CANTINA

Infinifi PR 209

Security Review

Cantina Managed review by:

R0bert, Lead Security Researcher
Slowfi, Security Researcher

December 11, 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

infiniFi is a self-coordinated depositor-driven system designed to tackle the challenges of duration gaps in traditional banking.

From Sep 24th to Sep 28th the Cantina team conducted a review of infinifi-contracts on commit hash 65854155. The team identified a total of **21** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	3	2	1
Medium Risk	3	2	1
Low Risk	7	4	3
Gas Optimizations	0	0	0
Informational	8	5	3
Total	**21**	**13**	**8**

3 Findings

3.1 High Risk

3.1.1 Early PT threshold lets anyone brick Pendle farm pre-init

Severity: High Risk

Context: PendleV2FarmV3.sol#L129-L130

Description: `PendleV2FarmV3` seeds `ptThreshold` to one PT, while both accounting trackers start at zero (`totalReceivedPTs` and `totalWrappedAssets`). The `onlyReconciled` modifier used by every wrap/unwrap/transfer checks `PT.balanceOf(this)` against `totalReceivedPTs` and reverts whenever the absolute difference exceeds `ptThreshold`. Immediately after deployment an external actor can freely push more than one PT into the contract (a plain ERC20 transfer succeeds because no guard is in place). At that point the live balance is > 1 PT while `totalReceivedPTs` remains 0, so every `onlyReconciled` gate now fails. Therefore, the farm cannot service its very first `wrapToPt`, `unwrapFromPt`, or `transferPt`.

The intended escape hatch, `reconcilePt`, also reverts because it calls `_estimateAssetsValue`, which requires both `totalReceivedPTs` and `totalWrappedAssets` to be non-zero, values that only change after the very first wrap. Consequently the protocol is stuck until governance manually intervenes (for example by raising `ptThreshold` high enough or forcing `_handleBalanceChange` through a `setMaturityPTDiscount` call). This is a straightforward griefing vector: a cheap PT transfer performed once, right after the farm deployment, locks the farm before it ever goes live.

Recommendation: Delay enforcing reconciliation until the farm has recorded its first wrap. For example, initialize `ptThreshold` to zero, skip the `onlyReconciled` difference check while `totalReceivedPTs == 0`, or allow `reconcilePt` to succeed when both tracked totals are zero so the farm can clear unsolicited deposits without governance intervention.

infiniFi: Fixed in commit 5fee6d0.

Cantina Managed: Fix verified.

3.1.2 Rebasing pivot tokens break Pendle PT accounting

Severity: High Risk

Context: PendleV2FarmV3.sol#L201-L207

Description: `PendleV2FarmV3` treats PT exposure as `totalReceivedPTs` worth of the Pendle SY's pivot token converted into the farm's `assetToken`. During `_handleBalanceChange` it measures that value once, using the current pivot -> asset exchange rate and stores it in `totalWrappedAssets`, setting (in the actual case described) `yieldDifference` and therefore `accrualRate` to zero. This implicitly assumes the pivot token's price relative to the asset token will stay constant until maturity. Markets whose `SY.assetInfo()` points to a share-style asset like stETH or sUSDe break this assumption because the share's value grows through an increasing exchange rate rather than a larger token balance. The farm keeps reporting the original value even though each PT now converts into more of the accounting asset. Yield accounting is therefore wrong: `assets()` stays flat, the interpolated yield never accrues and later reconciliation steps must absorb the entire appreciation in a single jump. Any component that trusts the farm's reported yield operates on these flattened numbers and then sees a discontinuous jump once the exchange-rate appreciation is finally folded in.

Proof of Concept: Check gist 95b565d4.

The integration test forks mainnet and uses a Pendle market where `SY.assetInfo()` is stETH (a share token whose exchange rate versus wstETH has compounded for years). Running `forge test --match-test testWrap_wstETH` shows:

1. The test "deals" exactly 1 wstETH to the farm; oracles price wstETH at 5,000 USD while stETH is fixed at 4,000 USD.
2. `_handleBalanceChange` mints 1.2227192155 PT (reflecting the wstETH->stETH unwrap) but records `totalWrappedAssets = assets` = 4,881,095,108 because it multiplies the PT amount by the stale 4,000 USD stETH price.
3. `accrualRate` and `remainingYield` remain zero.

Prices:

- 1 stETH = 4000 USD.
- 1 wstETH = 5000 USD.

Essentially 1 wstETH, gives 1.2227 PT. Considering that 1 PT is equal to 1 stETH at maturity, under the current stETH value (as the farm interprets), the wstETH provided equals to 1.2227 * 4000 = 4890.8 USD. A loss of around 110 USD. For that reason, `accrualRate` and `remainingYield` are set to 0, as they require that assets at maturity are higher than the current assets. However, that is not the case as the `ptToAssetsAtMaturity` function prices the stETH at the price it has now, not at the maturity date.

Recommendation: Always refuse Pendle markets whose `SY.assetInfo()` yields a share-style token.

infiniFi: Acknowledged. This is a limitation of the current implementation and we are aware of this behaviour and will perform due diligence before linking a Pendle market to their farms. We will never link such Pendle markets to our `PendleV2FarmV3`.

Cantina Managed: Acknowledged.

3.1.3 `config` declared as memory prevents persistence of state updates

Severity: High Risk

Context: SwapFarmV2.sol#L99-L105

Description: In `SwapFarmV2.sol:99`, the swap pair configuration is retrieved into a `memory` variable:

```
PairConfig memory config = getSwapPairConfig(_tokenIn, _tokenOut);
```

After performing checks, the modifier updates `config.lastSwap` to `block.timestamp`. Since `config` is declared as `memory`, this update does not persist to storage. As a result, the cooldown mechanism does not update correctly, and swaps may bypass intended throttling.

Recommendation: Consider to declare `config` as a `storage` variable instead of `memory`, ensuring updates to `lastSwap` are persisted in contract storage.

infiniFi: Fixed in commit 5fee6d0.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Aggregator swaps revert for secondary input tokens

Severity: Medium Risk

Context: SwapFarmV2.sol#L221

Description: The `SwapFarmV2.swapWithAggregator` function unconditionally sets the router allowance on `assetToken` before routing the call. The function otherwise allows any supported `_tokenIn`, but when `_tokenIn` differs from assetToken (i.e. secondary assets enabled through `MultiAssetFarmV2`), the router receives zero approval for the actual input token and the low-level call reverts. As soon as the farm needs to trade a secondary asset via an aggregator, swaps fail, blocking any rebalancing and leaving the farm stuck with the undesired asset.

Recommendation: Approve the router for the actual input token by replacing the hardcoded `assetToken` with `_tokenIn`, ensuring the allowance matches the token being transferred.

infiniFi: Fixed in commit 5fee6d0.

Cantina Managed: Fix verified.

3.2.2 Migration hook can allow minting existing unreconciled farm assets to the caller

Severity: Medium Risk

Context: MigrationController.sol#L126-L136

Description: `MigrationController.migrate` snapshots `assetsBefore`, transfers `_token` into the farm, executes an arbitrary farm hook from `config.selector`, obtains `assetsAfter`, and mints iUSD for the full `assetsAfter - assetsBefore`. The contract assumes that the only state change between the snapshots is the fresh `_amount` provided by the migrator.

In practice, a hook may reconcile pre-existing discrepancies and suddenly fold latent value into the farm's `assets()`. `PendleV2FarmV3` farms are a concrete example: they expose `reconcilePt()` to pull previously unaccounted PT balances (airdrop rewards, manual sweeps, emergency withdrawals, farm-to-farm transfers, etc.) into NAV. Calling that reconciliation inside the migration hook is enough for the first migrator to capture the entire PT surplus in addition to their own deposit, because the controller measures the delta after the hook has boosted `assetsAfter`. Tests already hint at this configuration by wiring a reconciliation-style selector. Nothing in the migration flow or role gating prevents such a hook from being used within the migration flow and the effect is that unrelated collateral is minted as fresh iUSD to whoever happens to migrate immediately after an unreconciled balance change.

Recommendation: Be aware of this behaviour and exclude hook-induced preexisting NAV adjustments from the migration payout. Invoke the hook (or any other reconciliation) before recording `assetsBefore`, or otherwise strip hook results out of the final delta. A safer alternative is to compute the minted iUSD strictly from the known `_amount` converted via oracle pricing, ensuring latent accounting changes cannot be siphoned by migrators.

infiniFi: Acknowledged. We will generally precede this with disabling migration feature for these farms as transferring PTs before maturity should be very very rare event. So rare that in normal operations it should never happen. In the past the only case when we had to do this was after maturity when migrating to a new farm. Fortunately this cancels out the negative effect there.

Cantina Managed: Acknowledged.

3.2.3 Unvalidated config in migrate may lead to loss of user funds

Severity: Medium Risk

Context: MigrationController.sol#L106

Description: In `src/funding/MigrationController.sol:106`, the function `migrate(address _recipient, address _farm, address _token, uint256 _amount, uint32 _unwindingEpochs)` may proceed when the provided `(_farm, _token)` pair has no existing configuration. In such case, funds are transferred to the farm but no iUSD is minted, resulting in user funds being lost. Additionally, there is no check that the resulting `iusdAmount` is greater than zero.

Recommendation: Consider to add explicit validation that a valid configuration exists for the given `(_farm, _token)` pair and that the calculated `iusdAmount` is greater than zero before transferring user funds.

infiniFi: Fixed in PR 209.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 Unable to disable supported asset when oracle fails

Severity: Low Risk

Context: MultiAssetFarmV2.sol#L219

Description: The `MultiAssetFarmV2._disableAsset` function validates `require(isAssetSupported(_asset), InvalidAsset(_asset));` before removing the asset from the set. `isAssetSupported` returns `_asset == assetToken || (_assetTokens.contains(_asset) && _hasOracle(_asset))`. The `_hasOracle` helper catches oracle failures and returns `false` when the oracle reverts or reports a zero price, exactly the state encountered when a feed fails or is intentionally halted:

```
function _hasOracle(address _asset) internal view returns (bool) {
    try Accounting(accounting).price(_asset) returns (uint256 _price) {
        return _price > 0;
    } catch {
        return false;
```

```
    }
}
```

In that scenario, `isAssetSupported` becomes `false` and `_disableAsset` reverts, leaving governance unable to offboard the asset despite the feed being unusable. The farm remains stuck with a broken asset in its registry, so accounting calls that rely on prices continue to revert, disabling the entire farm's operations.

Recommendation: Relax the validation in `_disableAsset` to check only membership in `_assetTokens` (and inequality with assetToken), e.g. `require(_assetTokens.contains(_asset), InvalidAsset(_asset));`, so governance can always disable an asset even when its oracle feed is offline.

infiniFi: Acknowledged. It is possible to swap out the oracle any time we want going through the 1 day timelock. Another problem is that this would cause the protocol to be unable to mint siUSD and not able to report yield. We have to live with this possibility unfortunately.

Cantina Managed: Acknowledged.

3.3.2 CoWSwap signing cooldown never enforced

Severity: Low Risk

Context: CoWSwapBase.sol#L47-L51

Description: `CoWSwapBase` advertises a 20-minute signing cooldown via `_SIGN_COOLDOWN` and even exposes a `SwapCooldown` error, yet `_checkSwapApproveAndSignOrder` simply calls `_validateSwap`, approves the relayer and signs the order without tracking any timestamp. There is no state variable such as `lastOrderSignTimestamp`, nor is `SwapCooldown` ever raised.

Contracts like `PendleV2FarmV3` that inherit `CoWSwapBase` therefore let any address with `FARM_SWAP_CALLER` sign an unlimited number of back-to-back CoW orders even though the interface and comments claim a cooldown guard exists.

Recommendation: Consider storing the last sign timestamp and revert when a new request arrives before `lastSign + _SIGN_COOLDOWN`.

infiniFi: Fixed in commit 5fee6d0.

Cantina Managed: Fix verified.

3.3.3 `SwapFarmV2` duration immutable left at zero

Severity: Low Risk

Context: SwapFarmV2.sol#L56-L60

Description: `SwapFarmV2` declares an immutable `duration` meant to delay maturity, yet the constructor never assigns it. Solidity initializes immutables to zero when omitted, so `maturity()` always returns the current timestamp instead of "now + duration." Any off-chain or on-chain component treating `IMaturityFarm.maturity()` as the lockup horizon will conclude the farm is immediately mature, undermining the intended illiquidity model and documentation.

Recommendation: Have the constructor accept a duration argument and set the immutable, e.g. `duration = _duration;`, so `maturity()` exposes the configured lock period.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.3.4 Potential reentrancy during wrap/unwrap calls

Severity: Low Risk

Context: PendleV2FarmV3.sol#L409-L423

Description: `PendleV2FarmV3.wrapToPt` deposits via `address(pendleRouter).call(_calldata)` before it refreshes `totalReceivedPTs`. During that external call, the router (or any downstream hook) can re-enter and invoke the unguarded `reconcilePt()`.

Because the freshly minted PTs already sit on the farm but have not yet been recorded, reconciliation observes a positive `ptDifference`, runs `_estimateAssetsValue` and immediately feeds the gain into `_handleBalanceChange`. When execution returns to `wrapToPt`, the function continues and executes its own `_handleBalanceChange(int256(assetAmountIn))`, so the same deposit is credited twice: `totalWrappedAssets` jumps early, remaining yield shrinks, and `assets()` becomes overstated. The mirror scenario on `unwrapFromPt` can drive the opposite distortion (double-counting the loss).

Recommendation: Prevent intra-call reentrancy by adding non-reentrancy to `wrapToPt`, `unwrapFromPt` and `reconcilePt`, or by enforcing a dedicated mutex.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.3.5 Swap pair key collisions corrupt router configuration

Severity: Low Risk

Context: SwapFarmV2.sol#L147-L153

Description: `SwapFarmV2.getSwapPairKey` derives the key as `keccak256(abi.encode(bytes20(_tokenIn) & bytes20(_tokenOut)))`. The bitwise `AND` zeroes every bit not shared by both addresses, so unrelated pairs frequently collapse to identical results. For example: `0x1111...1111 & 0x2222...2222` and `0x3333...3333 & 0x4444...4444` both produce `0x0000...0000`, yielding the same hash. Calling `setPairConfig` on (tokenA, tokenB) and later on (tokenC, tokenD) silently overwrites the first config in `pairConfig`. Subsequent swaps for (tokenA, tokenB) then enforce the cooldown/slippage defined for (tokenC, tokenD) (or vice versa), breaking risk limits and potentially triggering unexpected slippage or swap throttling.

Recommendation: Construct an order-independent key without discarding address bits. A minimal fix is to sort the addresses and hash them. For example:

```
address a = _tokenIn < _tokenOut ? _tokenIn : _tokenOut;
address b = _tokenIn < _tokenOut ? _tokenOut : _tokenIn;
return keccak256(abi.encodePacked(a, b));
```

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.3.6 Lack of native-token handling when `tokensOut` is zero address in Pendle markets

Severity: Low Risk

Context: PendleV2FarmV3.sol#L125-L127, PendleV2FarmV3.sol#L265-L303, PendleV2FarmV3.sol#L350-L381

Description: Pendle markets can configure `tokensOut` to the zero address to denote the native token. Current integrations appear to assume ERC-20 semantics (non-zero address and `IERC20` transfers/approvals). If a market (or its rewards) outputs the native token, flows such as transfers, approvals, accounting, and reward withdrawals can revert or mis-account because zero address is treated as an ERC-20. This also complicates downstream swapping/bridging paths that expect ERC-20 tokens.

Recommendation: Consider to either (a) explicitly reject Pendle markets where `tokensOut` is the native token, or (b) add first-class native-token support: normalize zero-address outputs to WETH for internal flows (wrap/unwrap at boundaries), avoid `IERC20` calls on zero address, use payable sends with proper reentrancy protection, and ensure reward withdrawal and swap paths handle native tokens consistently. Add targeted tests for deposits/withdrawals/rewards when the market outputs the native token.

infiniFi: Acknowledged. In case there is a need to support native tokens (in future) implementations we have taken note of this, but nothing for us to implement here right now.

Cantina Managed: Acknowledged.

3.3.7 Cap update resets `migrated` counter to zero

Severity: Low Risk

Context: MigrationController.sol#L94

Description: In `src/funding/MigrationController.sol:94`, the config initialization sets `migrated: uint112(0)`. If governance updates the cap by rebuilding/replacing the struct, the `migrated` amount can be unintentionally reset to zero. This desynchronizes accounting and may allow exceeding the intended cap or misreporting progress.

Recommendation: Consider to add an `updateConfig` path that preserves the existing `migrated` value (or reads it from storage and carries it forward) when changing the cap, or split cap and migrated into separate storage so cap updates do not reinitialize `migrated`.

infiniFi: Acknowledged. It's fine as it is now, e.g. if the protocol has a sUSDe migration with a cap of 20M and we have 12M already used, we know when we set the config again that we have to set a cap of 8M because that's what's remaining.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 Unused `minMigrationAmount` state variable

Severity: Informational

Context: MigrationController.sol#L54-L55

Description: `MigrationController` declares the `minMigrationAmount` state variable, initializes it to 1e18, and exposes it publicly, but no logic ever reads or updates this storage slot. The actual minimum-per-migration check in migrate relies on the per-farm configuration (`_config.minMigrationAmount`), making the global variable redundant.

Recommendation: Consider removing the unused `minMigrationAmount` state variable (and any related comments) or wire it into the migration check if a global minimum is desired.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.4.2 Pendle farm PT threshold initialized 10x lower than documented

Severity: Informational

Context: PendleV2FarmV3.sol#L129-L130

Description: `PendleV2FarmV3` intends to start with a `ptThreshold` worth roughly ten dollars, per the inline comment:

```
// set default threshold to 10$
ptThreshold = 10 ** (PT.decimals());
```

The implementation, however, sets `ptThreshold = 10 ** PT.decimals()`. For an 18-decimals PT this equals 1e18, i.e. one whole PT. Because PTs track a dollar-ish stable asset, the live threshold is ~$1, not the stated $10. That tenfold gap means even small reconciliation differences (e.g., <1.1 PT of oracle drift, airdropped rewards, or partial unwinds) exceed the guard, causing `onlyReconciled` to block wraps/unwraps and forcing operators to intervene much more often than anticipated.

Recommendation: Multiply by 10 to match the documented target, e.g. `ptThreshold = 10 * 10 ** PT.decimals();`.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.4.3 `PTZappedOut` event logs token/PT amounts swapped

Severity: Informational

Context: PendleV2FarmV3.sol#L340

Description: `PendleV2FarmV3` emits `PTZappedOut` with arguments (`timestamp`, `_tokenOut`, `_ptTokensIn`, `tokensOut`, `assetsOut`). The interface event definition expects (`timestamp`, `tokenOut`, `tokenOutAmount`, `ptTokensIn`, `assetsReceived`). Because the emit flips `tokenOutAmount` and `ptTokensIn`, listeners parsing the

log read the PT amount where they expect the actual token-output amount and vice versa. That corrupts downstream analytics and any off-chain automations relying on the event data for reconciliation or accounting.

Recommendation: Swap the middle arguments in the emit so it matches the event signature: `emit PTZappedOut(block.timestamp, _tokenOut, tokensOut, _ptTokensIn, assetsOut);`.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.4.4 Auto-reconcile on PT transfer reverts for fresh destination farms

Severity: Informational

Context: PendleV2FarmV3.sol#L402-L404

Description: `transferPt` optionally calls the receiver's `reconcilePt()` when `_reconcile=true`. That function invokes `_estimateAssetsValue`, which requires `totalReceivedPTs > 0` and `totalWrappedAssets > 0`. Newly deployed farms, or ones that have not wrapped assets yet, have both counters at zero, triggering `FarmNotUsed(...)` and reverting the entire transfer. In practice this means you cannot bootstrap a second farm by transferring PTs with reconciliation enabled: the first attempt to balance inventories fails unless governance pre-primes the destination via a wrap.

Recommendation: Before calling the receiver's `reconcilePt`, ensure the destination farm is primed (e.g., query a lightweight flag or skip reconciliation when its tracked totals are zero). Alternatively, modify `reconcilePt` to succeed when `totalReceivedPTs` and `totalWrappedAssets` are zero so the first incoming transfer can initialize the farm without manual preparation.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Consider using `safeTransfer`

Severity: Informational

Context: PendleV2FarmV3.sol#L397

Description: In `PendleV2FarmV3.sol#397`, the code uses `assert(PT.transfer(ptReceiver, _amount));`. Using `assert` for an external token transfer is not suitable: `assert` is meant for internal invariants and emits a generic panic on failure, while ERC-20 transfers can legitimately fail due to runtime conditions. Additionally, relying on the boolean return is brittle with non-standard tokens. Using `SafeERC20` provides safer handling and clearer failures.

Recommendation: Consider to replace the call with `SafeERC20.safeTransfer(PT, ptReceiver, _amount)` (or a `require` with a custom error) to handle non-standard tokens and return clear error semantics.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.4.6 Zero-balance check enables dust griefing, blocking asset disable

Severity: Informational

Context: MultiAssetFarmV2.sol#L222-L223

Description: In `src/integrations/MultiAssetFarmV2.sol:222`, the function enforces `require(balance == 0, InvalidBalance(_asset, balance));`. Any account can transfer a minimal amount of `_asset` to the contract, making `balance > 0` and causing the call to revert. This creates a denial-of-service on disabling the asset (and any admin flow that relies on a zero balance), potentially leaving configurations stuck.

Recommendation: Consider to avoid strict zero checks by either permitting a small dust threshold, or adding a controlled sweep/rescue path that forwards any unexpected `_asset` balance to a protocol address before enforcing the condition.

infiniFi: Fixed in 5fee6d0.

Cantina Managed: Fix verified.

3.4.7 Implicit peg assumptions for non fully backed stable coins

Severity: Informational

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

Description: Multiple flows appear to assume USDe is strictly $1. While USDC depeg is less probable, USDe has a higher likelihood of price variance. Hard-coded $1 assumptions (e.g., fixed `1e18` conversions, lack of oracle checks, or invariant pricing in mint/quote/accounting paths) can misprice assets and rewards, distort NAV, and allow operations to proceed under incorrect valuations during a depeg.

Recommendation: Consider to replace fixed-peg assumptions with per-asset pricing parameters and oracles (with deviation thresholds), apply conservative haircuts for USDe, and add circuit breakers to pause mint/swap/accounting transitions when stablecoins deviate beyond a configured band. Parameterize these controls per asset and include tests for depeg scenarios.

This is a protocol feature and it is a purely informational issue.

infiniFi: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 Excessive default slippage tolerance for some farms

Severity: Informational

Context: PendleV2FarmV3.sol#L131-L132

Description: `PendleV2FarmV3` initializes `maxSlippage` to 0.997e18 at construction, which enforces a strict 0.3% tolerance on every wrap, unwrap, zap and CoWSwap order. While that bound works for very stable PT markets, it becomes too tight as soon as the farm targets more volatile Pendle pools or sources liquidity from thinner venues. In those cases `wrapToPt`, `unwrapFromPt`, or a Cow order can revert because the on-chain quote legitimately lands a bit worse than 0.3%, effectively locking the farm out of rebalancing. A call to `setMaxSlippage` will be required to update the `maxSlippage` to a lower value and be able to execute the swap.

Recommendation: Set the constructor default to a configurable value depending on the targeted farm volatility.

infiniFi: Acknowledged. Ok with current values. Can be changed through the 1 day timelock.

Cantina Managed: Acknowledged.