Pendle Finance Security Review

Reviewers
0xleastwood

September 13, 2022

Contents

1 Introduction **3**

2 Risk classification **3**
2.1 Impact . . . 3
2.2 Likelihood . . . 3
2.3 Action required for severity levels . . . 3

3 Executive Summary **4**

4 Findings Part 1 **5**
4.1 Medium Risk . . . 5
4.1.1 `pyIndexCurrent()` and `rewardIndexesCurrent()` update state but do not utilise the `nonReentrant` modifier . . . 5
4.1.2 Native ether `SCY` deposits will revert in `_mintScyFromToken()` . . . 5
4.1.3 `SCY` token redemptions will be intermittently locked while the borrow rate is absurdly high . . . 5
4.2 Low Risk . . . 5
4.2.1 Potential forward compatibility issue if `SJOE` ever starts distributing rewards in the form of `JOE` tokens . . . 5
4.2.2 Certain Pendle market rewards could get skimmed by any user and sent to the treasury . . . 6
4.2.3 If interest/rewards are ever paid out in the form of `SCY` tokens, users could steal these tokens by calling `mintPY()` . . . 6
4.3 Informational . . . 6
4.3.1 Mismatch between comment on the `LOWER_BOUND_APPROVAL` variable and its implementation 6
4.3.2 Typo in `MathMarketCore.addLiquidity()` . . . 6
4.3.3 Potentially dangerous equality check in the Stargate implementation of `_redeem()` . . . 6
4.4 Gas Optimisations . . . 7
4.4.1 Unnecessary checks when distributing interest . . . 7
4.4.2 Unnecessary checks when distributing rewards . . . 7

5 Findings Part 2 **8**
5.1 Medium Risk . . . 8
5.1.1 Rewards on past epochs will be lost . . . 8
5.1.2 Total supply will go out of sync if `_broadcastPosition()` is not frequently called . . . 8
5.1.3 Users can game reward token distributions . . . 9
5.2 Low Risk . . . 9
5.2.1 Anyone can force users to withdraw their expired `vePENDLE` tokens . . . 9
5.2.2 Potential reentrancy in `_updateRewardIndex()` could allow a user to make state changes prior to reward allocations . . . 9
5.2.3 `_increasePosition()` truncates tokens . . . 10
5.2.4 Celer messages may fail and prevent users from broadcasting balance updates to sidechains 10
5.2.5 User may send too much native ether in `_sendMessage()` . . . 10
5.2.6 Users can instantly finalise and broadcast results after deployment . . . 11
5.3 Informational . . . 11
5.3.1 The protocol is not able to handle fee-on-transfer/rebasing tokens provided as rewards to stakers . . . 11
5.3.2 An overly inactive contract may become locked due to an excessive reliance on iterating through past epochs . . . 11
5.3.3 Token distributions can be front-run by sophisticated stakers . . . 11
5.3.4 `_getRewardTokens()` iterates through an unbounded array . . . 12
5.3.5 `getValueAt()` is missing certain invariant checks . . . 12
5.3.6 `_receiveVotingResults()` does not validate untrusted data . . . 12
5.3.7 The liveness of the protocol is impacted if the rewards contract is not consistently funded with Pendle tokens . . . 12
5.4 Gas Optimisations . . . 13
5.4.1 Compiler optimisation runs can be increased . . . 13

5.4.2 `_doTransferOutRewards()` can be heavily optimised, reducing total `SLOAD` count 13
5.4.3 Checkpointing mechanism adds a lot of gas overhead 13

1 Introduction

Pendle Finance enables the tokenisation and trading of yield tokens by splitting yield-bearing assets into their respective principal and yield components. Users can use the principal token to generate additional yield or altentively, they may also trade the future yield of their collateral to other market participants.

The focus of the security review was on the following attack vectors:

1. Any interaction with the Pendle AMM which would allow the trader to benefit from due to rounding.
2. Siphoning of interest and gauge rewards from other users.
3. Improper `SCY` implementation when integrating with yield generating protocols.
4. Any way a user could exit the protocol with more funds than intended.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time according to *two* specific commits which have undergone review by an independent auditor. Any modifications to the code will require a new security review.

2 Risk classification

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

2.1 Impact

- High - leads to a loss of a significant portion (*>10%*) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses *<10%* or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

2.2 Likelihood

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

2.3 Action required for severity levels

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

3 Executive Summary

Over the course of *3.5* weeks in total, Pendle Finance engaged with `0xleastwood` to review Pendle V2. In this time, a total of *X* issues were found. It is important to note that the audit was conducted on *two* separate timelines with different commit hashes.

The audit can be split into *two* distinct parts:

- **Liquidity Mining and vePENDLE** - *1* week
  - *core/LiquidityMining/**
  - *libraries/VeBalanceLib.sol*
  - *libraries/WeekMath.sol*
  - *libraries/MiniHelpers.sol*
- **Yield Tokenisation and Yield Trading** - *2.5* weeks
  In-scope assets include all contracts except for the following.
  - Any contract included as part of the Liquidity Mining and `vePENDLE` audit.
  - *offchain-helpers/**
  - *SuperComposableYield/SCY-implementations/AaveV3/**
  - *PendleERC4626SCY.sol*
  - *PendleWstEthSCY.sol*
  - *PendleYearnVaultSCY.sol*

Summary

Project Name	Pendle Finance
Type of Project	Yield Tokenisation
Repository	pendle-core-internal-v2
Part 1 Timeline	Aug 22, 2022 - Sept 7, 2022
Part 1 Commit	cbe8bc0c46837c7dbb8e19de977d77f34bdcc649
Part 2 Timeline	June 13, 2022 - June 17, 2022
Part 2 Commit	0066ceca2a414d14aaff262c2908324fc2a8eee1
Methods	Manual Review

Total Issues

Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	6
Informational	7
Gas Optimisations	3

4 Findings Part 1

4.1 Medium Risk

4.1.1 `pyIndexCurrent()` and `rewardIndexesCurrent()` update state but do not utilise the `nonReentrant` modifier

Severity: Medium

Context: SCYBaseWithRewards.sol#L75-L78, PendleYieldToken.sol#L183-L187

Description: The `rewardIndexesCurrent()` function allows any user to call it and update the balance state for all reward tokens. New token balances are distributed to `SCY` token holders, however, there are other functions which rely on the updated state of this function. If reentrancy is allowed, it may be possible to update the reward index at an unexpected point of execution. Similarly, `pyIndexCurrent()` updates the exchange rate index which is used to determine the value of a principal token at a given point in time. Both of these functions update sensitive data and should be protected accordingly.

Recommendation: Add the `nonReentrant` modifier to the aforementioned functions.

4.1.2 Native ether `SCY` deposits will revert in `_mintScyFromToken()`

Severity: Medium

Context: ActionSCYAndPYBase.sol#L24-L42, ActionSCYAndYTBase.sol#L170-L189, ActionYT.sol#L66-L81, ActionCore.sol#L170-L186

Description: The `_mintScyFromToken()` function is used by the action contracts to swap valid base tokens into their correspondng `SCY` token amounts via the Pendle AMM. However, the `ISuperComposableYield.deposit()` function is implemented such that native ether deposits should be accepted and hence, `_mintScyFromToken()` should also be able to handle such.

Recommendation: Consider updating the `_mintScyFromToken()` function to accept native ether deposits and mark `ActionYT.swapExactTokenForYt()` and `ActionCore.mintScyFromToken()` as `payable`.

4.1.3 `SCY` token redemptions will be intermittently locked while the borrow rate is absurdly high

Severity: Medium

Context: PendleQiTokenHelper.sol#L36, PendleQiSAvaxSCY.sol#L139-L142, PendleYieldToken.sol#L183-L187, PendleYieldToken.sol#L230

Description: There is currently a limitating factor in `_exchangeRateCurrentView()` whereby the function will revert if `borrowRateMantissa` exceeds the `borrowRateMaxMantissa`. In this case, all instances of the exchange rate being used will break, preventing all redemptions via the `PendleYieldToken.sol` contract.

Recommendation: Consider handling errors within the `pyIndexCurrent()` function silently. This function should not revert as it is relied upon by the `_redeemPY()` function.

4.2 Low Risk

4.2.1 Potential forward compatibility issue if `SJOE` ever starts distributing rewards in the form of `JOE` tokens

Severity: Low

Context: StableJoeStaking.sol#L331-L353, PendleSJoeSCY.sol

Description: Rewards are to be distributed to `JOE` stakers. It is possible at some point in time that `JOE` will be added as a reward tokent to `SJOE`. This would create further issues in the `PendleSJoeSCY.sol` contract.

Recommendation: Ensure this is understood and well-documented.

4.2.2 Certain Pendle market rewards could get skimmed by any user and sent to the treasury

Severity: Low

Context: PendleMarket.sol#L224-L230

Description: Liquidity providers in the Pendle market AMM receive rewards via the Pendle Gauge and through fees on each trade. If for whatever reason, rewards are paid out in the form of additional `SCY` or `PY` tokens, `skim()` will effectively transfer these owed rewards to the treasury address. This could create an issue on any mass withdrawal event.

Recommendation: Ensure this is understood and well-defined so in the event `SCY` or `PT` tokens are added as a reward to the Pendle Gauge, it will not be siphoned away via the `skim()` function.

4.2.3 If interest/rewards are ever paid out in the form of `SCY` tokens, users could steal these tokens by calling `mintPY()`

Severity: Low

Context: PendleYieldToken.sol#L67-L84, PendleYieldToken.sol#L276-L279

Description: While the `PendleYieldToken.sol` contract is designed such that *one* `YT` token yields more `SCY` tokens overtime, there are also additional rewards paid out on top of this. Hence, if rewards are ever paid out in the form of `SCY` tokens, `_getFloatingScyAmount()` will incorrectly calculate a floating amount which could be minted via `mintPY()`.

Recommendation: Ensure that `SCY` tokens are never offered as an additional reward to `YT` tokens.

4.3 Informational

4.3.1 Mismatch between comment on the `LOWER_BOUND_APPROVAL` variable and its implementation

Severity: Informational

Context: TokenHelper.sol

Description: The `LOWER_BOUND_APPROVAL` variable is used to represent the point at which a token will be re-approved to ensure the Kyberswap has a sufficient approval amount. While the comment seems to suggest that `type(uint96).max / 2` is chosen as the lower bound to maintain compatability with certain tokens, it seems the `_safeApproveInf()` function will attempt to approve `type(uint256).max`.

Recommendation: Ensure this amount is compatible with tokens which use *96* bits for approval.

4.3.2 Typo in `MathMarketCore.addLiquidity()`

Severity: Informational

Context: MarketMathCore.sol#L72-L81

Description: Call to `addLiquidityCore()` returns a variable denoted as `ptUsed`, however, `addLiquidity()` names it `_otUsed`.

Recommendation: Update this named variable to `_ptUsed` instead.

4.3.3 Potentially dangerous equality check in the Stargate implementation of `_redeem()`

Severity: Informational

Context: PendleStargateSCY.sol#L89-L112

Description: The `_redeem()` function will either burn Stargate LP tokens for the underlying asset or directly send LP tokens to the recipient. Because Stargate is a bridge, there is no guarantee that there are sufficient tokens to burn on the source chain, hence, the protocol may only burn up to a certain amount of tokens. However, Pendle's `SCY` token implementation for Stargate will actively revert when `amountLpUsed != amountSharesToRedeem`. This

may temporarily prevent users from redeeming their SCY tokens, although the user can instead opt to withdraw LP tokens directly and redeem them for the underlying asset out-of-band.

Recommendation: Ensure this is understood by users of the protocol.

4.4 Gas Optimisations

4.4.1 Unnecessary checks when distributing interest

Severity: Gas Optimisation

Context: InterestManagerYT.sol#L37-L43, InterestManagerYT.sol#L65

Description: The `_distributeInterestForTwo()` function will not call `_distributeInterestPrivate()` if interest is attempting to be distributed on the zero address or on the yield token contract itself. While these protections are important, the `_distributeInterestPrivate()` already asserts this behaviour.

Recommendation:

Consider removing the `assert` from `_distributeInterestPrivate()`.

4.4.2 Unnecessary checks when distributing rewards

Severity: Gas Optimisation

Context: RewardManagerAbstract.sol#L28-L36, RewardManagerAbstract.sol#L44

Description: The `_updateAndDistributeRewardsForTwo()` function will not call `_distributeRewardsPrivate()` if interest is attempting to be distributed on the zero address or on the reward contract itself. While these protections are important, the `_distributeRewardsPrivate()` already asserts this behaviour.

Recommendation:

Consider removing the `assert` from `_distributeRewardsPrivate()`.

5 Findings Part 2

5.1 Medium Risk

5.1.1 Rewards on past epochs will be lost

Severity: Medium

Context: VotingControllerStorage.sol#L206-L212, PendleVotingController.sol#L125-L131

Description: Before broadcasting results to to other sidechains, each epoch must first be finalised beforehand. During finalisation, the final pool vote for the epoch is set and then `_setAllPastEpochsAsFinalized()` will attempt to finalise all prior epochs.

However, because `wTime` decrements by `deployedWTime` on each iteration, it is expected that this will only allow for a single epoch to get finalised. As such, if epochs aren't regularly being finalised, it is possible for rewards to be completely lost.

```
function _setAllPastEpochsAsFinalized() internal {
    uint128 wTime = WeekMath.getCurrentWeekStart();
    while (wTime >= deployedWTime && isEpochFinalized[wTime] == false) {
        isEpochFinalized[wTime] = true;
        wTime -= deployedWTime;
    }
}
```

Recommendation: Update the `_setAllPastEpochsAsFinalized()` function to instead deduct `WEEK` from `wTime` instead of `deployedWTime`.

```
    while (wTime >= deployedWTime && isEpochFinalized[wTime] == false) {
        isEpochFinalized[wTime] = true;
+       wTime -= wTime;
-       wTime -= deployedWTime;
    }
```

5.1.2 Total supply will go out of sync if `_broadcastPosition()` is not frequently called

Severity: Medium

Context: VotingEscrowPendleMainchain.sol#L208-L226

Description: Upon locking `PENDLE` tokens in the voting escrow contract, it is in the best interest to broadcast balance updates to all other sidechains so as to maximise the accrual of rewards. However, due to inactivity on the mainchain, it is possible that the total supply on sidechains will decay at a higher rate due to the fact that `_updateTotalSupply()` is only called on the mainchain.

It is important to note that this issue only arises if on any given epoch, no user creates or increases their lock position. Typical voting and claiming of rewards on the mainchain will not broadcast the updated total supply to all sidechains. As such, there really is no incentive to do so as sophisticated users may wish to maximise their claim of rewards on all sidechains.

```
function _updateTotalSupply() internal returns (VeBalance memory, uint128) {
    VeBalance memory supply = _totalSupply;
    uint128 wTime = lastSupplyUpdatedAt;
    uint128 currentWeekStart = WeekMath.getCurrentWeekStart();

    if (wTime >= currentWeekStart) {
        return (supply, wTime);
    }

    while (wTime < currentWeekStart) {
        wTime += WEEK;
        supply = supply.sub(slopeChanges[wTime], wTime);
        totalSupplyAt[wTime] = supply.getValueAt(wTime);
    }

    _totalSupply = supply;
    lastSupplyUpdatedAt = wTime;

    return (supply, lastSupplyUpdatedAt);
}
```

Recommendation: While it is difficult to fully mitigate this, ensuring that the total supply is up to date before any action to vote or receive rewards on the mainchain will go a long way to preventing future abuse. It may be useful to modify the `_updateTotalSupply()` such that it also broadcasts the total supply if it has changed.

5.1.3 Users can game reward token distributions

Severity: Medium

Context: PendleGauge.sol#L51-L59, PendleGauge.sol#L88-L90, RewardManager.sol#L44

Description: Anyone can update the active balance of an arbitrary user by sending `1 wei` of tokens to the victim's account. The `_updateUserActiveBalance()` function takes the minimum of the user's staked LP balance and their boosted `vePENDLE` token balance. Because `vePENDLE` decays linearly, it makes sense that this active balance will decay at a similar rate unless the user regularly tops up their locked `vePENDLE` balance.

As such, users are incentvized to reduce the gauge's `totalActiveSupply` by updating the active balance of other accounts. By doing this, the user is able to claim a larger proportion of rewards distributed to the Pendle Gauge contract.

Recommendation: Ensure this is understood and well-documented.

5.2 Low Risk

5.2.1 Anyone can force users to withdraw their expired `vePENDLE` tokens

Severity: Low

Context: VotingEscrowPendleMainchain.sol#L91-L102

Description: Anyone can call `withdraw()` on a user with an expired lock position and force them to withdraw when they may wish to instead increase the lock duration of their existing lock position.

Recommendation: Ensure this is intended behaviour or remove the `user` argument from `withdraw()` and allow only `msg.sender` to withdraw their expired lock position.

5.2.2 Potential reentrancy in `_updateRewardIndex()` could allow a user to make state changes prior to reward allocations

Severity: Low

Context: RewardManager.sol#L38-L64, PendleGauge.sol#L79-L82

Description: If for any reason, `_redeemExternalReward()` relinquishes control over the flow of execution, it would be possible to make state changes via any function which expects the reward index to be up-to-date. This is because `lastRewardBlock` is updated prior to `_redeemExternalReward()` in `_updateRewardIndex()`, allowing for cross-contract reentrancy. Users may also be able to increase their active balance before the reward index had been updated.

Recommendation: Consider updating `lastRewardBlock` after `_redeemExternalReward()` such that following calls to `_updateRewardIndex()` do not short circuit.

5.2.3 `_increasePosition()` truncates tokens

Severity: Low

Context: VotingEscrowPendleMainchain.sol#L148-L178, VeBalanceLib.sol#L96-L104

Description: The `convertToVeBalance()` function will convert the locked position amount to its respective bias and slope amounts which are used to calculate the position's linear schedule for expiry. There will be some slight truncation in `position.amount / MAX_LOCK_TIME` which leads to a slightly lower starting `vePENDLE` token balance.

```
function convertToVeBalance(LockedPosition memory position)
    internal
    pure
    returns (VeBalance memory res)
{
    res.slope = position.amount / MAX_LOCK_TIME;
    require(res.slope > 0, "zero slope");
    res.bias = res.slope * position.expiry;
}
```

Recommendation: Ensure users understand this by encouraging that `position.amount` to be some multiple of `MAX_LOCK_TIME`. This can be enforced by reverting when `position.amount % MAX_LOCK_TIME != 0`.

5.2.4 Celer messages may fail and prevent users from broadcasting balance updates to sidechains

Severity: Low

Context: CelerSender.sol#L21-L26

Description: The `_sendMessage()` function is used to broadcast messages containing updates to balances and reward allocations. Because this function is sensistive in nature, it failing to broadcast data to other sidechains will impact the availability of the protocol on the mainchain. Hence, it would be safer to handle this unexpected behaviour by utilising Solidity's try/catch statements.

Recommendation: Consider using a try/catch statement when performing the `celerMessageBus.sendMessage()` external call.

5.2.5 User may send too much native ether in `_sendMessage()`

Severity: Low

Context: CelerSender.sol#L21-L26

Description: The `_sendMessage()` function will call `celerMessageBus.calcFee()` to check how much native ether should be provided to the `sendMessage()` call. However, there is no check to ensure that the caller of this function did not provide too much `msg.value` in this call.

Recommendation: Consider adding the following statement.`msg.value == celerMessageBus.calcFee(message)`.

```
    uint256 fee = celerMessageBus.calcFee(message);
+   require(msg.value == fee, "Incorrect value provided");
    celerMessageBus.sendMessage{ value: fee }(toAddr, chainId, message);
```

5.2.6 Users can instantly finalise and broadcast results after deployment

Severity: Low

Context: VotingControllerStorage.sol#L206-L212

Description: Upon finalising an epoch, the voting controller contract will attempt to also finalize all prior epochs and store the data within `isEpochFinalized`. Because the contract allows for instant finalisation of the first epoch, users could abuse this by voting on the pool, finalising and then reap all the Pendle rewards allocated. Therefore, it should not be possible to finalise the first epoch and it should instead be used to onboard pools and give ample time for users to stake and vote on these respective pools.

Recommendation: This can be improved by strictly *not* checking the case where `wTime == deployedWTime`.

```
+   while (wTime > deployedWTime && weekData[wTime].isEpochFinalized == false) {
-   while (wTime >= deployedWTime && weekData[wTime].isEpochFinalized == false) {
        weekData[wTime].isEpochFinalized = true;
        wTime -= WEEK;
```

5.3 Informational

5.3.1 The protocol is not able to handle fee-on-transfer/rebasing tokens provided as rewards to stakers

Severity: Informational

Context: RewardManager.sol, PendleGauge.sol, PendleYieldToken.sol

Description: Generally speaking, the protocol isn't completely equipped to handle tokens which implement some fee-on-transfer or rebasing mechanism. As such, the rewards for these token types will more than likely be incorrect and lead to accounting errors.

Recommendation: While it is assumed these tokens will not be used as rewards, it is best to ensure this holds true in all cases.

5.3.2 An overly inactive contract may become locked due to an excessive reliance on iterating through past epochs

Severity: Informational

Context: VotingEscrowPendleMainchain.sol#L186-L205, PendleVotingController.sol#L98-L115

Description: An extremely inlikely case may arise where no user broadcasts their updated lock position, meaning the total supply need not be updated. As a result, the `_updateTotalSupply()` function may fail to process all slope changes from `lastSupplyUpdatedAt` and until now. This breaks further deposits but it fortunately does not impact withdrawals. Similarly, if no pools have their vote updated for some time, finalisation and voting may also be broken.

Recommendation: Ensure it is understood that the protocol may fail under certain circumstances where it is not being actively used.

5.3.3 Token distributions can be front-run by sophisticated stakers

Severity: Informational

Context: PendleYieldToken.sol, InterestManagerYT.sol

Description: In the case where rewards are distributed to users who stake LP tokens and have Pendle tokens locked as `vePENDLE` tokens, the gamification of token distributions is mostly a non-issue. This is because the Pendle Gauge contract calculates the active balance as the minimum of the user's staked LP balance and their `VeBoostedLP` balance.

However, this doesn't necessarily apply to reward tokens distributed to the holders of yield tokens which receive interest and rewards over time. It would be possible for sophisticated stakers to call `mintPY()` prior to the distribution of reward tokens and subsequently redeem these tokens for their corresponding `SCY` tokens.

Recommendation: Ensure that this form of MEV may be unavoidable without considerable change to the protocol design. Alternatively, it may be useful to add an additional gap to reward claims and utilise some constant drip mechanism which distributes rewards on a per-block basis.

5.3.4 `_getRewardTokens()` iterates through an unbounded array

Severity: Informational

Context: PendleGauge.sol#L96-L100, RewardManager.sol#L46-L63, RewardManager.sol#L20-L75

Description: There are a few instances where reward distributions are made on an unbounded array of tokens. Because this may change depending on the `SCY` token implementation, it may be safer to cap the length of this list.

Recommendation: Consider adding some maximum token length for the list of rewards such that the protocol is protected from any malicious changes.

5.3.5 `getValueAt()` is missing certain invariant checks

Severity: Informational

Context: VeBalanceLib.sol#L58-L60

Description: Similar to the implementation of `getExpiry()`, it would be safer to check whether `t % WEEK == 0` holds true.

Recommendation: Consider implementing the aforementioned check.

5.3.6 `_receiveVotingResults()` does not validate untrusted data

Severity: Informational

Context: PendleGaugeController.sol#L88-L103, PendleGaugeControllerSidechain.sol#L15-L21, PendleVotingController.sol#L238-L247

Description: When the mainchain voting controller broadcasts results to all sidechains, the Celer network messaging system is used to securely relay relay data to be executed on the sidechain's voting controller. However, this data cannot be fully trusted, so it could be useful to check `wTime` is a valid timestamp.

Recommendation: Consider adding the following check to the `PendleGaugeControllerSidechain._executeMessage()` function. `require(wTime = WeekMath.getWeekStartTimestamp(wTime))`

5.3.7 The liveness of the protocol is impacted if the rewards contract is not consistently funded with Pendle tokens

Severity: Informational

Context: PendleGaugeController.sol#L73-L75, PendleGaugeController.sol#L60-L71, PendleGauge.sol#L79-L82, RewardManager.sol#L42

Description: The Pendle Gauge controller contract is in charge of holding all Pendle token funds distributed to markets. Because there is no guarantee that this contract will be sufficiently funded to allow markets to claim their allocated rewards, its entirely possible that `_redeemExternalReward()` will consistently revert until the Pendle Gauge controller contract has been funded. This would temporarily prevent tokens from being minted, burnt or transferred.

Recommendation: It may be worthwhile modifying the implementation of `_redeemExternalReward()` such that it will not revert under any circumstance. This could be done by utilising some try/catch statement on the `IPGaugeController(gaugeController).claimMarketReward()` external call.

5.4 Gas Optimisations

5.4.1 Compiler optimisation runs can be increased

Severity: Gas Optimisation

Context: hardhat.config.ts#L19

Description: The optimiser's `runs` argument allows the deployer to offset costs based on how often the contracts are to be used. A lower `runs` count will decrease deployment costs, but increase transaction costs for each user interacting with the protocol. Hence, if the contracts are to be frequently used, it makes sense to increase the optimiser's `runs`.

Recommendation: Consider increasing the optimiser's `runs`.

5.4.2 `_doTransferOutRewards()` can be heavily optimised, reducing total `SLOAD` count

Severity: Gas Optimisation

Context: RewardManager.sol#L97-L117

Description: Most of the computation in this function should only be done if `rewardAmounts[i] != 0` holds `true`. Hence, the following code provides a better optimisation for a function that is frequently called.

```
function _doTransferOutRewards(address user, address receiver)
    internal
    virtual
    returns (uint256[] memory rewardAmounts)
{
    address[] memory rewardTokens = _getRewardTokens();

    rewardAmounts = new uint256[](rewardTokens.length);
    for (uint256 i = 0; i < rewardTokens.length; ++i) {
        address token = rewardTokens[i];

        rewardAmounts[i] = userReward[token][user].accrued;

        if (rewardAmounts[i] != 0) {
            userReward[token][user].accrued = 0;
            rewardState[token].lastBalance -= rewardAmounts[i].Uint128();

            _transferOut(token, receiver, rewardAmounts[i]);
        }
    }
}
```

Recommendation: Consider implementing the above changes.

5.4.3 Checkpointing mechanism adds a lot of gas overhead

Severity: Gas Optimisation

Context: VotingControllerStorage.sol#L196-L204, PendleVotingController.sol#L259-L295, VeBalanceLib.sol#L67-L94

Description: Checkpoints are stored in the `userPoolCheckpoints` mapping whenever a user votes/unvotes on a pool. The `VeBalanceLib.sol` library allows for external view calls to query the checkpoint at a given timestamp. However, the process of checkpointing on each change in the pool's vote does add overhead costs.

Recommendation: While it is likely this will be used off-chain to track pool votes, it does add a lot of additional gas costs which could be unnecessary.