kpk Onchain Investment Vehicles

Security Review

Cantina Managed review by:
Gerard Persoon, Lead Security Researcher
Slowfi, Security Researcher

December 17, 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 level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 are 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

From Oct 22nd to Oct 26th the Cantina team conducted a review of karpatkey-tokenized-fund on commit hash df62b723. The team identified a total of **56** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	5	5	0
Low Risk	10	8	2
Gas Optimizations	11	9	2
Informational	28	24	4
Total	**56**	**48**	**8**

The Cantina Managed team reviewed kpk's onchain-investment-vehicles holistically on commit hash 395ceca9 and concluded that all findings were addressed and no new vulnerabilities were identified.

> **Note:** kpk's public repository onchain-investment-vehicles on commit hash 395ceca9 is the same version as the private repository karpatkey-tokenized-fund on commit hash fd9e1fb1 (modulo some directory changes).

2.1 Scope

The security review had the following components in scope for karpatkey-tokenized-fund on commit hash df62b723:

```
contracts/src
├── fund
│   ├── FeeModules
│   │   └── IPerfFeeModule.sol
│   ├── IkpkShares.sol
│   └── kpkShares.sol
└── RecoverFunds.sol
```

3 Findings

3.1 High Risk

3.1.1 Asset removal during pending subscriptions inflates supply via zero-decimals recomputation

Severity: High Risk

Context: kpkShares.sol#L493-L509, kpkShares.sol#L684-L694, kpkShares.sol#L852-L861

Description: Approving a pending subscription checks a price guard by recomputing `sharesOut` using the current asset *global storage mapping, but it mints the user's previously stored sharesAmount** (minOut). After an asset is removed, its config entry including decimals is deleted. The recomputation of `assetsToShares` reads that the given asset has zero decimals. This inflates the computed guard value and lets an oversized stored `sharesAmount` pass. The contract then mints the previously stored amount. The root cause is relying on mutable/global config at approval time for a request created under a different configuration.

Proof of Concept: The test below shows that removing the asset between request creation and approval causes the approval to mint far more shares than correct. After removal, the decimals used in the conversion are effectively `0`, and the resulting `sharesOut` is inflated (here by `1e6`), which is then minted to the user:

```
function test_processRequestsSubscriptionInflatesSharesAfterAssetRemoval() public {
    KpkShares shares = deployDefaultShares();
    uint256 price = 1e8;
    uint256 assetsIn = 400e6;
    uint256 correctShares = shares.assetsToShares(address(usdc), assetsIn, price);
    uint256 inflatedShares = correctShares * 1e6;

    vm.prank(investorA);
    usdc.approve(address(shares), assetsIn);

    vm.prank(investorA);
    uint256 requestId = shares.requestSubscription(assetsIn, inflatedShares,
    ↪  address(usdc), investorA);

    vm.prank(operator);
    shares.updateAsset(address(usdc), true, false, false);

    uint256[] memory approveList = new uint256[](1);
    approveList[0] = requestId;

    vm.prank(operator);
    shares.processRequests(approveList, new uint256[](0), address(usdc), price);

    uint256 minted = shares.balanceOf(investorA);
    assertEq(minted, inflatedShares);
    assertGt(minted, correctShares);
    assertEq(shares.subscriptionAssets(address(usdc)), 0);
}
```

Explanation: approval recomputes `sharesOut` using the current (now-deleted) asset config. With decimals read as `0`, scaling is wrong and the function over-mints.

Recommendation: Decoupling deprecation from deletion: first set `canDeposit = false` while keeping `canRedeem == true` to drain pending flows; only allow full deletion when `subscriptionAssets[asset]==0` and there are no pending requests.

So, consider to:

1. Revert subscription processing if asset is not approved or decimals missing.
2. Block asset removal while `subscriptionAssets[asset] > 0`.
3. Mint computed shares, not requested min.

kpk: Fixed in commit 20a83ba7.

Cantina Managed: Fix verified.

3.1.2 Subscription approval mints `sharesAmount` instead of computed `sharesOut` and event

Severity: High Risk

Context: kpkShares.sol#L684-L694

Description: In the subscription approval path, the mint uses the originally requested `sharesAmount` rather than the computed `sharesOut` at approval time. This couples the mint quantity to user input instead of the price-derived calculation and can lead to under-minting relative to the current price. The corresponding event also reflects the requested value rather than the computed one.

Recommendation: Consider to:

1. Compute `sharesOut` from the approved `assetsIn` and `sharesPriceInAsset` at approval,
2. Enforce `sharesOut >= minSharesOut` (the user's slippage bound), then.
3. Mint `sharesOut` to the receiver and emit the event using `sharesOut`.

kpk: Fixed in commit 1ecc6297.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Mistakes in `sharesPriceInAsset` have big consequences

Severity: Medium Risk

Context: kpkShares.sol#L382-L391

Description: Mistakes in `sharesPriceInAsset` of `processRequests()` have big consequences, enough to rekt the entire protocol. This could prevent institutions from using the protocol.

Recommendation: Consider adding more seperation of duties to reduce the probability of mistakes. For example have a seperate role that sets the prices. Also consider comparing to previous prices and only accept small deviations from the previous prices.

kpk: Fixed in commit b6ec22d5.

Cantina Managed: Fix verified.

3.2.2 `assetsToShares()` doesn't check the `asset` exists

Severity: Medium Risk

Context: kpkShares.sol#L493-L502, kpkShares.sol#L512-L521

Description: The funtion `assetsToShares()` doesn't check the `asset` exists. This allows for using a non existing asset.

This in one of the causes of the related issue: ”Asset removal during pending subscriptions inflates supply via zero-decimals recomputation”. It could also allow minting shares in exchange for non supported or worthless assets if the `operator` accidentally allows it. For comparison: `sharesToAssets()` does check the `asset` exists.

Recommendation: Consider changing the code to:

```
  function assetsToShares(address subscriptionAsset, uint256 assetAmount, uint256
  ↪  sharesPrice) ... {
    // ...
    ApprovedAsset memory assetConfig = _approvedAssetsMap[subscriptionAsset];
+   if (!assetConfig.canDeposit) revert ...;
    // ...
  }
```

kpk: Fixed in commit cf58493f.

Cantina Managed: Fix verified.

3.2.3 Performance fee gated by USD assets enables fee avoidance via asset choice and ordering

Severity: Medium Risk

Context: kpkShares.sol#L784-L803, kpkShares.sol#L793-L795

Description: Performance fees are only charged when processing a batch for an asset flagged `isUsd`. This creates two gaming vectors:

- Asset choice: at the same price, users are better off redeeming in a non-USD asset to avoid the performance fee entirely.
- Processing order/time: operators can first process non-USD batches and then a USD batch shortly after, such that `timeElapsed` is below the accrual threshold-skipping fees. The same effect can occur when there are multiple USD assets, depending on which one is used and when.

The result is systematic under-collection of performance fees driven by operational sequencing and asset selection rather than fund economics.

Recommendation: Consider to decouple performance-fee accrual from the batch asset and make it non-gameable:

- Always accrue on processing, regardless of the asset being processed; only use the asset for price input, not as a gate.
- Or track a `lastFeeUpdate` per asset (or per USD asset) and compute/mint fees using that asset's own clock to prevent cross-asset/order gaming.
- Add tests covering: non-USD-first sequencing, multiple USD assets, and short-interval back-to-back processing.

kpk: Fixed in commit fd9e1fb1.

Cantina Managed: Fix verified.

3.2.4 Asset removal bricks pending requests and enables sweeping escrowed funds

Severity: Medium Risk

Context: kpkShares.sol#L554-L557, kpkShares.sol#L852-L862

Description: Removing an `asset` while requests are `pending`, deletes its configuration and permits recovery to sweep escrowed subscription balances to the `portfolio safe`.

After a sweep, subscription cancels/approvals fail due to insufficient balance on the shares contract.

In parallel, pending redemptions for the removed asset cannot be approved because `sharesToAssets` enforces `canRedeem`, which is now false/missing bricking user exits. However users can then wait for `ttl` to expire and call `cancelRedemption` to get their shares back. Also it is possible for the operator to reject the redeems.

Additionally in `_assetRecoverableAmount()`, `_approvedAssetsMap[asset]` will be zero, so `_approvedAssetsMap[]` will not be used and the entire balance can be sweeped.

Proof of Concept: Subscriptions swept, cancel fails:

```
function test_cancelSubscriptionFailsAfterAssetRemovalAndRecovery() public {
    KpkShares shares = deployDefaultShares();
    uint256 assetsIn = 200e6;
    uint256 sharesOut = shares.assetsToShares(address(usdc), assetsIn, 1e8);

    vm.prank(investorA);
    usdc.approve(address(shares), assetsIn);

    vm.prank(investorA);
    uint256 requestId = shares.requestSubscription(assetsIn, sharesOut, address(usdc),
    ↪  investorA);

    vm.prank(operator);
    shares.updateAsset(address(usdc), true, false, false); // remove asset
```

```
    address;
    assets[0] = address(usdc);
    vm.prank(makeAddr("sweeper"));
    shares.recoverAssets(assets); // sweeps escrow to portfolioSafe

    vm.warp(block.timestamp + shares.subscriptionRequestTtl() + 1);

    vm.prank(investorA);
    vm.expectRevert(abi.encodeWithSignature(
        "ERC20InsufficientBalance(address,uint256,uint256)", address(shares), 0, assetsIn
    ));
    shares.cancelSubscription(requestId); // refund fails: funds swept
}
```

Redemptions bricked on approval:

```
function test_processRequestsRedemptionFailsAfterAssetRemoval() public {
    KpkShares shares = deployDefaultShares();
    uint256 subscriptionAssetsIn = 600e6;
    uint256 price = 1e8;

    // Subscribe and approve
    vm.startPrank(investorA);
    usdc.approve(address(shares), subscriptionAssetsIn);
    shares.requestSubscription(subscriptionAssetsIn,
        shares.assetsToShares(address(usdc), subscriptionAssetsIn, price),
        ↪  address(usdc), investorA);
    vm.stopPrank();
    uint256;
    approveSub[0] = 1;
    vm.prank(operator);
    shares.processRequests(approveSub, new uint256, address(usdc), price);

    // Request redemption
    uint256 sharesIn = shares.balanceOf(investorA);
    uint256 assetsOut = shares.sharesToAssets(sharesIn, price, address(usdc));
    vm.prank(investorA);
    uint256 requestId = shares.requestRedemption(sharesIn, assetsOut, address(usdc),
    ↪  investorA);

    // Remove asset before approval → UnredeemableAsset
    vm.prank(operator);
    shares.updateAsset(address(usdc), true, false, false);

    uint256;
    approveList[0] = requestId;
    vm.prank(operator);
    vm.expectRevert(IkpkShares.UnredeemableAsset.selector);
    shares.processRequests(approveList, new uint256[](0), address(usdc), price);
}
```

Explanation: Deleting the asset leaves pending requests referencing an asset whose config is gone. Recovery ignores escrow protection after deletion, enabling sweeps; redemptions then fail because exits are disabled for the removed asset.

Recommendation: Consider to:

- Two-phase deprecation: set `canDeposit=false` but keep `canRedeem=true` (”exit-only” mode) to drain pending flows.
- Gate full deletion: only allow removal when there are no pending subscriptions/redemptions and `subscriptionAssets[asset] == 0`.
- Harden recovery: return zero recoverable for tokens with pending requests and always exclude recorded escrow, regardless of the asset's current config.
- Optionally track per-asset pending counters (subs/reds) and enforce the guards above; as an extra

defense, snapshot an exit-eligible flag per redemption request at creation.

kpk: Fixed in commit 20a83ba7.

Cantina Managed: Fix verified.

3.2.5 Allowing the share token as an approved asset enables circular accounting

Severity: Medium Risk

Context: kpkShares.sol#L554-L562, kpkShares.sol#L852-L854

Description: Function `updateAsset` prevents the `zero` address but not `asset == address(this)`. During the redemption path, user shares are first escrowed on the shares contract before the burn occurs. If the share token itself can be registered as an approved asset, those escrowed shares become a pseudo-"underlying" balance and can be mis-accounted or swept by operational flows, enabling artificial inflation and a shares drain driven by the fact that shares temporarily remain on the contract during redemption.

Recommendation: Consider to reject `asset == address(this)` in `_updateAsset()` (and any asset add/update function). Make sure that accounting never treats escrowed `shares` as recoverable or depositable value. Also ensure any asset-recovery routine hard-blocks the `share` token. The `_assetRecoverableAmount()` function already tries to do this, but this fails if the `asset` is approved. So consider changing the code in the follow way as an extra safety precaution:

```
  function _assetRecoverableAmount(address token) ... {
+     if (token == address(this)) { return 0; }
      if (_approvedAssetsMap[token].asset != address(0)) {
          return IERC20(token).balanceOf(address(this)) - subscriptionAssets[token];
      }
-     if (token == address(this)) { return 0; }
      // ...
  }
```

kpk: Fixed in commit cd48604f.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Out of gas with a huge number of assets

Severity: Low Risk

Context: kpkShares.sol#L463-L469, kpkShares.sol#L893-L902

Description: If a huge number of assets are added then the functions `getApprovedAssets()` and `_shadowAsset()` could run out of gas. Adding assets is an authorized function so this won't happen in practice.

Recommendation: Make sure not to add a huge amount of assets. If desired this can also limited in the source code.

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Last asset can be removed

Severity: Low Risk

Context: kpkShares.sol#L784-L795, kpkShares.sol#L852

Description: Once the last asset has been removed it is no longer possible to do `requestRedemption()` or `previewRedemption()`. This could make it more difficult to query the value of the shares. When the last stablecoin asset is removed, then `_chargePerformanceFee()` will no longer be called, until a new stablecoin asset is added. Normally an operator would not remove the last asset and it could be the intention to temporarily `close` the contract. Other protocols often prevent removing a last item.

Recommendation: Consider preventing to remove the last (stablecoin) asset and/or asset with `canRedeem == true`.

kpk: Fixed in commit 71fa832e.

Cantina Managed: Fix verified.

3.3.3 `feeReceiverBalance` might not be accurate in `_chargeManagementFee()`

Severity: Low Risk

Context: kpkShares.sol#L823-L830

Description: The `feeReceiverBalance` might not be accurate in `_chargeManagementFee()` because it retrieved via `balanceOf(feeReceiver)`, but the `feeReceiver` could do other actions with the fees, for example put them a defi protocol to gain extra yield.

If something is done with the `shares` in the `feeReceiver` then `feeReceiverBalance` will be lower (possibly 0) and then the `feeAmount` will increase.

The intention of the project is to only redeem the `shares` for `assets`. But for investors in the protocol this is difficult to verify.

Recommendation: Consider to have a `feeReceiver` contract that only allows to redeem the `shares` for `assets`.

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.3.4 Status update after `_transfer()` in `_rejectRedeemRequest()`

Severity: Low Risk

Context: kpkShares.sol#L706-L714, kpkShares.sol#L768-L772

Description: Function `_rejectRedeemRequest()` does a status update to after `REJECTED` after the call to `_transfer()`. However if there would ever be an external call from `_transfer()` the status isn't updated yet and unwanted actions might be done. In the current code no external call can be done, but it is still safer to change the status first.

> **Note:** The comparable function `_rejectSubscriptionRequest()` changes the status first.

Recommendation: Consider changing the code to:

```
  function _rejectRedeemRequest(...) ... {
+     _requests[id].requestStatus = RequestStatus.REJECTED;
      _transfer(address(this), request.investor, request.sharesAmount);
-     _requests[id].requestStatus = RequestStatus.REJECTED;
      // ...
  }
```

kpk: Fixed in commit 38195cca.

Cantina Managed: Fix verified.

3.3.5 Updating `storage` while `memory` copy exists

Severity: Low Risk

Context: kpkShares.sol#L646-L659, kpkShares.sol#L663-L674, kpkShares.sol#L684-L701, kpkShares.sol#L706-L709, kpkShares.sol#L738-L753, kpkShares.sol#L768-L772

Description: Function `_approveSubscriptionRequest()` uses `memory` copy of `_requests[id]` and update the underlying storage via:

```
_requests[id].requestStatus = RequestStatus.PROCESSED;
```

After this statement the value of `request.requestStatus` is no longer in line with the `storage`. However `request` is still used.

This could lead to mistakes, both in the function and the calling function `_processApproved()`. In the current code there is no issue because `request.requestStatus` isn't used, but future updates could potentially use is.

The same situation exists in `_rejectSubscriptionRequest()`, `_approveRedeemRequest()` and `_rejectRedeemRequest()`.

Recommendation: Consider changing `request` to a `storage` pointer in `_processApproved()` and `_processRejected()`. Alternatively add comments to warn about the potential issue.

kpk: Fixed in commit d70ef1eb.

Cantina Managed: Fix verified.

3.3.6 Difference `_processApproved()` and `_processRejected()`

Severity: Low Risk

Context: kpkShares.sol#L646-L652, kpkShares.sol#L663-L667

Description: Function `_processApproved()` makes sure `request.asset == asset`, however function `_processRejected()` doesn't do this. This doesn't seem logical and could allow mistakes to go through.

Recommendation: Consider also adding the following to `_processRejected()`:

```
  function _processRejected(...) ... {
      for (...) {
          UserRequest memory request = _requests[rejectRequests[i]];
          if (!_validateRequest(request)) continue;
+         if (request.asset != asset) continue;
          // ...
      }
  }
```

kpk: Fixed in commit e99386cc.

Cantina Managed: Fix verified.

3.3.7 Function `requestSubscription()` updates administration before receiving tokens

Severity: Low Risk

Context: kpkShares.sol#L213-L240, kpkShares.sol#L256-L270, kpkShares.sol#L403-L407

Description: Function `requestSubscription()` updates the administration before receiving the tokens, it would be safer to first receive the tokens and then set the record.

Here is an example of theoretical risk, that does not manifest though: With `ERC777` tokens a callback is called from `safeTransferFrom()`. The callback could then try to do a `cancelSubscription()` and receive tokens. Luckily this fails because `timestamp + subscriptionRequestTtl` is checked and `subscriptionRequestTtl` may not be `0`. Also normally `ERC777` tokens would not be added.

Recommendation: Consider updating `_requests[]` after the tokens have been received with `safeTransferFrom()`.

kpk: Fixed in commit b8a675a7.

Cantina Managed: Fix verified.

3.3.8 Not all init functions are called

Severity: Low Risk

Context: kpkShares.sol#L156-L159

Description: The libraries `AccessControlUpgradeable` and `ERC20Upgradeable` have underlying contracts that are not initialized.

Although these functions are currently empty, future versions might have code so its safer to call the init functions.

Recommendation: Consider also calling the following init functions:

```
__Context_init();
__ERC165_init();
```

kpk: Fixed in commit 5401d6b3.

Cantina Managed: Fix verified.

3.3.9 Incorrect timestamp emitted for redemption request events

Severity: Low Risk

Context: kpkShares.sol#L213, kpkShares.sol#L242-L250, kpkShares.sol#L314, kpkShares.sol#L343-L345

Description: The redemption request event emits the current `timestamp`, whereas the interface documentation specifies emitting `timestamp + ttl`. This mismatch can lead off-chain indexers or integrators to derive an incorrect cancelable/expiry moment for redemption requests. For comparison, `requestSubscription()` does have `timestamp + ttl`.

Recommendation: Consider to emit `timestamp + ttl` in the redemption request event to align with the documented interface.

Alternatively, update the interface/docs and event naming to reflect that the field represents the creation timestamp, and add a separate field for the effective `timestamp + ttl` if needed. Also doublecheck `requestSubscription()`.

kpk: Fixed in commit 6c0ec543.

Cantina Managed: Fix verified.

3.3.10 `minAssetsOut` checked against gross instead of net in redemption

Severity: Low Risk

Context: IkpkShares.sol#L377-L384, kpkShares.sol#L746-L751

Description: The function `_approveRedeemRequest` from contract `kpkShares` validates `minAssetsOut` using `request.sharesAmount` before subtracting the redemption fee. Because the user's transfer is based on `request.sharesAmount - redemptionFee`, the actual assets sent can be lower than the user's stated `minAssetsOut`, misaligning expectations and weakening slippage protection.

Recommendation: Consider to enforce `minAssetsOut` on the net amount:

- Compute `netShares = request.sharesAmount - redemptionFee`, then `assetsOutNet = sharesToAssets(netShares, operatorPrice, request.asset)`, and compare `assetsOutNet >= request.assetAmount`.
- Alternatively, convert the fee to assets at the same price and compare `(assetsOutGross - feeInAssets) >= request.assetAmount`.

Keep `previewRedemption` and any related events/docs consistent with the net-based check.

kpk: Fixed in commit 866206da.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Validate parameters directly instead of a full struct

Severity: Gas Optimization

Context: kpkShares.sol#L722

Description: `_validateRequest` takes a full `UserRequest` struct but only uses `investor` and `requestStatus`. Passing just those parameters avoids extra memory reads/copies. The PoC below shows a small but measurable gas reduction (~4-13 gas depending on inputs):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.28;
import "hardhat/console.sol";

contract testmemory {
    enum RequestType { SUBSCRIPTION, REDEMPTION }
    enum RequestStatus { PENDING, PROCESSED, REJECTED, CANCELLED }
    struct UserRequest {
        RequestType requestType;
        RequestStatus requestStatus;
        address asset;
        uint256 assetAmount;
        uint256 sharesAmount;
        address investor;
        address receiver;
        uint64 timestamp;
    }

    function _validateRequest(UserRequest memory request) internal pure returns (bool) {
        if (request.investor == address(0) || request.requestStatus !=
        ↪  RequestStatus.PENDING) { return false; }
        return true;
    }

    function _validateRequest2(address investor, RequestStatus requestStatus) internal
    ↪  pure returns (bool) {
        if (investor == address(0) || requestStatus != RequestStatus.PENDING) { return
        ↪  false; }
        return true;
    }

    constructor() {
        UserRequest memory request;
        RequestStatus requestStatus = RequestStatus.PROCESSED;
        address investor = address(0);

        request.requestStatus = requestStatus;
        request.investor = investor;

        uint g1; uint g2;

        g1 = gasleft(); _validateRequest(request);  g2 = gasleft(); console.log(g1 -
        ↪  g2);      // e.g., 139
        g1 = gasleft(); _validateRequest2(investor, requestStatus); g2 = gasleft();
        ↪  console.log(g1 - g2); // e.g., 135
        // With investor != 0: 213 vs 200 in one run
    }
}
```

Recommendation: Consider to change the signature to accept only the needed fields, e.g., `_validateRequest(address investor, RequestStatus status)`, and update call sites accordingly. This is a micro-optimization; consider readability if the win is not critical.

kpk: Fixed in commit c43a5065.

Cantina Managed: Fix verified.

3.4.2 Single-use cache of `totalSupply()` adds minor gas overhead

Severity: Gas Optimization

Context: kpkShares.sol#L824

Description: A local variable caches `totalSupply()` and is used only once. Storing the value and then using it a single time introduces a small overhead versus calling `totalSupply()` directly at the subtraction site.

Recommendation: Consider to inline `totalSupply()` where it's used if referenced only once. If multiple references are needed, keep the cached variable for readability and to avoid repeated SLOADs.

kpk: Fixed in commit ada817c8.

Cantina Managed: Fix verified.

3.4.3 Calls to `_mint()` can be optimized

Severity: Gas Optimization

Context: kpkShares.sol#L823-L830, kpkShares.sol#L836-L845

Description: In function `_chargeManagementFee()` and `_chargePerformanceFee()`, the `feeAmount` / `performanceFee` could be 0. In that case the `_mint()` can be skipped. This pattern is also used in other parts of the code.

Recommendation: Consider only calling `_mint()` if `feeAmount > 0` or `performanceFee > 0`.

kpk: Fixed in commit f771a173.

Cantina Managed: Fix verified.

3.4.4 Result of `_chargeFees()` not used

Severity: Gas Optimization

Context: IkpkShares.sol#L118-L122, kpkShares.sol#L382-L388, kpkShares.sol#L784, kpkShares.sol#L784-L803

Description: The struct `FeesCharged` contains three fields, however `redemptionFee` is never filled or used. Function `_chargeFees()` is the only function that returns the `struct`, and that only fills `managementFee` and `performanceFee`.

Additionally the result from function `_chargeFees()` is never used.

Recommendation: Doublecheck the usefulness of returning the `fees`. If not useful consider removing the `fee` information struct and return value.

kpk: Fixed in commit f30ccf10.

Cantina Managed: Fix verified.

3.4.5 Redundant check in `previewRedemption()`

Severity: Gas Optimization

Context: kpkShares.sol#L186-L198, kpkShares.sol#L290-L299, kpkShares.sol#L493-L501, kpkShares.sol#L512-L521

Description: Function `previewRedemption()` checks `canRedeem` and then calls `sharesToAssets()` which also check for `canRedeem` . So the same check is done twice, which is redundant.

Recommendation: Consider changing the code to:

```
  function previewRedemption(...) ... {
-     if (!_approvedAssetsMap[redemptionAsset].canRedeem) {
-         revert NotAnApprovedAsset();
-     }
      uint256 assets = sharesToAssets(shares, sharesPrice, redemptionAsset); // checks
      ↪  canRedeem of redemptionAsset
      // ...
  }
```

Once the recommendation of finding "`assetsToShares()` doesn't check the `asset` exists" is implemented, the same can be done for `previewSubscription()`.

kpk: Fixed in commit 2e2ada7a.

Cantina Managed: Fix verified.

3.4.6 Calculations in `assetsToShares()` and `sharesToAssets()` can be optimized

Severity: Gas Optimization

Context: kpkShares.sol#L493-L532

Description: The temporary values used in `assetsToShares()` and `sharesToAssets()` are rather large, which limits the allowable ranges of assets. See the proof of concept below. Also there are two `mulDivs` in both functions, which are relatively expensive.

Proof of Concept: Function `assetsToShares()` reverts with an asset amount of `66_666_666_666_666e36`.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;
import "hardhat/console.sol";

import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";

contract testdecimals {
    using Math for uint256;
    uint256 private constant _PRECISION_WAD = 1e18;
    uint8 assetDec = 36;
    uint256 shareDecimals = 18;
    uint256 sharesPrice = 3e8;

    function assetsToShares(uint256 assetAmount) public view returns (uint256) {
        uint256 assetsValue = assetAmount.mulDiv((10 ** shareDecimals) * _PRECISION_WAD,
        ↪  sharesPrice, Math.Rounding.Floor);
        return assetsValue.mulDiv(1e8, (10 ** assetDec) * _PRECISION_WAD,
        ↪  Math.Rounding.Floor);
    }
    constructor() {
        uint256 a = 66_666_666_666_666e36;
        uint s=assetsToShares(a);  // this reverts
        console.log(s);
    }
}
```

Recommendation: Consider changing the code to:

```
function assetsToShares(uint256 assetAmount) public view returns (uint256) {
    uint256 assetsValue = assetAmount.mulDiv(1e8, sharesPrice, Math.Rounding.Floor);
    return scaleDecimals(assetsValue,assetDec,shareDecimals);
}
function sharesToAssets(uint256 shares) public view returns (uint256) {
    uint256 sharesValue = shares.mulDiv(sharesPrice, 1e8, Math.Rounding.Floor);
    return scaleDecimals(sharesValue,shareDecimals,assetDec);
}
function scaleDecimals(uint256 amount,uint8 fromDecimals,uint8 toDecimals) internal pure
↪  returns (uint256) {
    if (fromDecimals == toDecimals) { return amount; }
    if (fromDecimals > toDecimals) {
        return amount / (10 ** (fromDecimals - toDecimals));
    } else {
        return amount * (10 ** (toDecimals - fromDecimals));
    }
}
```

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.4.7 `Symbol` not used

Severity: Gas Optimization

Context: IkpkShares.sol#L79-L86, kpkShares.sol#L472-L474, kpkShares.sol#L852, kpkShares.sol#L875-L877

Description: The `symbol` of the supported `assets` is retrieved in `_updateAsset()` and then stored in `_approvedAssetsMap[]`. However it isn't used the rest of the code. It is only retrieved in `getApprovedAsset()`. This approach costs unneccesary gas.

Recommendation: Consider not storing the `symbol` in `storage` but only retrieve it in `getApprovedAsset()`. Then also consider retrieving and returning the `name` of the asset.

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 Array length in for loop

Severity: Gas Optimization

Context: kpkShares.sol#L463-L469, kpkShares.sol#L646-L647, kpkShares.sol#L663-L664

Description: In several for loops the value of the array length is evaluated every iteration. This costs additonal gas.

Recommendation: Consider caching the array length.

kpk: Fixed in commit 82116288.

Cantina Managed: Fix verified.

3.4.9 `memory` versus `storage` `UserRequest` in `cancel` functions

Severity: Gas Optimization

Context: kpkShares.sol#L256-L283, kpkShares.sol#L351-L375

Description: The functions `cancelSubscription()` and `cancelRedemption()` use a `storage` pointer for `request`. This means all accesses to `request` require access to storage, which is relatively expensive. In other parts of the code a `memory copy` is used.

Additionally the two functions use a different pattern to update the `requestStatus`:

- `cancelSubscription()` updates via `_requests[id]`;
- `cancelRedemption()` updates via the storage pointer request.

Recommendation: Consider changing the code to:

```
- UserRequest storage request = _requests[id];
+ UserRequest memory request = _requests[id];
  // ...
- request.requestStatus = RequestStatus.CANCELLED;   // only for cancelRedemption
+ _requests[id].requestStatus = RequestStatus.CANCELLED;  // already like this in
↪  cancelSubscription
```

kpk: Fixed in commit fdbb6c54.

Cantina Managed: Fix verified.

3.4.10 Redundant check in `request` functions

Severity: Gas Optimization

Context: kpkShares.sol#L186-L193, kpkShares.sol#L213-L221, kpkShares.sol#L314-L322

Description: The functions `requestSubscription()` and `requestRedemption()` check `assetConfig.asset == address(0)` in combination with `canDeposit` / `canRedeem`. However on other places in the code, like `previewSubscription`, only `canDeposit` / `canRedeem` are checked, which is sufficient.

Recommendation: Consider making the following changs is `requestSubscription()` and `requestRedemption()`:

```
- if (assetConfig.asset == address(0) || !assetConfig.can...) revert ...;
+ if (!assetConfig.can...) revert ...;
```

kpk: Fixed in commit 678aa324.

Cantina Managed: Fix verified.

3.4.11 `preview` functions could be `external`

Severity: Gas Optimization

Context: kpkShares.sol#L186-L190, kpkShares.sol#L290-L293

Description: The `preview` functions could be `external` because they aren't used in the code and to be consistent with the other `view` functions.

Recommendation: Consider making the functions `previewSubscription()` and `previewRedemption()` `external`.

kpk: Fixed in commit 3b390246.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Off-by-one boundary: redemption cancel not allowed exactly at `timestamp + ttl`

Severity: Informational

Context: kpkShares.sol#L362

Description: The function `cancelRedemption` from contract `kpkShares` checks `block.timestamp <= request.timestamp + redemptionRequestTtl` and reverts, which means cancellation is only allowed after the boundary (strictly greater than). The emitted event indicates cancelability starts at `timestamp + ttl`, so at the exact boundary second a cancellation aligned to the event will unexpectedly revert.

Recommendation: Consider to make the boundary inclusive for users by switching the guard to a strict `<` comparison:

```
if (block.timestamp < request.timestamp + redemptionRequestTtl) revert
↪  RequestNotPastTtl();
```

Ensure subscription and redemption paths both use the same inclusive boundary, and keep the event/docs consistent with the check.

kpk: Fixed in commit c43a5065.

Cantina Managed: Fix verified.

3.5.2 Difficult to use `preview` functions

Severity: Informational

Context: kpkShares.sol#L186, kpkShares.sol#L290

Description: The `preview` functions are difficult to use, because there is no easy way to get an estimate for `sharesPrice` for a given `subscriptionAsset` / `redemptionAsset`.

Recommendation: Consider keeping track of recent values for `sharesPrice` per `asset` and use that in the `view` functions.

kpk: Fixed in commit b6ec22d5.

Cantina Managed: Fix verified.

3.5.3 Suggestion for `calculatePerformanceFee()`

Severity: Informational

Context: IPerfFeeModule.sol#L14-L17, kpkShares.sol#L823-L827

Description: The function `calculatePerformanceFee()` receives very limited information. It might be helpful to have more information available to do the calculations.

Recommendation: Consider passing `totalSupply()` as a parameter. Perhaps also subtract the `shares` that have been send to the `feeReceiver`, similar to `_chargeManagementFee()`.

kpk: Fixed in commit e9a8195a.

Cantina Managed: Fix verified.

3.5.4 Comment in `_assetRecoverableAmount()` not accurate

Severity: Informational

Context: RecoverFunds.sol#L26-L29

Description: A comment in `_assetRecoverableAmount()` is not accurate: the comment indicates a `boolean` return type, while the code has an `uint256` return type.

Recommendation: Consider updating the comment.

kpk: Fixed in commit 46382626.

Cantina Managed: Fix verified.

3.5.5 Tables in `README.md` not accurate

Severity: Informational

Context: README.md#L1

Description: The tables in `README.md` are not accurate:

Subscription Flow:

Function	...	Shares Destination	Shares Amount	...
`requestSubscription`	...	None (calculated only)	Calculated shares	...

No share calculation is done in `requestSubscription()`.

Redemption Flow:

Function	...	Asset Destination	Asset Amount	...
`requestRedemption`	...	None (calculated only)	Calculated assets	...

No asset calculation is done in `requestRedemption()`.

Fee Collection Patterns:

Fee Type	...	Frequency	...	Event Emission
Performance Fee	...	Only when `timeElapsed > MIN_TIME_ELAPSED`	...	...
Redemption Fee	...	...	...	N/A (no event)

The frequency for the performance fee also depends on `isUsd`. There is an event for the Redemption Fee: it is part of emit RedemptionApproval.

Recommendation: Consider updating the tables.

kpk: Fixed in commit e9a8195a.

Cantina Managed: Fix verified.

3.5.6 Formula and `sharesPrice` decimals not clear

Severity: Informational

Context: IkpkShares.sol#L287-L297, IkpkShares.sol#L302-L312, kpkShares.sol#L40, kpkShares.sol#L40-L43, kpkShares.sol#L493, kpkShares.sol#L504-L508, kpkShares.sol#L512, kpkShares.sol#L525-L531

Description: The formula for the price calculations in relation to the `sharesPrice` decimals is not clear. The different formulas used are:

- In `IkpkShares::sharesToAssets()`:

```
assets = shares * sharesPrice / 1e18
```

- Comment in `kpkShares::sharesToAssets()`:

```
assets = (shares * sharesPrice * 10^assetDec) / 1e18
```

- In Solidity of `kpkShares::sharesToAssets()`:

```
assets  = (shares * sharesPrice * 10^assetDec) / (1e8 * 1e18)
```

It also depend on the number of decimals of `sharesPrice`, which isn't clear. The comment of `IkpkShares::sharesToAssets()` states:

```
The current price per share in asset units.
```

This could be:

- `asset.decimals()`, e.g. 6 decimals for USDC.
- `_NORMALIZED_PRECISION_USD == 1e8` , e.g. 8 decimals.
- `_PRECISION_WAD == 1e18`, e.g. 18 decimals.

Recommendation: Doublecheck the formulas and make sure they are consistent. Also explicitly document the decimals for the `sharesPrice`.

kpk: Fixed in commit 04341f29.

Cantina Managed: Fix verified.

3.5.7 Comment of event `RedemptionRequest` not accurate

Severity: Informational

Context: IkpkShares.sol#L185-L188

Description: The comments of event `RedemptionRequest` are not accurate.

Recommendation: Consider making the following changes:

```
- /// @param assetsAmount The number of shares being redeemed.
+ /// @param assetsAmount The number of assets being redeemed.
- /// @param sharesAmount The number of assets being used.
+ /// @param sharesAmount The number of shares being used.
```

kpk: Fixed in commit 46382626.

Cantina Managed: Fix verified.

3.5.8 Event name `ManagementRateUpdate` different than other events

Severity: Informational

Context: kpkShares.sol#L937-L947

Description: The event name `ManagementRateUpdate` is slightly different than other events.

Recommendation: Consider making the following change:

```
- emit ManagementRateUpdate(newRate);
+ emit ManagementFeeRateUpdate(newRate);
```

kpk: Fixed in commit 465138e6.

Cantina Managed: Fix verified.

3.5.9 OpenZeppelin `EnumerableMap/EnumerableSet` could be used

Severity: Informational

Context: IkpkShares.sol#L79-L86, kpkShares.sol#L64-L69, kpkShares.sol#L893-L902

Description: Assets are stored in a combo of `_approvedAssets` and `_approvedAssetsMap`, which takes some effort to update. There are library that can simplify this.

Recommendation: Consider using `OpenZeppelin` `EnumerableMap/EnumerableSet`.

> **Note:** the amount of information that can be stored then is limited, but the `symbol` could be left out, see finding "`Symbol` not used"

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.5.10 `decimals()` might not exist

Severity: Informational

Context: kpkShares.sol#L877

Description: In the ERC20 standard, the `decimals()` functions is not mandatory, so the call to `decimals()` might fail.

Recommendation: Make sure to only add tokens that have a `decimals()` function.

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.5.11 Comment in `_approveRedeemRequest()` not accurate

Severity: Informational

Context: kpkShares.sol#L738, kpkShares.sol#L760-L761

Description: A comment in `_approveRedeemRequest()` is not accurate because the `assets` are send to the `receiver` and not the `investor`.

Recommendation: Consider changing the comment in the following way:

```
- // Transfer assets to investor
+ // Transfer assets to receiver
```

kpk: Fixed in commit d70ef1eb.

Cantina Managed: Fix verified.

3.5.12 `_validateRequest()` versus `_validateRequestParams()`

Severity: Informational

Context: kpkShares.sol#L629-L631, kpkShares.sol#L636-L640, kpkShares.sol#L722-L728

Description: The function `_validateRequest()` and `_validateRequestParams()` have a similar name behave differently:

- `_validateRequest()` returns false in unwanted situations;
- `_validateRequestParams()` reverts in unwanted situations.

This makes reviewing and maintaining the code more error phrone.

Recommendation: Consider makeing the difference clear in the function names. Also apply any changes to `_validateCancellationAuthorization()`.

kpk: Fixed in commit b36b7029.

Cantina Managed: Fix verified.

3.5.13 Value of USDC can fluctuate

Severity: Informational

Context: kpkShares.sol#L596-L597

Description: A comment in the code assumes no oracle is needed because USDC is always $1. This assumption isn't correct because the value of stable coins fluctuates and in an extreme case USDC has depegged to $0.88

The price is important during `processRequests()` so is under the influence of the `operator`.

Recommendation: The `operator` should take large fluctuations of USDC/stablecoins into account and perhaps cancel actions during large fluctuations.

kpk: Fixed in commit b36b7029.

Cantina Managed: Fix verified.

3.5.14 `_validateInitializationParams()` allows `performanceFeeModule == address(0)`

Severity: Informational

Context: kpkShares.sol#L452-L456, kpkShares.sol#L579-L586, kpkShares.sol#L836-L839

Description: Function `_validateInitializationParams()` allows `performanceFeeModule == address(0)`, however `setPerformanceFeeModule()` does not allow this. On the other hand `_chargePerformanceFee()` is able to handle `performanceFeeModule == address(0)`.

Recommendation: Determine if an empty `performanceFeeModule` is a desired possibility. If so, also allow it in `setPerformanceFeeModule()`. If it is undesirable, consider checking for it in `_validateInitializationParams()`.

kpk: Fixed in commit 7c75ce83.

Cantina Managed: Fix verified.

3.5.15 Order of parameters `assetsToShares()` and `sharesToAssets()`

Severity: Informational

Context: kpkShares.sol#L493, kpkShares.sol#L512

Description: The functions `assetsToShares()` and `sharesToAssets()` are similar, but the order of the parameters is different. This make the code more difficult to read and maintain.

Recommendation: Consider using the same ordering of parameters.

kpk: Fixed in commit fff0d26e.

Cantina Managed: Fix verified.

3.5.16 Different approach to validate authorization in cancel functions

Severity: Informational

Context: kpkShares.sol#L256-L272, kpkShares.sol#L351-L366, kpkShares.sol#L636-L640

Description: Function `cancelSubscription()` uses `_validateCancellationAuthorization()`, while function `cancelRedemption()` uses an explicit check. This make the code more difficult to maintain and review.

Recommendation: Consider using `_validateCancellationAuthorization()` everywhere:

```
  function cancelRedemption(uint256 id) external {
      // ...
-     if (request.investor != msg.sender && request.receiver != msg.sender) revert
 ↪  NotAuthorized();
+     _validateCancellationAuthorization(request.investor, request.receiver);
      // ...
  }
```

kpk: Fixed in commit d68f21c0.

Cantina Managed: Fix verified.

3.5.17 Different approaches to check `RequestStatus`

Severity: Informational

Context: kpkShares.sol#L256-L262, kpkShares.sol#L351-L357, kpkShares.sol#L646-L649, kpkShares.sol#L722-L728

Description: Function `cancelSubscription()` checks `request.timestamp == 0` to check the record exists. However function `cancelRedemption()` does a similar check via `request.investor == address(0)`. Additionally function `_processApproved()` does a similar check in `_validateRequest()`. The different checks make maintaining and checking the code more difficult.

Recommendation: Consider using the same checks everywhere. This can be done via `_validateRequest()`, that also checks `RequestStatus`, which is also done in `cancelSubscription()` and `cancelRedemption()`.

```
-  if (request.timestamp == 0) revert UnknownRequest(); // in case of cancelSubscription
-  if (request.investor == address(0)) revert NoPendingRedeemRequest(); // in case of
 ↪  cancelRedemption
-  if (request.requestStatus != RequestStatus.PENDING) {
+  if (!_validateRequest()) {
       revert RequestNotPending();
   }
```

kpk: Fixed in commit 7976f634.

Cantina Managed: Fix verified.

3.5.18 `previewSubscription` returns redundant info

Severity: Informational

Context: kpkShares.sol#L186-L210, kpkShares.sol#L290-L311

Description: The functions `previewSubscription()` and `previewRedemption()` return an entire struct, however only one field of the `struct` contains useful information. Changing this simplifies the function as well as the documentation and allows for easier integration from other contracts.

Recommendation: Consider only returning the following:

- `previewSubscription()` : return `shares`;
- `previewRedemption()` : return `assets`.

kpk: Fixed in commit fbcfe5b9.

Cantina Managed: Fix verified.

3.5.19 Missing dual timeouts allows cancel/approve race and no user-set expiry

Severity: Informational

Context: kpkShares.sol#L268-L270

Description: The current `subscriptionRequestTtl` only defines when a requester is allowed to cancel; it does not make the request expire. After TTL has passed, an operator can still approve while a user is attempting to cancel, creating a small race at the boundary. Users also lack a way to set (or rely on) a fixed maximum lifetime for their requests.

Recommendation: Consider to introduce an additional expiry deadline per request while keeping the existing TTL:

- On creation, set `expiryAt = block.timestamp + maxLifetime` (configurable/governable).
- Approval paths should require `block.timestamp <= expiryAt`; otherwise reject or auto-cancel.
- Keep `subscriptionRequestTtl` as the minimum cancel delay (i.e., users may cancel only after TTL).
- Optionally, have batch processing report/emit when a request was skipped due to cancel/expiry to aid monitoring.
- Mirror the same dual-timeout scheme for redemption requests and document both cancelableFrom and expiryAt in events/docs.

kpk: Fixed in commit 93430689.

Cantina Managed: Fix verified.

3.5.20 Checks missing in the `preview` functions

Severity: Informational

Context: kpkShares.sol#L186, kpkShares.sol#L213-L217, kpkShares.sol#L290-L293, kpkShares.sol#L314-L318, kpkShares.sol#L629-L631

Description: The `preview` functions check less than then `request` functions. This way the `preview` functions give different results than the `request` functions.

Recommendation: Consider also calling `_validateRequestParams()` in `previewSubscription()` and `previewRedemption()`.

kpk: Acknowledged.

Cantina Managed: Acknowledged.

3.5.21 Use of USDC is not fully documented

Severity: Informational

Context: kpkShares.sol#L127, kpkShares.sol#L139-L140, kpkShares.sol#L596-L597, kpkShares.sol#L596-L598, kpkShares.sol#L852

Description: According to the comment at `_initializeState()`, the base asset should be `USDC`. However this is not documented elsewhere.

The asset certainly has to be a stablecoin because `_updateAsset(..., true, ...)` is called with `isUsd == true`.

Recommendation: Make use of `USDC` or USD stablecoin explicit in the comments and in the `README.md`.

kpk: Fixed in commit a07bb464.

Cantina Managed: Fix verified.

3.5.22 `SECONDS_PER_YEAR` doesn't take leap years into account

Severity: Informational

Context: kpkShares.sol#L51-L52

Description: The constant `SECONDS_PER_YEAR` doesn't take leap years into account. A more accurate approximation could be used.

Recommendation: Consider using `365.2425` days per year:

```
- uint256 public constant SECONDS_PER_YEAR = 365 days;
+ uint256 public constant SECONDS_PER_YEAR = 365.2425 days;
```

kpk: Acknowledged. We prefer to keep the number as an integer to make apy calculations easier on the client side.

Cantina Managed: Acknowledged.

3.5.23 Incompatibility with rebasing and fee-on-transfer tokens in escrow/recovery accounting

Severity: Informational

Context: kpkShares.sol#L555-L557

Description: The function `_assetRecoverableAmount` derives recoverable value as `balanceOf(this) - subscriptionAssets[token]`. This assumes balances only change via explicit transfers and that escrow equals the nominal amounts recorded in `subscriptionAssets`. With rebasing tokens (e.g., sUSDe), the contract balance can increase/decrease without transfers, making "recoverable" drift away from true free funds (e.g., sweeping positive rebase that should remain backing requests, or creating deficits). With fee-on-transfer tokens, the recorded escrow may exceed the actual received amount, leading to later approval/cancel paths attempting to move more tokens than held and causing reverts; the subtraction can also revert due to underflow when computing "recoverable".

Recommendation: Consider disallowing rebasing/FoT assets at the allowlist level, or wrapping them (e.g., non-rebasing ERC4626 wrapper) before use.

kpk: Fixed in commit 3e4e054b.

Cantina Managed: Fix verified.

3.5.24 Parameter semantic mismatch between interface/docs and implementation (price vs `minSharesOut`)

Severity: Informational

Context: IkpkShares.sol#L334

Description: The function `requestSubscription` from contract `IkpkShares` documents its second parameter as `sharesPrice` ("current price per share"), while the implementation treats that position as `minSharesOut`. The function names/parameter names also diverge between interface and implementation (same pattern applies to redemption). This inconsistency will cause integrators to pass a price where a minimum output is expected, leading to incorrect slippage checks, unexpected reverts, or unintended acceptances.

Recommendation: Consider to align semantics and naming across interface, implementation, and events:

- Option A: keep minSharesOut as the second parameter, rename it in the interface/docs, and update param names for clarity.
- Option B: accept sharesPrice and compute `sharesOut` internally; enforce `sharesOut >= minSharesOut` supplied separately.
- In either case, version the interface, add unit tests for parameter order/units, and update integrator docs/examples.

kpk: Fixed in commit c5f70390.

Cantina Managed: Fix verified.

3.5.25 Magic number used for BPS denominator in redemption fee calculation

Severity: Informational

Context: kpkShares.sol#L810

Description: The redemption fee uses a hard-coded `10_000` as the basis-points denominator. Using a magic number reduces readability and can lead to inconsistencies if the denominator ever needs to change or if other fee paths use a different literal.

Recommendation: Consider to introduce a single named constant (e.g., `BPS_DENOMINATOR = 10_000`) and use it across all fee calculations. Consider to also clamp fee inputs to `<= BPS_DENOMINATOR` to keep checks consistent.

kpk: Fixed in commit bebed253.

Cantina Managed: Fix verified.

3.5.26 `redemptionFee` documented in assets but emitted in shares

Severity: Informational

Context: IkpkShares.sol#L202

Description: The interface comment states that `redemptionFee` is "the amount of assets charged as redemption fee," but the implementation emits/handles this amount in shares. This doc/semantics mismatch can mislead integrators and analytics, producing incorrect UI/accounting when interpreting the event.

Recommendation: Consider to align the documentation and code:

- If the implementation stays in shares, consider to rename/document the field as `redemptionFeeShares` and clarify units across events and helpers.
- If assets are preferred, consider to convert the fee to assets at the approval price and emit the asset-denominated amount for consistency with user-facing semantics.

kpk: Fixed in commit 4f5fe6a6.

Cantina Managed: Fix verified.

3.5.27 TTL updates apply to existing requests, contradicting docs

Severity: Informational

Context: IkpkShares.sol#L396-L402

Description: The interface comments state that `setSubscriptionRequestTtl` and `setRedemptionRequestTtl` apply "for new requests," but the implementation uses the current TTL during validations, so updating these values effectively changes the behavior for existing pending requests as well.

Recommendation: Consider:

- Updating the docs to reflect the current global behavior, or.
- Snapshoting the TTL at request creation (store per-request TTL) and use that snapshot in all subsequent validations and event data, so updates only affect future requests.

kpk: Fixed in commit ce3aedc2.

Cantina Managed: Fix verified.

3.5.28 Unused global state variables / constants

Severity: Informational

Context: kpkShares.sol#L43, kpkShares.sol#L89, kpkShares.sol#L508, kpkShares.sol#L528

Description: Both the public state variable `updateRequestTtl` and the constant `_NORMALIZED_PRECISION_USD` are declared but not referenced elsewhere. Unused declarations

add maintenance noise, expand the upgrade surface, and can suggest features (request TTL updates, USD normalization) that are not implemented.

Recommendation: Consider to remove `updateRequestTtl`. Consider using `_NORMALIZED_PRECISION_USD` where the value `1e8` is used.

kpk: Fixed in commit 4a2ac96e.

Cantina Managed: Fix verified.