Pendle v2 (Part 1) Audit Report

June 28, 2022

WATCHPUG

Table of Contents

Summary

This report has been prepared for Pendle v2 (Part 1) Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

Project Summary

Project Name	**Pendle v2 / Part 1**
Codebase	**https://github.com/pendle-finance/pendle-core-internal-v2**
Commit	**74b400862eff66f28b604d457e202d4f20acc387**
Language	**Solidity**

Audit Summary

Delivery Date	**June 28, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**10**

WP-H0: Improper handling of deposit with a `baseToken` that is also a `rewardToken` may result in users losing the rewards

High

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/SCY-implementations/PendleQiTokenSCY.sol#L128-L132

```
128     function _getRewardTokens() internal view override returns (address[] memory res) {
129         res = new address[](2);
130         res[0] = QI;
131         res[1] = WAVAX;
132     }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/SCY-implementations/PendleQiTokenSCY.sol#L134-L144

```
134     function _redeemExternalReward() internal override {
135         address[] memory holders = new address[](1);
136         address[] memory qiTokens = new address[](1);
137         holders[0] = address(this);
138         qiTokens[0] = qiToken;
139 
140         IBenQiComptroller(comptroller).claimReward(0, holders, qiTokens, false, true);
141         IBenQiComptroller(comptroller).claimReward(1, holders, qiTokens, false, true);
142 
143         if (address(this).balance != 0) IWETH(WAVAX).deposit{ value: address(this).balance };
144     }
```

There will be certain amounts of RewardTokens accumulated and unclaimed rewards in the `PendleQiTokenSCY` contract, which belong to the existing users and can be claimed later.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/SCY-implementations/PendleQiTokenSCY.sol#L153-L157

```
153     function getBaseTokens() public view override returns (address[] memory res) {
154         res = new address[](2);
155         res[0] = qiToken;
156         res[1] = underlying;
157     }
```

For example, when the `PendleQiTokenSCY` 's underlying (L156) is `QI` token, and one of the RewardTokens is also `QI` token.

And when `deposit()` with `QI` :

At L53 of `SCYBase` , the `amountDeposited` returned from `_getFloatingAmount()` including not only the amount of BaseToken (`QI`) transferred in before this `deposit()` , but also the unclaimed rewards in rewardToken (also `QI`) belongs to the existing users.

At L55 of `SCYBase` , the implementation of `_deposit()` (L71 of `PendleQiTokenSCY`) will taken the newly transferred baseToken (`QI`) and the rewardToken (`QI`) existing in the contract before the deposit to the `QiErc20` contract, and mint the SCY shares to the `receiver` (`SCYBase` L58).

As a result, the depositor has now been incorrectly credited with the unclaimed rewardTokens that belong to the existing users.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L42-L60

```
42     function deposit(
43         address receiver,
44         address tokenIn,
45         uint256 amountTokenToPull,
46         uint256 minSharesOut
```

```
47      ) external payable nonReentrant updateReserve returns (uint256
    amountSharesOut) {
48          require(isValidBaseToken(tokenIn), "SCY: Invalid tokenIn");
49
50          if (tokenIn == NATIVE) require(amountTokenToPull == 0, "can't pull eth");
51          else if (amountTokenToPull != 0) _transferIn(tokenIn, msg.sender,
    amountTokenToPull);
52
53          uint256 amountDeposited = _getFloatingAmount(tokenIn);
54
55          amountSharesOut = _deposit(tokenIn, amountDeposited);
56          require(amountSharesOut >= minSharesOut, "insufficient out");
57
58          _mint(receiver, amountSharesOut);
59          emit Deposit(msg.sender, receiver, tokenIn, amountDeposited,
    amountSharesOut);
60      }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L122-L125

```
122     function _getFloatingAmount(address token) internal view virtual returns
    (uint256) {
123         if (token != yieldToken) return _selfBalance(token);
124         return _selfBalance(token) - yieldTokenReserve;
125     }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/f75a73c77e0ae4d90cf79ca8949870be5d6fc587/contracts/SuperComposableYield/SCY-implementations/PendleQiTokenSCY.sol#L57-L77

```
57      function _deposit(address tokenIn, uint256 amount)
58          internal
59          override
60          returns (uint256 amountSharesOut)
61      {
62          if (tokenIn == qiToken) {
63              amountSharesOut = amount;
```

```
64          } else {
65              // tokenIn is underlying -> convert it into qiToken first
66              uint256 preBalanceQiToken = _selfBalance(qiToken);
67
68              if (underlying == NATIVE) {
69                  IQiAvax(qiToken).mint{ value: amount }();
70              } else {
71                  uint256 errCode = IQiErc20(qiToken).mint(amount);
72                  require(errCode == 0, "mint failed");
73              }
74
75              amountSharesOut = _selfBalance(qiToken) - preBalanceQiToken;
76          }
77      }
```

PoC

Given:

- PendleQiTokenSCY.underlying == `QI`
- Unclaimed `QI` rewards sitting on `PendleQiTokenSCY` : 1000e18

The attacker can:

1. `PendleQiTokenSCY.deposit(PendleQiTokenSCY, QI, 0, 0)`
   - SCYBase L53 `uint256 amountDeposited = _getFloatingAmount(QI)` returns `1000e18`
     - as `_getFloatingAmount()` did not consider the case of `tokenIn` == rewardToken, at SCYBase L123 it returned `QI.balanceOf(address(this))`
   - SCYBase L55 took `1000e18 QI` as the amount of `QI` tokens tranferred in for the deposit, `amountSharesOut = _deposit(QI, 1000e18)`
     - `PendleQiTokenSCY._deposit(QI, 1000e18)` called `IQiErc20(qiToken).mint(1000e18)` at L71, and deposited `1000e18 QI` to `QiErc20`, at L75, the shares minted is returned, the amount of shares is d_s
   - SCYBase L58 took d_s shares and `_mint` to the `receiver` address specified by the attacker, ie, the `PendleQiTokenSCY` contract, and the `PendleQiTokenSCY` received d_s SCY token
2. `redeem(attacker, 0, QI, 0)`
   - SCYBase L78 `uint256 amountSharesToRedeem = _getFloatingAmount(address(this))` ;

`amountSharesToRedeem` == d_s;
- SCYBase L80 called `PendleQiTokenSCY._redeem(QI, amountSharesToRedeem)` and redeemed `1000e18 QI`
- SCYBase L84 `_transferOut(QI, attacker, 1000e18)` sent `1000e18 QI` to the attacker

As a result:

- QI.balanceOf(PendleQiTokenSCY): 1000e18 -> 0
- QI.balanceOf(attacker): 0 -> 1000e18
- attacker stolen the `1000e18 QI` of rewards

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L65-L87

```
65      function redeem(
66          address receiver,
67          uint256 amountSharesToPull,
68          address tokenOut,
69          uint256 minTokenOut
70      ) external nonReentrant updateReserve returns (uint256 amountTokenOut) {
71          require(isValidBaseToken(tokenOut), "SCY: invalid tokenOut");
72
73          if (amountSharesToPull != 0) {
74              _spendAllowance(msg.sender, address(this), amountSharesToPull);
75              _transfer(msg.sender, address(this), amountSharesToPull);
76          }
77
78          uint256 amountSharesToRedeem = _getFloatingAmount(address(this));
79
80          amountTokenOut = _redeem(tokenOut, amountSharesToRedeem);
81          require(amountTokenOut >= minTokenOut, "insufficient out");
82
83          _burn(address(this), amountSharesToRedeem);
84          _transferOut(tokenOut, receiver, amountTokenOut);
85
86          emit Redeem(msg.sender, receiver, tokenOut, amountSharesToRedeem,
    amountTokenOut);
87      }
```

Status

✓ Fixed

WP-I1: New YieldContract can be created with the permisionless `createYieldContract()` with a non-uponly SCY

Informational

Issue Description

As per the whitepaper of SCYS:

> SCYS works on all SCY tokens where the compound interest is always positive (meaning, 'scyIndex(t)' is a non-decreasing function).

I.e., the `YieldContract` does not support non-uponly SCYs yet.

However, since the creation of `YieldContract` is open to anyone, a `YieldContract` may get created for a non-uponly SCY, which may malfunction or even cause fund loss to the users.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/core/YieldContracts/PendleYieldContractFactory.sol#L80-L92

```
80  function createYieldContract(address SCY, uint256 expiry)
81      external
82      returns (address PT, address YT)
83  {
84      require(expiry > block.timestamp, "expiry must be in the future");
85
86      require(expiry % expiryDivisor == 0, "must be multiple of divisor");
87
88      require(getPT[SCY][expiry] == address(0), "PT already existed");
89
90      ISuperComposableYield _SCY = ISuperComposableYield(SCY);
91
92      (, , uint8 assetDecimals) = _SCY.assetInfo();
```

Resolution

`YieldContract` can now support non-uponly SCYs. The whitepaper has been updated as well.

Status

✓ Fixed

WP-G2: Sending the rewards to the treasury in every token transfer after the expiry is gas inefficient

Gas

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/core/YieldContracts/PendleYieldToken.sol#L283-L308

```
283  /// @dev override the default updateRewardIndex to avoid distributing the rewards
     after
284  /// YT has expired. Instead, these funds will go to the treasury
285  function _updateRewardIndex() internal virtual override {
286      if (!_isExpired()) {
287          super._updateRewardIndex();
288          return;
289      }
290
291      // For the case of expired YT
292      if (lastRewardBlock == block.number) return;
293      lastRewardBlock = block.number;
294
295      _redeemExternalReward();
296
297      address[] memory rewardTokens = _getRewardTokens();
298      address treasury = IPYieldContractFactory(factory).treasury();
299
300      for (uint256 i = 0; i < rewardTokens.length; i++) {
301          address token = rewardTokens[i];
302
303          uint256 currentBalance = _selfBalance(token);
304          uint256 rewardAccrued = currentBalance - rewardState[token].lastBalance;
305
306          _transferOut(token, treasury, rewardAccrued);
307      }
308  }
```

Recommendation

L297-307 can be moved out as a standalone permissionless function as the rewards only need to be settled once a while rather than every transfer after YT has expired.

Status

✓ Fixed

WP-I3: `_updateAndDistributeInterest` will overflow when `exchangeRateCurrent > 3e38`

Informational

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/SuperComposableYield/SCY-implementations/PendleERC4626SCY.sol#L63-L70

```
63  function exchangeRateCurrent() public virtual override returns (uint256
    currentRate) {
64      currentRate = IERC4626(yieldToken).convertToAssets(Math.ONE); // @audit when 1
    share = 3e20 Assets, currentRate = 3e38
65
66      emit ExchangeRateUpdated(exchangeRateStored, currentRate);
67
68      exchangeRateStored = currentRate;
69  }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/core/YieldContracts/PendleYieldToken.sol#L237-L256

```
237 function _updateAndDistributeInterest(address user) internal {
238     uint256 prevIndex = userInterest[user].index;
239
240     (, uint256 currentIndexBeforeExpiry) = getScyIndex();
241
242     if (prevIndex == currentIndexBeforeExpiry) return;
243     if (prevIndex == 0) {
244         userInterest[user].index = currentIndexBeforeExpiry.Uint128();
245         return;
246     }
247
248     uint256 principal = balanceOf(user);
249
250     uint256 interestFromYT = (principal * (currentIndexBeforeExpiry -
    prevIndex)).divDown(
```

```
251             prevIndex * currentIndexBeforeExpiry
252         );
253
254         userInterest[user].accrued += interestFromYT.Uint128();
255         userInterest[user].index = currentIndexBeforeExpiry.Uint128();
256     }
```

If the `yieldToken` 's PPS (price per share) went crazy for whatever reason, the `PendleYieldToken` will malfunction due to overflow in `_updateAndDistributeInterest()` when casting `currentIndexBeforeExpiry` to `uint128` .

Recommendation

This issue is extremely unlikely to happen in practice, one possible solution is adding a check to ensure when creating a `PendleYieldToken` with a `PendleERC4626SCY` , the pps must not exceed a certain upper bound.

Status

✓ Fixed

WP-H4: `PendleYearnVaultScy.sol#_redeem()` When withdrawing from a `yvVault`, some of the shares tokens may not be used but still burned from the user's balance

High

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/SuperComposableYield/SCY-implementations/PendleYearnVaultScy.sol#L59-L72

```
59  function _redeem(address tokenOut, uint256 amountSharesToRedeem)
60      internal
61      virtual
62      override
63      returns (uint256 amountTokenOut)
64  {
65      if (tokenOut == yvToken) {
66          amountTokenOut = amountSharesToRedeem;
67      } else {
68          // tokenOut == underlying
69          IYearnVault(yvToken).withdraw(amountSharesToRedeem);
70          amountTokenOut = _selfBalance(underlying);
71      }
72  }
```

https://github.com/yearn/yearn-vaults/blob/v0.4.3/contracts/Vault.vy#L988-L1122

```
988  @external
989  @nonreentrant("withdraw")
990  def withdraw(
991      maxShares: uint256 = MAX_UINT256,
992      recipient: address = msg.sender,
993      maxLoss: uint256 = 1,  # 0.01% [BPS]
994  ) -> uint256:
995      shares: uint256 = maxShares  # May reduce this number below
996
```

```
997       # Max Loss is <=100%, revert otherwise
998       assert maxLoss <= MAX_BPS
999
1000      # If _shares not specified, transfer full share balance
1001      if shares == MAX_UINT256:
1002          shares = self.balanceOf[msg.sender]
1003
1004      # Limit to only the shares they own
1005      assert shares <= self.balanceOf[msg.sender]
1006
1007      # Ensure we are withdrawing something
1008      assert shares > 0
1009
1010      # See @dev note, above.
1011      value: uint256 = self._shareValue(shares)
1012
1013      if value > self.token.balanceOf(self):
1014          ....
1015          # NOTE: We have withdrawn everything possible out of the withdrawal queue
1016          #       but we still don't have enough to fully pay them back, so adjust
1017          #       to the total amount we've freed up through forced withdrawals
1018          vault_balance: uint256 = self.token.balanceOf(self)
1019          if value > vault_balance:
1020              value = vault_balance
1021              # NOTE: Burn # of shares that corresponds to what Vault has on-hand,
1022              #       including the losses that were incurred above during
                  withdrawals
1023              shares = self._sharesForAmount(value + totalLoss)
```

Unlike many other protocols, when withdrawing from a Yearn vault, it does not always consume all the `amountSharesToRedeem` requested in cases where strategies cannot withdraw all of the requested tokens (an example strategy where this can occur is with Compound and AAVE where funds may not be accessible because they were lent out).

The `amountSharesToRedeem` parameter in `IYearnVault(yvToken).withdraw(amountSharesToRedeem);` is more like a desired amount, and it's called `maxShares` in yearn's code, which also indicates that this won't always be burnt and withdrawn in full.

However, in the current implementation of `PendleYearnVaultScy#_redeem()`, when the

`amountSharesToRedeem` is not fully consumed, the user will suffer a fund loss of the unspent portion of the `amountSharesToRedeem` as it will not be returned to the user.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L65-L87

```
65  function redeem(
66      address receiver,
67      uint256 amountSharesToPull,
68      address tokenOut,
69      uint256 minTokenOut
70  ) external nonReentrant updateReserve returns (uint256 amountTokenOut) {
71      require(isValidBaseToken(tokenOut), "SCY: invalid tokenOut");
72
73      if (amountSharesToPull != 0) {
74          _spendAllowance(msg.sender, address(this), amountSharesToPull);
75          _transfer(msg.sender, address(this), amountSharesToPull);
76      }
77
78      uint256 amountSharesToRedeem = _getFloatingAmount(address(this));
79
80      amountTokenOut = _redeem(tokenOut, amountSharesToRedeem);
81      require(amountTokenOut >= minTokenOut, "insufficient out");
82
83      _burn(address(this), amountSharesToRedeem);
84      _transferOut(tokenOut, receiver, amountTokenOut);
85
86      emit Redeem(msg.sender, receiver, tokenOut, amountSharesToRedeem,
    amountTokenOut);
87  }
```

Status

✓ Fixed

WP-H5: Improper handling of `redeem()` with a `tokenOut` that is also a rewardToken

High

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/SuperComposableYield/base-implementations/SCYBase.sol#L65-L87

```
65  function redeem(
66      address receiver,
67      uint256 amountSharesToPull,
68      address tokenOut,
69      uint256 minTokenOut
70  ) external nonReentrant updateReserve returns (uint256 amountTokenOut) {
71      require(isValidBaseToken(tokenOut), "SCY: invalid tokenOut");
72
73      if (amountSharesToPull != 0) {
74          _spendAllowance(msg.sender, address(this), amountSharesToPull);
75          _transfer(msg.sender, address(this), amountSharesToPull);
76      }
77
78      uint256 amountSharesToRedeem = _getFloatingAmount(address(this));
79
80      amountTokenOut = _redeem(tokenOut, amountSharesToRedeem);
81      require(amountTokenOut >= minTokenOut, "insufficient out");
82
83      _burn(address(this), amountSharesToRedeem);
84      _transferOut(tokenOut, receiver, amountTokenOut);
85
86      emit Redeem(msg.sender, receiver, tokenOut, amountSharesToRedeem,
    amountTokenOut);
87  }
```

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/SuperComposableYield/SCY-implementations/PendleQiTokenSCY.sol#L85-L103

```
85  function _redeem(address tokenOut, uint256 amountSharesToRedeem)
86      internal
87      override
88      returns (uint256 amountBaseOut)
89  {
90      if (tokenOut == qiToken) {
91          amountBaseOut = amountSharesToRedeem;
92      } else {
93          if (underlying == NATIVE) {
94              uint256 errCode = IQiAvax(qiToken).redeem(amountSharesToRedeem);
95              require(errCode == 0, "redeem failed");
96          } else {
97              uint256 errCode = IQiErc20(qiToken).redeem(amountSharesToRedeem);
98              require(errCode == 0, "redeem failed");
99          }
100
101         amountBaseOut = _selfBalance(underlying);
102     }
103 }
```

Similar to [WP-H0], when the `PendleQiTokenSCY` 's underlying (L156) is also one of the RewardTokens (eg, `QI`).

There will be certain amounts of RewardTokens accumulated and unclaimed rewards in the `PendleQiTokenSCY` contract, which belong to the existing users and can be claimed later.

However, the current implementation of `redeem()` cant handle it properly, as a result, the unclaimed rewards will be sent to the latest user who redeemed with the `tokenOut` being the rewardToken (`QI`).

Recommendation

Consider comparing the before and after balance for the `amountBaseOut` .

Status

✓ Fixed

WP-L6: The initial liquidity provider will lose some LP which can never be redeemed

Low

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/core/Market/PendleMarket.sol#L117-L120

```
117   if (lpToReserve != 0) {
118       market.setInitialLnImpliedRate(index, initialAnchor, block.timestamp);
119       _mint(address(1), lpToReserve);
120   }
```

Unlike the Uniswap v2 pairs, `PendleMarket` is more like a disposable contract that only works before the `expiry` .

However, the current implementation will permanately locked a certain amount of LP tokens for the initial liquidity provider.

> At time t, when $L(t_*) = 0$, a user u can add dscy SCY tokens and dpt PT into the market to bootstrap it. A portion of L_{locked} of liquidity token is locked forever in an pseudo-user ux, such that $L(t)$ will never be 0 again and the market can only be bootstrapped once.

Consider allowing the initial liquidity provider to get the locked lp back after expiry.

Recommendation

Consider storing the initial liquidity provider's address and allowing the initial liquidity provider to `_burn(address(1), lpToReserve);` after `expiry` .

Status

Acknowledged

WP-I7: An attacker can front-run the initial liquidity provider and add imbalance PT/SYC liquidity to rug the liquidity provider

Informational

Issue Description

> UPDATE: This attack vector requires the first liquidity provider to call the router with minLpOut as 0 or call the PendleMarket.addLiquidity() directly.

> Thus, we downgraded it to `informational` . The issue was first discovered while we are examining the PendleMarket contract (which comes with no slippage control), and only while we try to get the exact numbers, we were aware of the rather strict bounds of the market exchange rate (constrained by the algo), which further lowered the severity of the issue.

> For a user using the router to add liquidity, this issue won't affect them, which in practice, renders this issue invalid.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/libraries/math/MarketMathCore.sol#L160-L178

```
160   if (market.totalLp == 0) {
161       lpToAccount = index.scyToAsset(scyDesired).subNoNeg(MINIMUM_LIQUIDITY);
162       lpToReserve = MINIMUM_LIQUIDITY;
163       scyUsed = scyDesired;
164       ptUsed = ptDesired;
165   } else {
166       int256 netLpByPt = (ptDesired * market.totalLp) / market.totalPt;
167       int256 netLpByScy = (scyDesired * market.totalLp) / market.totalScy;
168       if (netLpByPt < netLpByScy) {
169           lpToAccount = netLpByPt;
170           ptUsed = ptDesired;
171           scyUsed = (market.totalScy * lpToAccount) / market.totalLp;
172       } else {
173           lpToAccount = netLpByScy;
```

```
174             scyUsed = scyDesired;
175             ptUsed = (market.totalPt * lpToAccount) / market.totalLp;
176         }
177     }
```

In the current implementation, the initial liquidity provider can add arbitrary amounts of PT/SYC (as long as `exchangeRate` >= 1), and the next liquidity provider must provide at the same ratio.

This makes it possible for the attacker to manipulate the `exchangeRate` by adding liquidity using a large amount of PT and a small amount of SCY.

https://github.com/pendle-finance/pendle-core-internal-v2/blob/89d401ab42226b6155f339796471ed3074babc42/contracts/core/actions/base/ActionSCYAndPTBase.sol#L44-L58

```
44  MarketState memory state = IPMarket(market).readState(false);
45  (, netLpOut, scyUsed, ptUsed) = state.addLiquidity(
46      SCY.newIndex(),
47      scyDesired,
48      ptDesired,
49      false
50  );
51
52  // early-check
53  require(netLpOut >= minLpOut, "insufficient lp out");
54
55  if (doPull) {
56      IERC20(SCY).safeTransferFrom(msg.sender, market, scyUsed);
57      IERC20(PT).safeTransferFrom(msg.sender, market, ptUsed);
58  }
```

PoC

Given:

- `scalarRoot` = 1e18
- `initialAnchor` = 1.1e18

- `timeToExpiry` = 1 year
- Alice is the first liquidity provider;
- Bob is the attacker who monitors the txs in the mempool then frontrun and backrun the first liquidity provider's `addLiquidity` tx.

1. Alice (the first liquidity provider) call `addLiquidity` with:
   - `scyDesired` = 10,000e18
   - `ptDesired` = 10,000e18
   - `minLpOut` = 0
   - `doPull` = true
2. Bob front-run Alice's tx, call `addLiquidity` with:
   - `scyDesired` = 2,000
   - `ptDesired` = 48,000
   - `minLpOut` = 0
   - `doPull` = true

   Market State:
   - market.totalScy = 2,000
   - market.totalPt = 48,000
   - market.totalLp = 2,000
3. When Alice's tx was minted:
   - netLpByPt = (ptDesired * market.totalLp) / market.totalPt = 416666666666666666666;
   - netLpByScy = (scyDesired * market.totalLp) / market.totalScy = 1000000000000000000000;
   - scyUsed = 41666666666666666666
   - ptUsed = 100000000000000000000
   - lpToAccount = 41666666666666666666

   Market State:
   - market.totalScy = 41666666666666668666
   - market.totalPt = 100000000000000048000
   - market.totalLp = 41666666666666668666

   Since there are 24x more `PT` than `SCY` in the Market's reserves, the `exchangeRate` will deviate from the expected exchange rate.

   `lastLnImpliedRate` is now `1451613827240532992`.
4. Bob back-run Alice's tx and buy 100e18 (`100000000000000000000`) PT with 43e18 (`43913415919725870826`) SCY

Recommendation

Consider requiring the first liquidity provider to add with the equivalent value worth of PT and SCY tokens.

Status

ⓘ Acknowledged

WP-I9: If a newly added `rewardToken` happens to be the underlying yield token, users's deposit may be wrongfully distributed as rewards

Informational

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/SuperComposableYield/base-implementations/RewardManager.sol#L38-L65

```
38  function _updateRewardIndex() internal virtual {
39      if (lastRewardBlock == block.number) return;
40      lastRewardBlock = block.number;
41
42      _redeemExternalReward();
43
44      uint256 totalShares = _rewardSharesTotal();
45
46      address[] memory rewardTokens = _getRewardTokens();
47
48      for (uint256 i = 0; i < rewardTokens.length; ++i) {
49          address token = rewardTokens[i];
50
51          uint256 currentBalance = _selfBalance(token);
52          uint256 rewardAccrued = currentBalance - rewardState[token].lastBalance;
53
54          uint256 rewardIndex = rewardState[token].index;
55
56          if (rewardIndex == 0) rewardIndex = INITIAL_REWARD_INDEX;
57          if (totalShares != 0) rewardIndex += rewardAccrued.divDown(totalShares);
58
59          rewardState[token] = RewardState({
60              index: rewardIndex.Uint128(),
61              lastBalance: currentBalance.Uint128()
62          });
63      }
64  }
```

In the current implementation, some of the SCY implementations, eg `PendleAaveV3SCY` will get the `rewardTokens` list from `AaveRewardsController` in real time.

However, if a newly added `rewardToken` happens to be the underlying `yieldToken`, our contract will wrongfully take all the balance of underlying yield token as new rewards.

This issue is very unlikely to happen in practice.

Status

ⓘ Acknowledged

WP-G10: Reuse arithmetic results can save gas

Gas

Issue Description

https://github.com/pendle-finance/pendle-core-internal-v2/blob/19fb35e236c7916152b4a353ce09bd7c1903bc7a/contracts/libraries/math/MarketMathCore.sol#L416-L444

```
416  function setInitialLnImpliedRate(
417      MarketState memory market,
418      SCYIndex index,
419      int256 initialAnchor,
420      uint256 blockTime
421  ) internal pure {
422      /// ------------------------------------------------------------
423      /// CHECKS
424      /// ------------------------------------------------------------
425      require(blockTime < market.expiry, "market expired");
426
427      /// ------------------------------------------------------------
428      /// MATH
429      /// ------------------------------------------------------------
430      int256 totalAsset = index.scyToAsset(market.totalScy);
431      uint256 timeToExpiry = market.expiry - blockTime;
432      int256 rateScalar = _getRateScalar(market, timeToExpiry);
433
434      /// ------------------------------------------------------------
435      /// WRITE
436      /// ------------------------------------------------------------
437      market.lastLnImpliedRate = _getLnImpliedRate(
438          market.totalPt,
439          totalAsset,
440          rateScalar,
441          initialAnchor,
442          market.expiry - blockTime
443      );
444  }
```

`market.expiry - blockTime` at L442 is calculated before at L431, since it's a checked arithmetic operation with memory variables, resue the result instead of doing the arithmetic operation again can save gas.

Recommendation

Change to:

```
416  function setInitialLnImpliedRate(
417      MarketState memory market,
418      SCYIndex index,
419      int256 initialAnchor,
420      uint256 blockTime
421  ) internal pure {
422      /// ------------------------------------------------------------
423      /// CHECKS
424      /// ------------------------------------------------------------
425      require(blockTime < market.expiry, "market expired");
426
427      /// ------------------------------------------------------------
428      /// MATH
429      /// ------------------------------------------------------------
430      int256 totalAsset = index.scyToAsset(market.totalScy);
431      uint256 timeToExpiry = market.expiry - blockTime;
432      int256 rateScalar = _getRateScalar(market, timeToExpiry);
433
434      /// ------------------------------------------------------------
435      /// WRITE
436      /// ------------------------------------------------------------
437      market.lastLnImpliedRate = _getLnImpliedRate(
438          market.totalPt,
439          totalAsset,
440          rateScalar,
441          initialAnchor,
442          timeToExpiry
443      );
444  }
```

Status

✓ Fixed

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.