CANTINA

Sky: Grove Basin

Security Review

Cantina Managed review by:
Christoph Michel, Lead Security Researcher
Xmxanuel, Lead Security Researcher

May 11, 2026

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
3.1 Medium Risk . . . 4
3.1.1 Deployment scripts use incorrect `creditToken` and `PSM` addresses . . . 4
3.2 Low Risk . . . 4
3.2.1 LP total asset loss if PSM fees are enabled . . . 4
3.2.2 Use asset decimals for `convertedAmount` in `LiquidityDrawn` event . . . 4
3.2.3 Owner role permissions do not match spec for role management . . . 5
3.2.4 `maxSwapSize` limit is easy to circumvent . . . 5
3.2.5 Standardize ERC20 interactions around SafeERC20 usage . . . 6
3.2.6 Enforce USDS as swap token for `UsdsUsdcPocket` . . . 6
3.2.7 `BUIDLTokenRedeemer.completeRedeem` can be called before settlement resulting in a loss . . . 6
3.2.8 Disallow setting fee equal to BPS . . . 7
3.2.9 `pendingCreditTokenBalance` can overstate `totalAssets` . . . 7
3.2.10 `collateralToken` deficit branch in `_withdrawLiquidityInPocket` is unreachable from `withdraw` . . . 7
3.3 Informational . . . 8
3.3.1 Avoid using `msg.sender` as deployer in deployment scripts . . . 8
3.3.2 BUIDL basin usage depends on allowlisting the Basin, redeemer, and swappers . . . 8
3.3.3 Validate Chronicle oracle decimals instead of assuming precision . . . 8
3.3.4 Handle USDS asset explicitly and revert invalid assets in `UsdsUsdcPocket.withdrawLiquidity` . . . 9
3.3.5 Simplify USDC custody behavior in `UsdsUsdcPocket` . . . 9
3.3.6 LP deposits can fail when pocket auto-deposits into capped vaults . . . 10
3.3.7 Pause Basin before updating rate provider to prevent sandwiching . . . 10
3.3.8 Enforce minimum initial deposit in `GroveBasin` itself . . . 10
3.3.9 Malicious redeemer can still block its removal with try / catch . . . 10
3.3.10 `MANAGER_ADMIN_ROLE` can control fees indirectly via bounds clamping . . . 11
3.3.11 `JTRSYTokenRedeemer` only supports full redemptions - no partial redeem or cancellation . . . 11
3.3.12 Minor issues, Typos & Documentation . . . 12

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

Sky Protocol is a decentralised protocol developed around the USDS stablecoin.

From Apr 6th to Apr 26th the Cantina team conducted a review of grove-basin on commit hash c293e805. The team identified a total of **23** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	10	6	4
Gas Optimizations	0	0	0
Informational	12	7	5
Total	**23**	**14**	**9**

The Cantina Managed team reviewed Sky's grove-basin holistically on commit hash a79269a3 and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Medium Risk

3.1.1 Deployment scripts use incorrect `creditToken` and `PSM` addresses

Severity: Medium Risk

Context: SetupJTRSYUsdsUsdcBasin.s.sol#L45, SetupJTRSYUsdsUsdcBasin.s.sol#L57, SetupJTRSYMorphoUsdtBasin.s.sol#L50

Description:

1. Both deployment scripts pass `Ethereum.SUSDS` as `creditToken`, but the redeemer is `JTRSYTokenRedeemer` targeting the Centrifuge JTRSY vault. The credit token should be the JTRSY share token (`0x8c213ee79581Ff4984583C6a801e5263418C4b86` , 6 decimals).
2. `SetupJTRSYUsdsUsdcBasin` uses `Ethereum.PSM` (`0xf6e72Db5454dd049d0788e411b06CfAF16853042`) for the `UsdsUsdcPocket` constructor. This is the raw LitePSM which produces DAI from `sellGem`, not USDS. The fork test at `test/fork/UsdsUsdcPocketForkTest.sol:42` correctly uses the `UsdsPsmWrapper` at `0xA188EEC8F81263234dA3622A406892F3D630f98c` with the comment `"Actual USDS PSM Wrapper on mainnet"` .

The incorrect Lite `PSM` address can lead to locked funds. If the manager would call `UsdsUsdcPocket.depositLiquidity(usdc)` directly, the USDC would be swapped to `Dai` and there would be no way to get the Dai out of the contract.

Recommendation:

1. Replace `Ethereum.SUSDS` with the JTRSY share token address in both deployment scripts and redeemer constructor calls.
2. Replace `Ethereum.PSM` with the `UsdsPsmWrapper` address (`0xA188EEC8F81263234dA3622A406892F3D630f98c`) in `SetupJTRSYUsdsUsdcBasin` .

Sky: Fixed in 0973b7d.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 LP total asset loss if PSM fees are enabled

Severity: Low Risk

Context: UsdsUsdcPocket.sol#L106

Description: The `UsdsUsdcPocket.withdrawLiquidity(USDC)` path assumes fee-free conversions for `PSM.buyGem` . If Basin remains fee-free while the PSM introduces a fee, Basin's swap calls invoke `withdrawLiquidity(deficit, USDC)` and route through this `buyGem` path, so repeated USDS <> USDC cycling realize PSM fees and erode LP USDS inventory at no cost for the attacker.

Recommendation: Consider reverting this `UsdsUsdcPocket.withdrawLiquidity(USDC)` path whenever the relevant PSM fee (`tout`) is non-zero. This would prevent eroding LP value while still allowing migration to a new pocket via `setPocket` , since `setPocket` does not rely on the USDC withdrawal path (and the pocket should never contain USDC after the suggested cleanup in another issue and can be `skim` med).

Sky: Fixed in commit cc317a72.

Cantina Managed: Fix verified.

3.2.2 Use asset decimals for `convertedAmount` in `LiquidityDrawn` event

Severity: Low Risk

Context: UsdsUsdcPocket.sol#L106

Description: For `UsdsUsdcPocket`, `convertedAmount` is emitted in `usdsInwad` units (18 decimals) for the `asset == USDC` path, but the `LiquidityDrawn` event is expected to report the amount in the asset that is withdrawn (USDC, 6 decimals).

```
/**
 *  @dev   Emitted when liquidity is withdrawn from the pocket.
 *  @param asset           Address of the asset withdrawn.
 *  @param amount          Amount of the asset requested.
 *  @param convertedAmount Amount converted from another asset to fulfill the withdrawal.
 */
event LiquidityDrawn(address indexed asset, uint256 amount, uint256 convertedAmount);
```

This would be consistent with other event usage in the other pockets.

Recommendation: Update the event emission so `convertedAmount` is reported in `asset`.

```
IPSMLike(psm).buyGem(address(this), remainder);
convertedAmount = remainder;
```

Sky: Fixed in commit be156a2.

Cantina Managed: Fix verified.

3.2.3 Owner role permissions do not match spec for role management

Severity: Low Risk

Context: GroveBasin.sol#L158

Description: The spec states:

> "Top-level admin. Can grant and revoke all other roles. Can set purchase fee, redemption fee."

However, the current implementation does not allow `owner` to grant/revoke all other roles because `MANAGER_ADMIN_ROLE` is the admin for the following four roles:

```
_setRoleAdmin(MANAGER_ROLE,           MANAGER_ADMIN_ROLE);
_setRoleAdmin(PAUSER_ROLE,            MANAGER_ADMIN_ROLE);
_setRoleAdmin(REDEEMER_CONTRACT_ROLE, MANAGER_ADMIN_ROLE);
_setRoleAdmin(REDEEMER_ROLE,          MANAGER_ADMIN_ROLE);
```

Recommendation: Either grant `owner` also the `MANAGER_ADMIN_ROLE` so it can administer all roles, or revise the spec to reflect that these permissions belong to `MANAGER_ADMIN_ROLE` instead of `owner`.

Sky: Acknowledged. Will revise the spec. The intention of the spec was that owner can assume `MANAGER_ADMIN_ROLE` for themselves to control the downstream roles.

Cantina Managed: Acknowledged.

3.2.4 `maxSwapSize` limit is easy to circumvent

Severity: Low Risk

Context: GroveBasin.sol#L416

Description: The `maxSwapSize` check can be bypassed by splitting a large swap into multiple smaller calls. As implemented, it does not meaningfully restrict total swap volume and its intended protection is unclear if repeated calls are allowed.

Recommendation: Clarify the security objective of `maxSwapSize` and either remove it if it provides no real protection, or replace it with an enforcement mechanism that cannot be trivially bypassed through repeated transactions (for example, rate limits or cumulative accounting over a time window).

Sky: Acknowledged. We'll keep it as it is and add rate limits in the next iteration of the product. For now, the implied rate limit would be the amount of liquidity we provide.

Cantina Managed: Acknowledged.

3.2.5 Standardize ERC20 interactions around SafeERC20 usage

Severity: Low Risk

Context: GroveBasin.sol#L955

Description: Not all ERC20 interactions consistently use the `SafeERC20` `safe*` helpers, even though some paths already do so to handle non-standard tokens such as USDT.

Recommendation: Use `SafeERC20` wrappers consistently for all ERC20 approvals and transfers across the contract, or explicitly document which of the three supported tokens is allowed to be USDT-like/non-standard.

Sky: Fixed in commit a1eba7b.

Cantina Managed: Fix verified.

3.2.6 Enforce USDS as swap token for `UsdsUsdcPocket`

Severity: Low Risk

Context: GroveBasin.sol#L271

Description: There is an implicit assumption on `UsdsUsdcPocket` that the `swapToken == USDS` and `creditToken == USDC`, not the other way.

Otherwise, this leads to issues:

- `Basin._withdrawLiquidityInPocket`: `else if (asset == collateralToken)` branch attempts to withdraw USDC here, attempting to withdraw USDS here does not do anything.
- Can block `setPocket`: Its `_getAvailableBalance(swapToken)` call can overestimate the assets that can actually be withdrawn from the pocket. For USDC, it returns `usdc.balanceOf(address(this)) + usds.balanceOf(address(this)) * _usdcPrecision / _usdsPrecision`. If the PSM has a `tout` fee, buying `usds.balanceOf(address(this)) * _usdcPrecision / _usdsPrecision` of USDC costs more than the USDS balance and would revert.

Recommendation: Enforce that `UsdsUsdcPocket` may only be configured with `swapToken = USDS` and `collateralToken = USDC`, for example, by checking `basin.collateralToken() == usdc && basin.swapToken() == usds` in its constructor.

Sky: Fixed in commit 10aefca.

Cantina Managed: Fix verified.

3.2.7 `BUIDLTokenRedeemer.completeRedeem` can be called before settlement resulting in a loss

Severity: Low Risk

Context: BUIDLTokenRedeemer.sol#L73-L76, GroveBasin.sol#L962-L975

Description: Unlike `JTRSYTokenRedeemer` where the `vault.redeem` reverts if not yet fulfilled, the `BUIDLTokenRedeemer` has no guard against premature completion. `completeRedeem` reads `Math.min(request.collateralTokenAmount, IERC20(collateralToken).balanceOf(address(this)))`. If called before the issuer has sent USDC back, `balanceOf` is 0, so `collateralTokenReturned = 0`.

The basin's `_completeRedeem` then deletes the request and decrements `pendingCreditTokenBalance` by the full `creditTokenAmount`.

This results in a complete loss of the redeemed credit tokens with zero collateral returned.

Recommendation: The risk of a unintended too early call should be documented in case this feature is needed to write-off requests which got never fulfilled. Otherwise, consider not allowing a too early `completeRedeem` call.

Sky: Fixed in commit e17e4dc. We assume the manager only completes redemptions once the entire request has been filled.

Cantina Managed: Fix verified. Note that a manager-admin `sweep` function for `credit` / `collateral` tokens was also added, introducing an additional trust assumption that the manager admin does not call `sweep` before a legitimate `completeRedeem`.

3.2.8 Disallow setting fee equal to BPS

Severity: Low Risk

Context: GroveBasin.sol#L986, GroveBasin.sol#L240

Description: The logic at this call site reverts when `fee == BPS`, but the contract currently allows that value to be configured. This creates an invalid configuration that can later cause execution to fail.

Recommendation: Add validation to reject `fee >= BPS` when setting or updating the fee bounds in `setFeeBounds`.

```
- if (newMaxFee > BPS)        revert MaxFeeExceedsBps();
+ if (newMaxFee >= BPS)        revert MaxFeeExceedsBps();
```

Sky: Fixed in commit 2dcab3e.

Cantina Managed: Fix verified.

3.2.9 `pendingCreditTokenBalance` can overstate `totalAssets`

Severity: Low Risk

Context: GroveBasin.sol#L674-L678

Description: `totalAssets()` values `pendingCreditTokenBalance` at the current oracle rate, but the actual settlement price is determined by the vault operator at fulfillment time (for example the Centrifuge pool operator for JTRSY). If `completeRedeem` is delayed and the credit token price rises continuously, `totalAssets` overstates the basin's true value and inflating share prices until the actual collateral is received.

In addition, there can be redemption fees (like for BUIDL) that are ignored in the `pendingCreditTokenBalance`, overstating the `totalAssets()` again.

Since only the `liquidityProvider` can deposit, this is not externally exploitable. The risk is an internal accounting inaccuracy that grows with time between `initiateRedeem` and `completeRedeem`.

Recommendation: Ensure operational procedures complete redemptions promptly to minimize the gap between oracle rate and fulfillment price.

Sky: Acknowledged. note that TBills currently appreciate slowly, which mitigates the impact of a `pendingRedemption`.

Cantina Managed: Acknowledged.

3.2.10 `collateralToken` deficit branch in `_withdrawLiquidityInPocket` is unreachable from `withdraw`

Severity: Low Risk

Context: GroveBasin.sol#L865

Description: The `if (basinBalance < amount)` condition in the `collateralToken` branch of `_withdrawLiquidityInPocket` is always false for the `withdraw` flow.

The problem is that `previewWithdraw` uses `_getAvailableBalance` which only calls the pocket for the `swapToken`, not for the `collateralToken`. This means for a `collateralToken` withdrawal, the `amount` will always be equal to (or less than) the collateral token balance of the Basin and doesn't consider additional `collateralToken` liquidity in the pocket.

For the swap functions, additional needed liquidity is requested from the pocket the deficit branch fires correctly since the actual amount is based on the parameter input.

The LP doesn't lose value in the withdraw case (shares are burned proportionally to the partial withdrawal), but they can't extract their full value in a single call for a specific token when the basin's direct balance is insufficient.

Recommendation: Consider supporting the `withdraw` case as well with additional pocket `collateralToken` liquidity. This means the `collateralToken` balance needs to be considered in `_getAvailableBalance`.

Sky: Acknowledged. The asymmetry between swap and withdraw is intentional. In practice the LP holds most of its stablecoin liquidity in USDS, most users will want to swap to USDC, and the LP is not expected to need to withdraw more USDC than is already held by the Basin.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Avoid using `msg.sender` as deployer in deployment scripts

Severity: Informational

Context: SetupJTRSYUsdsUsdcBasin.s.sol#L41

Description: The script relies on `msg.sender` to identify the deployer and grants it the `owner` role, which is fragile in Forge scripts because it only matches the intended deployer if `--sender <deployer>` is explicitly provided. Using `--account` alone does currently **not** set `msg.sender`, so the script can accidentally use Forge's default sender and produce inconsistent behavior across script invocations.

Recommendation: Consider using an explicit deployer address by hardcoding it where appropriate or reading it from an environment variable.

Sky: Fixed in commit 1896db1d.

Cantina Managed: Fix verified.

3.3.2 BUIDL basin usage depends on allowlisting the Basin, redeemer, and swappers

Severity: Informational

Context: BUIDLTokenRedeemer.sol#L66

Description: BUIDL basin operation depends on BUIDL issuer allowlisting both the `Basin` contract (as custodian) and this redeemer (as intermediary because of `initiateRedeem`'s `transferFrom(basin, this)`), and any swapper interacting with the basin must also be allowlisted or swaps will revert.

Recommendation: Ensure both Basin and the redeemer contract are allowlisted.

Sky: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Validate Chronicle oracle decimals instead of assuming precision

Severity: Informational

Context: ChronicleRateProvider.sol#L20

Description: The rate provider currently assumes a fixed `CHRONICLE_PRECISION` value. This can silently break if the oracle uses a different decimals configuration.

Recommendation: Avoid hardcoding the precision assumption. Either derive `CHRONICLE_PRECISION` from `10 ** oracle.decimals()` or explicitly assert during initialization that `oracle.decimals() == 18` and revert otherwise.

Sky: Though Chronicle Scribes expose decimals, the Chronicle Proof of Asset oracles do not. From Chronicle's documentation, it seems they will always use 18, and rate providers are governance gated, which adds an additional level of protection.

Cantina Managed: Acknowledged.

3.3.4 Handle USDS asset explicitly and revert invalid assets in `UsdsUsdcPocket.withdrawLiquidity`

Severity: Informational

Context: UsdsUsdcPocket.sol#L87-L118

Description: The asset handling in `UsdsUsdcPocket` should match the pattern used in the other pockets.

1. For `depositLiquidity`, any unsupported asset should revert with `InvalidAsset()` rather than falling through implicit behavior.
2. For `withdrawLiquidity`, when `asset == usds`, the function should explicitly return `amount`. Any unsupported asset should revert with `InvalidAsset()` rather than falling through implicit behavior.

Recommendation: Update the conditional logic to explicitly handle USDS and revert otherwise, e.g.:

```
else if (asset == address(usds)) {
    return amount;
} else revert InvalidAsset();
```

Sky: Fixed in commit e56ad40.

Cantina Managed: Fix verified.

3.3.5 Simplify USDC custody behavior in `UsdsUsdcPocket`

Severity: Informational

Context: UsdsUsdcPocket.sol#L13

Description: The current `UsdsUsdcPocket` pocket behavior is difficult to reason about because it differs from standard pocket behavior of only having custody of the swap token - here, it can have custody of both swap token (USDS) and collateral token (USDC). In practice, USDC is usually only transiently present during Basin flows, however, a manager-triggered `Pocket.withdrawLiquidity(USDC)` can leave USDC stranded in the pocket.

This creates asymmetry in accounting and custody semantics: `availableBalance(USDS)` does not reflect any USDC balance, while `availableBalance(USDC)` includes both USDC and USDS balances. This also contributes to edge cases such as USDC being left behind after `setPocket` transitions.

Recommendation: Clean up the design by enforcing a clearer invariant for the USDS/USDC configuration: the pocket should custody only the swap token, and `collateralToken` should not remain stored in the pocket after all possible manager or Basin operations.

1. As part of that, consider making `withdrawLiquidity(USDC)` forward USDC directly to the Basin instead of retaining it in the pocket. (This requires the removal of the `transferFrom` in `Basin`'s withdraw. This resolves the issue of manual manager calls leaving the USDC stranded in the pocket.
2. The only way `collateralToken`s can appear in the pocket then is from donations or flows outside of Basin. Consider adding a `basinOrManagerOnly` `sweep` function that transfers any `collateralToken` balance to the basin.
3. Also consider cleaning up `availableBalance` for the `asset = USDC` special case returning `usdc.balanceOf(address(this)) + usds.balanceOf(address(this)) * _usdcPrecision / _usdsPrecision`, it is currently never called from `Basin`, so it's purely an off-chain functionality. If it is not used, consider removing the special case.

Sky: Fixed in commit cfb099d.

Cantina Managed: Fix verified.

3.3.6 LP deposits can fail when pocket auto-deposits into capped vaults

Severity: Informational

Context: AaveV3UsdtPocket.sol#L69

Description: Basin LP deposits currently attempt to automatically allocate the swap token into the pocket vault. If the underlying vault cannot accept additional supply, such as when an Aave market supply cap is reached, `pocket.depositLiquidity` can revert and cause the entire LP deposit to fail.

Recommendation: Decide whether this behavior is acceptable as a business/product constraint. If not, avoid auto-depositing during LP deposits and leave funds idle in the pocket for the manager to allocate separately when vault capacity is available.

Sky: Fixed in commit bf82f9c0.

Cantina Managed: Fix verified, it's using `try` / `catch` for the deposit call now.

3.3.7 Pause Basin before updating rate provider to prevent sandwiching

Severity: Informational

Context: GroveBasin.sol#L166

Description: Updating the rate provider while the `Basin` remains active can expose LPs to sandwich attacks if the new provider returns materially different rates than the old one. Attackers could trade around the update and extract profit from the sudden rate provider pricing change.

Recommendation: Consider pausing the Basin before changing rate providers.

Sky: Acknowledged, will document.

Cantina Managed: Acknowledged.

3.3.8 Enforce minimum initial deposit in `GroveBasin` itself

Severity: Informational

Context: GroveBasin.sol#L475

Description: `assetsToDeposit` is currently not validated in `GroveBasin` itself to ensure the initial deposit is at least `10 ** asset.decimals()`. Although this check exists in the deployer factory, direct usage of `GroveBasin` can bypass that guard, allowing an unsafe initialization that does not prevent the first depositor issue, such as depositing 1 wei and minting 1 share.

Recommendation: Consider adding the same minimum initial deposit validation in `GroveBasin` so all initialization paths enforce `assetsToDeposit >= 10 ** asset.decimals()`, not just deployments routed through the factory.

Sky: Fixed in commit 5e6466d.

Cantina Managed: Fix verified.

3.3.9 Malicious redeemer can still block its removal with try / catch

Severity: Informational

Context: GroveBasin.sol#L301

Context: GroveBasin.sol#L301

Description: When removing a redeemer via `Basin.removeTokenRedeemer`, it performs a last cleanup call on the redeemer with a try/catch:

```
try ITokenRedeemer(redeemer).tearDown(address(this)) {} catch {}
```

The intention of using `try` / `catch` here is to prevent a malicious redeemer from blocking its own removal. However, the redeemer is currently invoked with 63/64 of the remaining gas which can be fully consumed by them. It might happen that the final `_revokeRole()` in Basin runs out of gas.

Note that a removal is not strictly necessary for the Basin operations, as multiple redeemers can be added, so the impact of a `removeTokenRedeemer` failing is low.

Recommendation: Consider capping the gas the redeemer can consume in `tearDown` to some max gas value.

```
(bool,) = redeemer.call{gas: GAS_STIPEND}(
   abi.encodeCall(ITokenRedeemer.tearDown, (address(this)))
);
```

Sky: Acknowledged. The try-catch wasn't necessarily to prevent a malicious redeemer from blocking its own removal. Given our trust assumptions and because the `REDEEMER_CONTRACT` can only be set by governance (`MANAGER_ADMIN`) or the issuer timelock (`ISSUER_ROLE` , via `MANAGER_ADMIN`), we think this is acceptable to leave as is.

Cantina Managed: Acknowledged.

3.3.10 `MANAGER_ADMIN_ROLE` can control fees indirectly via bounds clamping

Severity: Informational

Context: GroveBasin.sol#L238-L263

Description: The role hierarchy designates `OWNER_ROLE` (timelock) for fee control and `MANAGER_ADMIN_ROLE` (governance) for bounds control. However, `setFeeBounds` clamps current fees to the new bounds:

```
if (redemptionFee_ < newMinFee) {
    _setRedemptionFee(newMinFee);
} else if (redemptionFee_ > newMaxFee) {
    _setRedemptionFee(newMaxFee);
}
```

By calling `setFeeBounds(x, x)` , `MANAGER_ADMIN_ROLE` can force both `purchaseFee` and `redemptionFee` to exactly `x` , bypassing `OWNER_ROLE` -only `setPurchaseFee` / `setRedemptionFee` .

Recommendation: Accept that governance has effective fee control and document it or change `setFeeBounds` to revert if clamping would be needed, requiring `OWNER_ROLE` to first adjust fees within the new bounds.

Sky: Fixed in commit 21b5668.

Cantina Managed: Fix verified, `setFeeBounds` no longer changes the actual fees, instead, it is verified that the current fee is within the new bounds.

3.3.11 `JTRSYTokenRedeemer` only supports full redemptions - no partial redeem or cancellation

Severity: Informational

Context: JTRSYTokenRedeemer.sol#L76-L87

Description: `JTRSYTokenRedeemer.completeRedeem` calls `vault.redeem(request.creditTokenAmount, ...)` with the full request amount. Centrifuge's `AsyncRequestManager.redeem` enforces `require(shares <= maxRedeem(vault_, controller), ExceedsMaxRedeem())` , where `maxRedeem` returns only the fulfilled, claimable portion and not the total pending request. (See: Centrifuge AsyncRequestManager.sol#L396) This means:

1. **No partial redemptions:** If the Centrifuge pool operator has fulfilled 40 out of 100 requested shares, `completeRedeem` reverts because `100 > maxRedeem(controller) = 40` . The Basin must wait until 100% is fulfilled before it can complete. For large `redeemRequest` amounts, this means longer wait times.
2. **No cancellation path:** The Centrifuge vault supports cancelling pending redeem requests before fulfillment.

Recommendation: Consider adding support for partial redeemptions and cancellations for the Centrifuge JTRSY vault.

Sky: Added a sweep function in commit a92a72a2 controlled by the `MANAGER_ADMIN` role. Redemptions are assumed to always be filled completely (not partially) by JTRSY.

Cantina Managed: Fix verified. In addition, `JTRSYTokenRedeemer` now also verifies that only one redemption is active at a time.

3.3.12 Minor issues, Typos & Documentation

Severity: Informational

Context: (*See each case below*)

Description:

1. JTRSYTokenRedeemer.sol#L61-L62: Consider adding `vault.share() == creditToken_` as a constructor check, mirroring the existing `collateralToken != vault.asset()` validation.
2. UsdsUsdcPocket.sol#L62-L64: Consider adding a sanity check to verify that `usdc != usds`, similar to the token checks in the Basin constructor. This could also be achieved by verifying that `basin.swapToken()` equals `usds` and `basin.collateralToken()` equals `usdc`, or by verifying the params against the `psm`.
3. GroveBasin.sol#L549-L552: `previewWithdraw` has an explicit `_requireValidAsset(asset)` check, but `previewDeposit` relies on the implicit check inside `_getAssetValue`. Consider adding it to `previewDeposit` for consistency, or removing it from `previewWithdraw`.
4. README.md#L36: The README references `deploy/GroveBasinDeploy.sol` for the first depositor attack but this file does not exist. The initial deposit is now done in the factory.
5. GroveBasin.sol#L799-L800: Consider changing the two `InvalidSwap()` checks to: `if !(asset == creditToken || quoteAsset == creditToken) revert InvalidSwap()` to capture the intention better: After verifying that both are valid assets and they are not the same, the swap must involve the credit token.
6. MorphoUsdtPocket.sol#L69-L87: In the `setPocket` migration case, the pocket can end up with dust vault share tokens in the Morpho vault. This is because `availableBalance` calls `convertToAssets` with floor rounding, and the migration calls `vault.withdraw(underReported)` which burns fewer shares than the total. For a clean migration, calling `vault.redeem(vault.balanceOf(pocket))` instead of `vault.withdraw(availableBalance)` would burn all shares and leave no dust.
7. GroveBasin.sol#L579: The `previewSwap*` functions currently enforce conditions such as max swap size, so they are not acting as purely calculation-only helpers. Given that they already validate operational constraints, they should also consider whether swaps are paused to avoid returning previews for actions that cannot currently be executed.
8. SetupJTRSYUsdsUsdcBasin.s.sol#L41: Deploy script sets the deployer (`msg.sender`) as `owner`. Requires additional post-deployment steps to transfer `OWNER_ROLE` to the timelock and `MANAGER_ADMIN_ROLE` to the governance. **Recommendation:** Consider adding the constructor validations and other suggestions mentioned above.
9. GroveBasin.sol#L913: The code reaches a branch that should be unreachable under normal swap flow. Consider reverting here with `InvalidSwap()` instead of returning the global pause key.
10. GroveBasin.sol#L949: `request.creditTokenAmount > 0` is later used to indicate if a request has been initiated or not. However, one can initiate requests with a credit token amount of 0. Disallow this by checking `creditTokenAmount > 0` when creating new requests in `_initiateRedeem`.
11. GroveBasin.sol#L471: The spec indicates that only "`Withdrawals` and `completeRedeem` cannot be paused.". Consider checking for global pause also in `depositInitial`.

12. MorphoUsdtPocket.sol, AaveV3UsdtPocket.sol: The `MorphoPocket` and the `AavePocket` are actually generic and not specific for USDT. USDT specific is only the needed approve zero first but this doesn't limit other ERC20s.
13. AaveV3UsdtPocket.sol#L53: Consider validating that the A-Token matches by checking `aUsdt.UNDERLYING_ASSET_ADDRESS() == usdt`.
14. GroveBasinFactory.sol#L63: One can still precompute the create2 deployment address and frontrun it with a donation. Instead of calling `previewDeposit` in `depositInitial` (which looks at `totalAssets()` already), consider setting `newShares = _getAssetValue(asset, assetsToDeposit, false)`, basically treating `totalAssets() == 0` for `previewDeposit` → `convertToShares` for the initial call.

Sky:

- 1. Addressed in commit 8ce517d.
- 3. Addressed in commit 14508ad.
- 4. Addressed in commit 01b1a7a.
- 5. Addressed in commit 1487844.
- 7. Addressed in commit 554b5c5.
- 10. Addressed in commit d15f018.
- 11. Addressed in commit 179fb5a.
- 13. Addressed in commit ee0828c.

Rest is acknowledged.

Cantina Managed: Verified.