` are now 100 and `maxRedeemRequest` is 0.
3. Borrower liquidates his position off-chain and accepts the lender's redemption request.
4. After acceptance the lender's `requestedShares` becomes 0 and `maxRedeemRequest` becomes 100.

5. Now the lender can transfer his 100 shares (pool tokens) to another whitelisted wallet or keep them on the same wallet.
6. Lender creates new redemption request of 100 shares again.
7. Borrower will again liquidate itself to fulfil the request.
8. Step 3 to 6 are executed again.

While in this scenario the lender cannot get hold of more than 100 USDC, he can surely cause unnecessary off-chain liquidations for borrower by creating "fake" redemption requests.

Recommendation: Consider adding these changes:

```
  function maxRedeemRequest(address owner) public view returns (uint256 maxShares) {
-     maxShares = balanceOf(owner) - _lenderData[owner].requestedShares;
+     maxShares = balanceOf(owner) - _lenderData[owner].requestedShares - _lenderData[owner].acceptedShares;
  }
```

An alternative design could be to simply burn the pool tokens of lender when they initiate a redemption request.

OpenTrade: Fixed in commit 57f3e7ae. Did not want to do the burn at request because it would be a different behavior than existing pools.

Cantina Managed: Fix verified.

3.1.6 DoS of `repayRedemption` by increasing the `activeWithdrawKeys` array

Severity: High Risk

Context: PoolDynamic.sol#L389-L397

Description: The `repayRedemption` function iterates over the `activeWithdrawKeys` array to find and remove an `uuid`.

The iteration over all `activeWithdrawKeys` can be misused by a malicious lender. Any lender can create large number of redemption requests of small amounts (or `0` amounts) which will increase the length of `activeWithdrawKeys` array. After that when admin tries to repay legitimate requests of other users, the transaction can hit the block gas limit and revert. Leading of permanent DoS of `repayRedemption` feature for all lenders.

Recommendation: Consider removing the need to iterate over an array. If that's not possible then use OpenZeppelin's `EnumerableSet` library for storing `activeWithdrawKeys`.

OpenTrade: Fixed in commit a029a613. I limited the number of withdraw requests of any lender to 4 to limit this abuse:

```
if (_lenderData[lender].outstandingRequests > 4) {
    revert InvalidRedemptions();
}
```

Cantina Managed: Fix verified.

3.1.7 Removal of a `uuid` from `activeWithdrawKeys` is implemented incorrectly

Severity: High Risk

Context: PoolDynamic.sol#L389-L401

Description: When a redemption request is repaid the `repayRedemption` function iterates over the `activeWithdrawKeys` array to find the `uuid`. Once the uuid is found at an index `i` the function tries to swap the uuids at index `i` with `i + 1`, this is done so that after the `for` loop ends the found uuid gets moved to last index, which can then be simply popped from the array.

```
for (uint i = 0; i < activeWithdrawKeys.length; i++) {
    bytes32 key = activeWithdrawKeys[i];
    if (key == uuid) {
        isFound = true;
    }
    if (isFound && i < activeWithdrawKeys.length - 1) {
        activeWithdrawKeys[i] = activeWithdrawKeys[i + 1];
    }
}
if (isFound) {
    delete _activeWithdraws[uuid];
    activeWithdrawKeys.pop();
}
```

However, code is only moving `i + 1` key to `i` and have missed moving `i` to `i + 1`. This leads to double entry of key `i` in the `activeWithdrawKeys`. Also at the end an incorrect `uuid` is popped.

Example:

- `activeWithdrawKeys` has three uuids - `[ 0xa, 0xb, 0xc ]` and admin want to repay `0xa`.
- After first loop iteration the array will become - `[ 0xb, 0xb, 0xc ]`.
- Now since `0xa` is absent from the array, the array will remain same till the `for` loop ends.
- Then at `L400` array will be popped which will incorrectly remove uuid `0xc` from the array.
- At the end of function we will have this array - `[ 0xb, 0xb ]`.

In this case the uuid `0xc` got removed and uuid `0xb` got duplicated.

Recommendation: The current way of removing elements from array is inefficient for smart contracts. Instead, as soon as the key is found swap it with last key of the array, then pop the array.

```
for (uint i = 0; i < activeWithdrawKeys.length; i++) {
    bytes32 key = activeWithdrawKeys[i];
    if (key == uuid) {
        isFound = true;
        activeWithdrawKeys[i] = activeWithdrawKeys[activeWithdrawKeys.length - 1];
        break;
    }
}
if(!found) revert KeyNotFound();
delete _activeWithdraws[uuid];
activeWithdrawKeys.pop();
```

An even better alternative will be remove the `activeWithdrawKeys` array from pool contract.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 `addAdjustmentAmount`: debit adjustment can be prevented by transferring all pool tokens

Severity: Medium Risk

Context: PoolDynamic.sol#L548-L550

Description: As per the `addAdjustmentAmount` function, debit cannot be performed for a lender which has `0` pool token balance. This mechanism can be misused. To escape an upcoming debit adjustment a user can transfer all its pool tokens to its another wallet. This will force the debit adjustment to fail.

Recommendation: Consider removing the `if (balanceOf(lender) == 0) revert` check.

OpenTrade: Fixed in commit 57f3e7ae. Pool Tokens can only be transferred to another lender on the allowed list. We would then do the debit adjustment to that account.

Cantina Managed: Fix verified.

3.2.2 Duplicate `uuid` entries can be pushed in the `activeWithdrawKeys` array

Severity: Medium Risk

Context: PoolDynamic.sol#L299-L310, PoolDynamic.sol#L321

Description: The `requestRedeem` function does not prevent creation of duplicate `uuid` values. Due to this, requesting multiple redemptions of same amount in same block will push duplicated `uuid` entries in the `activeWithdrawKeys` array.

```
bytes32 uuid = keccak256(abi.encodePacked(block.timestamp, lender, shares, assets));
_activeWithdraws[uuid] = IPoolWithdrawDynamic({
    // ...
});
activeWithdrawKeys.push(uuid);
```

This behaviour can impact the `repayRedemption` withdrawal flow.

Recommendation: Consider reverting when `_activeWithdraws[uuid].uuid != bytes32(0)`. Or if multiple redemptions of same amount in same block need to be supported then a new state variable `uint256 withdrawalCount` can be added which always gets incremented with redemption request creation, it should also be included in `uuid` generation.

```
uint256 public withdrawalCount;

function requestRedeem(/*...*/) {
    // ...
    bytes32 uuid = keccak256(abi.encodePacked(block.timestamp, withdrawalCount, lender, shares, assets));
    withdrawalCount++;
    // ...
}
```

OpenTrade: Fixed in commit 57f3e7ae. Added in function requestRedeemExecute to prevent duplicate withdrawals:

```
if (_activeWithdraws[uuid].lender != address(0)) {
    revert InvalidRedemptions();
}
```

Cantina Managed: Fix verified.

3.2.3 `repayRedemption`: tokens can be burned from an unrelated lender for a `uuid` request

Severity: Medium Risk

Context: PoolDynamic.sol#L334, PoolDynamic.sol#L352

Description: The `acceptRedemption` and `repayRedemption` functions are missing checks to ensure input `lender` parameter is the correctly associated address with the provided `uuid`. Currently an unrelated `lender` can be provided. Due to this pool tokens of an unrelated lender can be burned and accepted/requested shares/assets state will be updated.

Recommendation: Add this check in `acceptRedemption` and `repayRedemption` functions.

```
if (_activeWithdraws[uuid].lender != lender) revert
```

OpenTrade: Fixed in commit 57f3e7ae.

Cantina Managed: Fix verified.

3.2.4 Redemption can be costly because of the unbounded loop in exchange rate computation if the compounding rate is set

Severity: Medium Risk

Context: PoolDynamic.sol#L481-L482

Description: The exchange rate accounting is important because the Pool replies the function `exchangeRate()` to compute the shares worth. However, the code contains an unbounded loop. The number loops grows linearly with the number of days. After 1 year, the loop runs 365 times. After 10 years, the loop

runs 3650 times. Such unbounded loop would cost excessive gas in redemption (User has to spend a lot of gas because of the large for loop).

Recommendation:

```
if (numberOfDays > 0) {
    _exchangeRate = _accountings.exchangeRate;

    // Calculate compounded rate without a loop
    uint256 compoundedRate = _accountings.exchangeRateCompoundingRate**numberOfDays;
    _exchangeRate = _exchangeRate.mul(compoundedRate).div(1e18**numberOfDays);
}
```

OpenTrade: The exchange Rate settting has been refactored in order to cover the issues you raised and to fit the contract into 24k. It is commits a8b1c251 and 84a676b1, followed by additional contract size reduction in commit 1ffb20a4.

Now there is an `exchangeRateType` set when the pool is created and never changes. The computed Exchange rate is calculated based on the exchange rate type. The exchange Rate validation is implemented in the `Controller` to save space.

Note the exchange rate for compounding is limited to compounding 7 days. It is intended to reset the exchange rate every business day so we avoid long loops.

The `exchangeRateCompounding` is correct in that the expected and validated value will is 1 + daily interest multiplier, while the linear `exchangeChange` rate is the amount the value will increase. Validation is set to ensure at most a 10% change in value per day, (I could potentially lower it to 1% per day).

Cantina Managed: Fix verified.

3.2.5 Pool tokens can be transferred when pool contract is paused

Severity: Medium Risk

Context: PoolDynamic.sol#L690

Description: The `PoolDynamic._beforeTokenTransfer` hook is missing `onlyNotPaused` modifier due to which pool tokens can be transferred when the pool contract is paused. Contracts are generally paused in critical scenario (bug disclosures, active exploits, upgrades, etc...) and allowing token transfers in paused state can lead to unintended outcomes for the protocol and its users.

Recommendation: Consider adding the `onlyNotPaused` modifier to `beforeTokenTransfer` hook.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.2.6 Pool tokens can be transferred when pool contract is not `Active`

Severity: Medium Risk

Context: PoolDynamic.sol#L690

Description: The `PoolDynamic._beforeTokenTransfer` hook is missing `atState(IPoolLifeCycleStateDynamic.Active)` modifier due to which pool tokens can be transferred when the pool contract is not in `Active` state. Most of the pool operations like deposit, request redemption, repay redemption, etc are only allowed when pool's state is `Active`. Pool token transfers should also follow the same principle otherwise unintended outcomes can occur for protocol and its users.

Recommendation: Consider adding the `atState(IPoolLifeCycleStateDynamic.Active)` modifier to `_beforeTokenTransfer` hook.

OpenTrade: Fixed in commit 954c5287.

Cantina Managed: Fix verified.

3.2.7 Non-whitelisted accounts can perform pool token transfer via `transferFrom` function

Severity: Medium Risk

Context: PoolDynamic.sol#L690

Description: The `_beforeTokenTransfer` hook is missing the whitelist check on `msg.sender`.

As pool tokens can be transferred via `transferFrom` function, two scenarios can occur:

1. A non-whitelisted accounts can can request redemption by doing a pool token transfer to pool contract.
2. A non-whitelisted account can perform a pool token transfer from one whitelisted account to another whitelisted account.

- Scenario 1:
  - A lender is whitelisted and mints some pool tokens.
  - Lender gets removed from whitelist so he cannot call `requestRedeem`.
  - Lender transfers the pool tokens to pool contract which initiates the `requestRedeemViaTransfer` flow.
  - A redemption requests gets created for lender.
- Scenario 2:
  - Assume Alice and Bob are two whitelisted users and Charlie is a non-whitelisted user.
  - Alice approves Charlie to transfer its pool tokens.
  - Charlie can then transfer pool tokens of Alice to Bob.

This issue can break the compliance requirement of OpenTrade by allowing pool operations to non-whitelisted users.

Recommendation: Add `onlyPermittedLender` modifier to `_beforeTokenTransfer` hook.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.2.8 `PoolControllerDynamic`: `poolAdmin` and `borrowerManager` addresses cannot be changed

Severity: Medium Risk

Context: PoolControllerDynamic.sol#L29, PoolControllerDynamic.sol#L123-L125

Description: The `PoolControllerDynamic` contract stores `poolAdmin` and `borrowerManager` addresses. These addresses are set at pool initialization.

These addresses are responsible for calling these functions on contracts:

- `borrowerManager` -- `PoolDynamic.acceptRedemption`.
- `poolAdmin`.
- `PoolDynamic` -- `depositFromTransfer`, `depositOffChain`, `changeRedemtionDestination` & `repayRedemption`.
- `PoolControllerDynamic` -- all setter functions.

In case the protocol admins want to change these addresses (due to wallet compromise, governance transfer, etc) then that won't be possible. This leaves the protocol stuck with the original `poolAdmin` and `borrowerManager` addresses set at initialization.

Recommendation: Consider adding function to change `poolAdmin` and `borrowerManager` addresses.

OpenTrade: Fixed in commit b67def02.

Cantina Managed: Fix verified.

3.2.9 `PoolControllerDynamic`: `FIAT_GATEWAY` is set as fee collector address

Severity: Medium Risk

Context: PoolControllerDynamic.sol#L132

Description: The `setfeeCollectorAddress` function incorrectly checks `isFiatGateway` role for the input `_feeCollectorAddress`. Due to this an address with `FEE_COLLECTOR` role cannot be set as fee collector. Instead an address with `FIAT_GATEWAY` role can only be set as the fee collector address.

Recommendation: Consider adding these changes:

```
  function setfeeCollectorAddress(address _feeCollectorAddress) external {
-     if (!IServiceConfigurationV5(serviceConfiguration).isFiatGateway(_feeCollectorAddress)) {
+     if (!IServiceConfigurationV5(serviceConfiguration).isfeeCollector(_feeCollectorAddress)) {
          revert NotfeeCollector();
      }
      _settings.feeCollectorAddress = _feeCollectorAddress;
  }
```

OpenTrade: Fixed in commit ce3f8d26.

Cantina Managed: Fix verified.

3.2.10 Pausing the deposits on pool contract also pauses withdrawals

Severity: Medium Risk

Context: PoolControllerDynamic.sol#L201-L203, PoolDynamic.sol#L278-L283

Description: The deposits on `PoolDynamic` contract can be paused by calling the `PoolControllerDynamic.pauseDeposits` function. From its naming it seems that this is intended to only pause new deposits in the pool. However, the withdrawal functions on `PoolDynamic` contract (`requestRedeem`, `changeRedemptionDestination`, `acceptRedemption` & `repayRedemption`) reverts when pool's state is not `Active`. Due to which withdrawals also gets paused when deposits are paused for a pool.

Recommendation: Consider allowing withdrawals when pool is in `PausedDeposits` state.

OpenTrade: Fixed in commit 3626270e. Disconnected `DepositState` (inactive and active) from `withdrawState` (inactive and active).

Cantina Managed: Fix verified.

3.2.11 `PoolControllerDynamic.reactivateAfterDisrution` cannot be executed

Severity: Medium Risk

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

Description: The commit `6f22a0f` added a new `reactivateAfterDisrution` function. As the function has `atActiveOrClosedState` modifier and also has `state == DisruptionOrDefault` check, these contrasting conditions can never be met. Hence the function can never be executed.

Recommendation: Consider removing the `atActiveOrClosedState` modifier.

OpenTrade: Fixed in commit 23eccfd9.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Pool address can be passed as lender in `addAdjustmentAmount` function

Severity: Low Risk

Context: PoolDynamic.sol#L540-L544, PoolDynamic.sol#L693-L704

Description: In the `addAdjustmentAmount` function, pool address can be passed as `lender` with a non-zero `credit` amount. This can trigger a `_mint` to pool address, which will also trigger the `requestRedeemViaTransfer` logic of `_beforeTokenTransfer` function.

Recommendation: Add a check in `addAdjustmentAmount` to validate that:

```
lender != address(this)`.
```

A check can be added in `_beforeTokenTransfer` function to prevent minting to pool address and stop an unwanted `requestRedeemViaTransfer` flow.

```
    if (to == address(this)) {
+       if (from == address(0)) revert MintingToPool();
        requestRedeemViaTransfer(from, amount);
    } else {
        //...
```

OpenTrade: Fixed in commit c88412ad:

```
if (lender == address(0) || lender == address(this)) {
    revert InvalidAccess();
}
```

Cantina Managed: Fix verified.

3.3.2 Missing whitelist check for lender in `addAdjustmentAmount` function

Severity: Low Risk

Context: PoolDynamic.sol#L540-L544

Description: The `addAdjustmentAmount` function is missing whitelist check for `lender`. Due to this credit can be given and pool tokens can be minted to a non-whitelisted lender.

Scenario:

- Lender gets whitelisted.
- Lender deposits asset and mints some pool token (this is needed to pass `balanceOf` check).
- Lender gets removed from whitelist.
- Now pool tokens can be minted to lender via credit.

Recommendation: Consider adding the whitelist check for lender so that no pool tokens can be minted to non-whitelisted lenders.

OpenTrade: Fixed in commit 57f3e7ae. Added:

```
if (!poolAccessControl.isAllowed(lender)) revert NotLender();
```

Cantina Managed: Fix verified.

3.3.3 Rounding direction in `addAdjustmentAmount` favours lender

Severity: Low Risk

Context: PoolDynamic.sol#L564

Description: The credit case of `addAdjustmentAmount` uses `divideAndRoundUp` function to determine the amount of pool shares that should be minted to a lender. This rounding direction favours the lenders as opposed to the pool. As per the general vault development practices, rounding direction must always favour pool instead of users.

Recommendation: Consider using `convertToShares` to calculated lender's pool shares.

OpenTrade: Fixed in commit 57f3e7ae. Total addAdjustment function is now:

```
function addAdjustmentAmount(
    address lender,
    uint256 debit,
    uint256 credit
) public atState(IPoolLifeCycleStateDynamic.Active) onlyPoolController {
    if (debit > 0 && credit > 0) {
        revert InvalidAccess();
    }
    if (lender == address(0) || lender == address(this)) {
        revert InvalidAccess();
    }

    if (!poolAccessControl.isAllowed(lender)) revert NotLender();
    if (debit > 0) {
        if (debit > assetBalanceOf(lender)) {
            revert DebitGreaterThanAssets();
        }
        uint256 shares = convertToShares(debit);
        _burn(lender, shares);
        emit AccountDebit(lender, debit);
    } else if (credit > 0) {
        uint256 shares = convertToShares(credit);
        _mint(lender, shares);

        emit AccountCredit(lender, credit, shares);
    } else {
        revert InvalidAccess();
    }
}
```

Cantina Managed: Fix verified.

3.3.4 `onActivated` function can be called multiple times

Severity: Low Risk

Context: PoolDynamic.sol#L159-L161

Description: The `onActivated` function of `PoolDynamic` contract can be called multiple times. This can be used to change the `activatedAt` timestamp of pool multiple times.

Recommendation: Consider adding these changes:

```
  function onActivated() external onlyPoolController {
+     if (activatedAt != 0) revert AlreadyActivated();
      activatedAt = block.timestamp;
  }
```

OpenTrade: Fixed in commit 57f3e7ae. Changed to:

```
function onActivated() external onlyPoolController {
    if (activatedAt == 0) {
        activatedAt = block.timestamp;
    }
}
```

Cantina Managed: Fix verified. After the fix the `onActivated` can still be called multiple times but `activatedAt` timestamp will not get updated.

3.3.5 Lenders can create redemption requests of `0` share amount

Severity: Low Risk

Context: PoolDynamic.sol#L278-L280

Description: The `requestRedeem` function does not validate that the input `shares` value is non-zero. Due to this any lender can create unlimited number of redemption requests of `0` share amount.

Recommendation: Consider adding this check in `requestRedeem` function:

```
if (shares == 0) revert ZeroAmount();
```

OpenTrade: Fixed in commit 57f3e7ae. Changed in `redeemRequestExecute`:

```
function requestRedeemExecute(address lender, uint256 shares, bool transferRedeem) internal returns (uint256
↪  assets) {
    assets = convertToAssets(shares);

    if (shares == 0) {
        revert ZeroAmount();
    }
```

Cantina Managed: Fix verified.

3.3.6 Redemption cannot be requested if debit of lender is equal to its asset balance

Severity: Low Risk

Context: PoolDynamic.sol#L289

Description: The `requestRedeem` function reverts when asset balance of user is equal to his debit amount. If a user has 100 shares (= 100 assets) and 100 debits then he cannot request a redemption. His pool token balance will remain in his wallet but those tokens will be unusable. Ideally the function should accept the lender's request by issuing him `0` asset tokens. It should only revert if `convertToAssets(shares) < _lenderData[lender].debitAmount`.

Recommendation: Replace <= with <.

OpenTrade: Fixed in commit 57f3e7ae. This is now in `RequestRedeemExecute` and does not convert to assets first:

```
function requestRedeemExecute(address lender, uint256 shares, bool transferRedeem) internal returns (uint256
↪  assets) {
    assets = convertToAssets(shares);

    if (shares == 0) {
      revert ZeroAmount();
    }
    if (shares > maxRedeemRequest(lender)) {
      revert RedeemExceedsBalance();
    }
```

Cantina Managed: Fix verified.

3.3.7 `changeRedemtionDestination`: missing existence check for `uuid`

Severity: Low Risk

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

Description: Missing existence check for `uuid` in `changeRedemtionDestination` function. Admin can pass `address(0)` as lender and any non-existent `uuid` value, the function will successfully write `fundsDestination` for that `uuid`.

Recommendation: Consider reverting if `_activeWithdraws[uuid].stage == 0`.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.3.8 Dynamic pool can be closed with leftover fund

Severity: Low Risk

Context: PoolControllerDynamic.sol#L216

Description: If the pool is closed, the check below ensures there are no funds left:

```
if (pool.totalAssets() > 0) revert PoolSettingsInvalid();
```

However, the `pool.totalAssets()`is a empty function in PoolDynamic.sol so the pool can be closed with leftover funds:

```
function totalAssets() public view returns (uint256) {}
```

Recommendation: Implement the `pool.totalAssets()` is a empty function method.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.3.9 Unsafe typecasting in `repayRedemption` function

Severity: Low Risk

Context: PoolDynamic.sol#L378

Description: The `repayRedemption` function performs unsafe typecasting of `uint256` values to `int256` values.

```
    int256 adjustment = int256(repayment) + int256(fees) - int256(_activeWithdraws[uuid].assets);
```

Unsafe typecasting can result in integer overflow/underflow.

Recommendation: Consider using `Safecast` library for safe typecasting.

OpenTrade: Fixed in commit 6e8d4df6.

Cantina Managed: Fix verified.

3.3.10 `previewRedeemRequest` & `previewWithdrawRequest` do not account the requested & accepted shares of lender

Severity: Low Risk

Context: PoolDynamic.sol#L267-L276

Description: The `IRequestWithdrawable` interface mentions that:

```
/**
 * @dev Simulate the effects of a redeem request at the current block.
 * Returns the amount of underlying assets that would be requested if this
 * entire redeem request were to be processed at the current block.
 *
 * Note: This is equivalent of EIP-4626 `previewRedeem`
 */
function previewRedeemRequest(uint256 shares) external view returns (uint256 assets);

/**
 * @dev Simulate the effects of a withdrawal request at the current block.
 * Returns the amount of `shares` that would be burned if this entire
 * withdrawal request were to be processed at the current block.
 *
 * Note: This is equivalent of EIP-4626 `previewWithdraw`
 */
function previewWithdrawRequest(uint256 assets) external view returns (uint256 shares);
```

However the `PoolDynamic` contract's `previewRedeemRequest` and `previewWithdrawRequest` functions do not account the already `requestedShares` and `acceptedShares` of lender. Due to this the when lender with requested/accepted shares call the preview functions, the calls will succeed but the actual redemption request call will fail.

Recommendation: Consider accounting the `requestedShares` and `acceptedShares` of lender in the `previewRedeemRequest` and `previewWithdrawRequest` functions.

OpenTrade: Fixed in commit ffddf19d. Added

```
if (shares > maxRedeemRequest(msg.sender)) {
    revert RedeemExceedsBalance();
}
```

to both functions.

Cantina Managed: Fix verified.

3.3.11 Multiple functions of `PoolDynamic` contract are empty and always return `0`

Severity: Low Risk

Context: PoolDynamic.sol#L260-L262, PoolDynamic.sol#L513, PoolDynamic.sol#L523-L532, PoolDynamic.sol#L711-L732

Description: There are many externally accessible functions in `PoolDynamic` contract that are empty and/or always return `0`. These are:

1. `crossChainTransferApproveSource`.
2. `crossChainTransferApproveDestination`.
3. `crossChainTransferBurnSource`.
4. `crossChainTransferMintDestination`.
5. `crossChainTransferStatus`.
6. `version`.
7. `totalAssets`.
8. `maxDeposit`.
9. `totalAvailableSupply`.
10. `liquidityPoolAssets`.

Empty function and functions that always return `0` can cause integration issues with other contracts that interact with `PoolDynamic` contract, leading to unintended outcomes.

Recommendation: Consider explicitly reverting in functions that are not implemented.

```
function foo(/*...*/) public /*...*/ {
  revert NotImplemented();
}
```

OpenTrade: Fixed in commit 597e5256. These are there for backward compatibility to the prior generation of Pools I moved defintions out of the base interface to the `IPool` and `IPoolFlex` to remove the function.

Cantina Managed: Fix verified.

3.3.12 Interest rate setters can be improved

Severity: Low Risk

Context: PoolDynamic.sol#L411, PoolDynamic.sol#L418, PoolDynamic.sol#L429, PoolDynamic.sol#L444, PoolDynamic.sol#L457

Description: The interest rate setter functions have multiple issues such as:

1. Lack of input validation:
   - `0` can be provided as input in `setExchangeRateDynamic`. As shares are calculated by dividing assets by exchange rate, dividing by zero can result in evm panic error.
   - `1e18` can be provided as input in `setExchangeRateCompounding`.
   - same case with other setters.
2. Setter functions do not emit events. Setting interest rates is a crucial parameter change. Ideally appropriate events must be emitted whenever interest parameters are changed.
3. The interest rate setter functions can be combined and simplified. It'll be better to take 6 input parameters (or a struct) in a single `setExchangeRate` function, some validations can be performed to differentiate the types of interest rates.

Recommendation: Consider performing validation for all inputs, emit events and simplify the setter functions.

OpenTrade: The exchange Rate settting has been refactored in order to cover the issues you raised and to fit the contract into 24k. It is commits a8b1c251 and 84a676b1, followed by additional contract size reduction in commit 1ffb20a4.

Note the exchange rate for compounding is limited to compounding 7 days. It is intended to reset the exchange rate every business day so we avoid long loops.

The `exchangeRateCompounding` is correct in that the expected and validated value will is 1 + daily interest multiplier, while the linear `exchangeChange` rate is the amount the value will increase. Validation is set to ensure at most a 10% change in value per day, (I could potentially lower it to 1% per day).

Now there is an `exchangeRateType` set when the pool is created and never changes. The computed `Exchange` rate is calculated based on the exchange rate type. The exchange Rate validation is implemented in the Controller to save space. Note the exchange rate for compounding is limited to compounding 7 days. It is intended to reset the exchange rate every business day so we avoid long loops. The `exchangeRateCompounding` is correct in that the expected and validated value will is 1 + daily interest multiplier, while the linear `exchangeChange` rate is the amount the value will increase. Validation is set to ensure at most a 10% change in value per day, (I could potentially lower it to 1% per day).

Removing `SafeMafe` did not reduce contract size so I kept it.

Cantina Managed: Fix verified.

3.3.13 `PoolControllerDynamic`: Pool states can be toggled randomly

Severity: Low Risk

Context: PoolControllerDynamic.sol#L201-L224

Description: A `PoolDynamic` contract can have these possible states:

```
{ Initialized, Active, PausedDeposits, Closed, DisruptionOrDefault }
```

The `PoolControllerDynamic` has setter functions to change pool states. However as per their current implementations, pool states can be toggle in any order, for example:

- `activatePool` can be used to change state from `DisruptionOrDefault` to `Active`.
- `resumeDeposits` can be used to activate the pool for the first time.
- `activatePool` can be used to change state from `Closed` to `Active`.
- And many more...

All these possible permutations of pool state transition are unintended and can cause issues with normal operations of pool.

Recommendation: Consider adding validations to limit the pool state transition scenarios. Every function must check the current pool state before changing it.

Example:

```
function changeStateToYYY() external ... {
    IPoolLifeCycleStateDynamic _currentState = state();
    if (_currentState != XXX) {
        revert InvalidStateTransition();
    }
    _setState(YYY);
}
```

OpenTrade: Disconnected `DepositState` (inactive and active) from `withdrawState` (inactive and active).

Cantina Managed: Partially fixed at commit 6f22a0ff.

3.3.14 Missing whitelist check for `lender` in `repayRedemption`

Severity: Low Risk

Context: PoolDynamic.sol#L352-L357

Description: The `repayRedemption` function does not check the whitelist status of `lender` due to which a redemption can be processed for a non-whitelisted lender. This can happen when the lender gets removed from whitelist after creating a redemption request.

Recommendation: Consider adding a whitelist check for `lender` in the `repayRedemption` function.

OpenTrade: Fixed in commit c1219f81. Changed, we may need to liquidate after removal from whitelist we can do that with offchain transfers or liquidate then removal.

Cantina Managed: Fix verified.

3.3.15 Hardcoded pool token decimals

Severity: Low Risk

Context: PoolDynamic.sol#L139-L141

Description: The `decimals` value of `PoolDynamic` token is hardcoded to `6`. This works fine for `_liquidityAsset` which also has `6` decimals like USDC, but it won't be ideal for assets whose decimals are not `6` like DAI. For a token with `18` decimals, pool token decimals = `6`, exchange rate = `1e18`, for `deposit` of `100e18` asset tokens, the number of share will be calculated as:

```
shares = assets.mul(1e18).div(exchangeRate());
shares = 100e18 * 1e18 / 1e18
shares = 100e18
```

These `100e18` pool tokens will be minted to lender. As pool token decimals are `6` the lender's pool token balance will be shown as `1,000,000,000,000` tokens in their wallets and etherscan. This could be unintended.

Recommendation: Consider changing the `decimals` function to this:

```
function decimals() public pure override returns (uint8) {
    return _liquidityAsset.decimals();
}
```

This way the decimals work all type of asset tokens with different decimals. Or, in the `initialize` function validate that `_liquidityAsset.decimals() == 6`.

OpenTrade: Fixed in commit c22d1bb1.

Cantina Managed: Fix verified.

3.3.16 Consider update the max deposit amount

Severity: Low Risk

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

Description: The max deposit is hardcoded.

```
function maxDeposit(address owner) public view override returns (uint256) {
    if (
        _serviceConfiguration.paused() == true ||
        !isPermittedLender(owner) ||
        poolControllerDynamic.state() != IPoolLifeCycleStateDynamic.Active ||
        poolControllerDynamic.depositState() != IPoolDepositActiveStateDynamic.Inactive
    ) {
        return 0;
    }
    return 1_000_000_000_000000;
}
```

The `1_000_000_000_000000` is `10**15`. However, if the underlying liquidity token is DAI (a 18 decimals), and one DAI is `10**18` wei. Then the max deposit `10**15` is too small.

Recommendation: Consider add a state to store the max deposit amount and add a setter in case the admin needs to update the max deposit given a user.

```
function maxDeposit(address owner) public view override returns (uint256) {
    if (
        _serviceConfiguration.paused() == true ||
        !isPermittedLender(owner) ||
        poolControllerDynamic.state() != IPoolLifeCycleStateDynamic.Active ||
        poolControllerDynamic.depositState() != IPoolDepositActiveStateDynamic.Inactive
    ) {
        return 0;
    }
    return maxDepsoit;
}
```

and

```
function updateMaxDepsoit(uint256 _maxDeposit) onlyPoolAdmin {
    maxDepsoit = _maxDeposit
}
```

OpenTrade: Fixed in commit a029a613. Max deposit isnt really used, I did change it return 1_000_000_-000 * 10 ** decimals(); to make it scale to the decimals. Using another vaible puts it over the contract size limit, which doesn't make sense for an unused parameter.

Cantina Managed: Fix verified.

3.3.17 `maxDeposit` returns `0` when pool's deposit state is `Active`

Severity: Low Risk

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

Description: The `PoolDynamic.maxDeposit` function returns `0` when `poolControllerDynamic.depositState() != IPoolDepositActiveStateDynamic.Inactive`. Due to which the `maxDeposit` returns `0` when deposits are active.

Recommendation: Consider changing `Inactive` to `Active` in the check.

OpenTrade: Fixed in commit b89b8f85. Max deposit isn't really used, I did change it return `1_000_000_-000 * 10 ** decimals()`; to make it scale to the decimals. Using another vaible puts it over the contract size limit, which doesn't make sense for an unused parameter.

Cantina Managed: Fix verified.

3.3.18 `PoolControllerDynamic.activatedAt` state is never set

Severity: Low Risk

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

Description: The commit 6f22a0ff moved the `activatedAt` state from `PoolDynamic` to `PoolControllerDynamic` contract. However the `PoolControllerDynamic.activatedAt` state is never set and always remains `0`.

Recommendation: Consider setting the state when pool is activated.

OpenTrade: Fixed in commit 23eccfd9.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Self-approval allowance can be removed to save gas in `PoolControllerDynamic.sol`

Severity: Gas Optimization

Context: PoolControllerDynamic.sol#L112

Description: There is no fund transfer fund in PoolControllerDynamic.sol, so self-approval allowance can be removed to save gas.

Recommendation:

```
- _liquidityAsset.safeApprove(address(this), type(uint256).max);
```

OpenTrade: Fixed in commit 57f3e7ae.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Address open TODO in the comment

Severity: Informational

Context: PoolDynamic.sol#L473

Description: Open `TODO`s should be addressed.

Recommendation: Ensure the interest rate compound at end of business day.

OpenTrade: Fixed in commit a029a613. We decided to always compound at midnight GMT time. Removed the `TODO` comment.

Cantina Managed: Fix verified.

3.5.2 `safeTransfer` can be used to save gas in `PoolDynamic.sol`

Severity: Informational

Context: PoolDynamic.sol#L200

Description: The code gives max self-approval to ensure that the code can trigger `_liquidityAsset.safeTransferFrom`.

```
_liquidityAsset.safeTransferFrom(address(this), fundsDestination, repayment);
```

However, if the fund is in the PoolDynamic Smart contract, `safeTransfer` can be used to save gas.

```
_liquidityAsset.safeTransfer(fundsDestination, repayment);
```

Recommendation: Use `safeTransfer` can be used to save gas and remove the self-approval.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.5.3 Other informational issues

Severity: Informational

Context: PoolControllerDynamic.sol#L63, PoolControllerFactoryDynamic.sol#L33-L40, IPoolDynamicStructures.sol#L65, PoolDynamic.sol#L148, PoolDynamic.sol#L321, PoolDynamic.sol#L445-L446, PoolDynamic.sol#L490, PoolDynamic.sol#L496-L497, PoolDynamic.sol#L515, PoolDynamic.sol#L519, PoolDynamic.sol#L613, PoolDynamic.sol#L632, PoolDynamic.sol#L732, ServiceConfigurationV5.sol#L32

List of issues:

1. `PoolDynamic.sol#L148`: Rename `onlyBorrowerManger` to `onlyBorrowerManager`.
2. `PoolDynamic.sol#L321`: Rename `changeRedemtionDestination` to `changeRedemptionDestination`.
3. `PoolDynamic.sol#L732`: `crossChainTransferStatus` can be marked as `view`.
4. `PoolDynamic.sol#L445-L446`: Across the contract, either remove _ from all function parameters or make all parameters to start with _.
5. `PoolDynamic.sol#L496-L497`: No need to multiply and divide by `1e6`.
6. `PoolDynamic.sol#L632`: Validate that `transferTxHash` is not `0`.
7. `PoolDynamic.sol#L613`: Throughout the contract the `<= 0` checks for `uint256` parameters can be replaced with `== 0`.

8. PoolDynamic.sol#L515-L521: `convertToShares` and `convertToAssets` function can be marked as `public`.
9. PoolDynamic.sol#L490: Replace > with >=.
10. IPoolDynamicStructures.sol#L65: Explicitly use `enum` datatype instead of `uint8` to store redemption request stage.
11. ServiceConfigurationV5.sol#L32: Rename `isfeeCollector` to `isFeeCollector`.
12. PoolControllerFactoryDynamic.sol#L35: Validate that the input `serviceConfiguration` parameter of `createController` is equal to `_serviceConfiguration` state.
13. PoolControllerDynamic.sol#L63: Replace `NotPaused` error with `Paused` error.

OpenTrade: Fixed in commit a029a613.

Cantina Managed: Fix verified.

3.5.4 Any fee amount can be charged from lenders at withdrawal

Severity: Informational

Context: PoolDynamic.sol#L352-L357

Description: As per the `repayRedemption` function, the `repayment` amount has no relation to the requested `assets` amount. There is no relation with current `exchangeRate` as well. Similarly, `fees` can also be any amount irrespective of requested `assets` amount. The pool admin determines and chooses appropriate `repayment` and `fees` amounts off-chain. These values may or may not be acceptable to lenders.

Recommendation: In an ideal scenario, lenders should be able to choose `minRepayment` and `maxFees` values for their asset withdrawals.

OpenTrade: This was a design decision, we can put confirmations into the UI in the future if need be.

Cantina Managed: Acknowledged.

3.5.5 Breaking CEI pattern in `repayRedemption`

Severity: Informational

Context: PoolDynamic.sol#L370-L386

Description: The `repayRedemption` function in `PoolDynamic` contract breaks the checks-effects-interactions (CEI) pattern. The function performs `_liquidityAsset.safeTransferFrom` external call before updating its own storage states. Depending upon the implementation of `_liquidityAsset` token, it may expose the `repayRedemption` to re-entrancy issues.

Recommendation: Consider performing the `safeTransferFrom` call after updating the storage of `PoolDynamic` contract.

OpenTrade: Fixed in commit b5b1edb5.

Cantina Managed: Fix verified.

3.5.6 Unused code and file imports

Severity: Informational

Context: PoolDynamic.sol#L28-L29

Description: The following files have unused code, libraries and imports which can be removed:

- `PoolDynamic.sol`:
  - import and usage of `EnumerableSet` library.
  - import and usage of `SafeMath` library.
  - import of `IVault.sol`.
- `PoolControllerDynamic.sol`:

- import and usage of `SafeERC20` library.
- import of `IVault.sol`.
- import of `PoolLib.sol`.
- import of `IVaultFactory.sol`.

- ServiceConfigurationV5.sol':
  - import of `AccessControlUpgradeable.sol`.
  - import of `DeployerUUPSUpgradeable.sol`.

Recommendation: Consider removing unnecessary code.

OpenTrade: Fixed in commit 328dd239. I kept `SafeMath` in `PoolDynamic`. Using it made the contract slightly smaller.

Cantina Managed: Fix verified.

3.5.7 `requestRedeemViaTransfer` and `requestRedeem` contain duplicate logic

Severity: Informational

Context: PoolDynamic.sol#L654-L664

Description: `requestRedeemViaTransfer` has duplicate logic with the function `requestRedeem`, both function create a transfer request to start the redeem process.

Recommendation: Recommend merge them to not implement the same function twice.

OpenTrade: Fixed by merging the duplicate logic to the function `requestRedeemExecute`.

Cantina Managed: Fix verified.

3.5.8 Unused code after the fix

Severity: Informational

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

Description: After the fix, there are some unused code.

1. The admin or borrorwer role can update the state below but these code is only used in the view function:

```
_settings.closeOfDepositTime = _closeOfDepositTime;
_settings.closeOfWithdrawTime = _closeOfWithdrawTime;
_settings.transferOutDays = _transferOutDays;
```

2. `maxDeposit` is not used.
3. Pool controller does not have the transfer function.

So the check `msg.sender != address(poolControllerDynamic)` can be removed.

Recommendation: Remove or use the unused code.

OpenTrade: Acknowledegd:

1. These are intended only to be view only. there are informational and kept around for backward compatibility.
2. Max deposit is also backward compatibility I put it at $1B because we are not really limited in the ability to buy the backing assets.
3. `msg.sender != address(poolControllerDynamic)` is needed for the add a credit to an account fromt he pool controller.

Cantina Managed: Acknowledged.

3.5.9 Explicitly check return value of `EnumerableSet`'s `add` and `remove` functions

Severity: Informational

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

Description: The `EnumerableSet`'s `add` and `remove` functions return `boolean` value which is ignored by the `PoolDynamic` contract.

Recommendation: Consider explicitly checking the returned value.

```
if (!activeWithdrawKeys.add(uuid)) revert();
```

```
if (!activeWithdrawKeys.remove(uuid)) revert();
```

OpenTrade: Fixed in commit 5b54741a.

Cantina Managed: Fix verified.